Marques and Ricardo *EURASIP Journal on Wireless Communications and Networking* (2017) 2017:37
DOI 10.1186/s13638-017-0821-7

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

Synchronization of application-driven WSN

Bruno Marques[1,2*] and Manuel Ricardo[2]

Abstract

The growth of wireless sensor networks (WSN) has resulted in part from requirements for connecting sensors and advances in radio technologies. WSN nodes may be required to save energy and therefore wake up and sleep in a synchronized way. In this paper, we propose an application-driven WSN node synchronization mechanism which, by making use of cross-layer information such as application ID and duty cycle, and by using the *exponentially weighted moving average* (EWMA) technique, enables nodes to wake up and sleep without losing synchronization. The results obtained confirm that this mechanism maintains the nodes in a mesh network synchronized according to the applications they run, while maintaining a high packet reception ratio.

Keywords: Wireless sensor network (WSN), ContikiRPL, Nodes synchronization, EWMA

1 Introduction

Recently, there has been an increasing trend towards the deployment of WSN, where a large number of tiny devices interacting with their environments may be inter-networked and accessible through the Internet. For that purpose, several communication protocols have been defined making use of the IEEE 802.15.4 Physical and MAC layers [1]. The 6LoWPAN Network Layer adaptation protocol [2] is also used to enable the interconnection between low-power devices and the IP network. Since its release, the design of routing protocols became increasingly important [3] and RPL [4] emerged as the IETF proposed standard protocol for IPv6-based multi-hop WSN.

WSNs are constituted by sensor devices equipped with their own local clock for internal operations [5]. Events related to them, which include sensing, processing, and communication, are normally associated to timing information. In the particular case of WSNs, there are challenges and factors related to node synchronization, which include low-cost clocks, effects of wireless communication, and node failures. Moreover, WSNs are distributed and their nodes have multiple hardware and software constraints such as low processing power, low memory and storage capabilities, and low-power consumption. These characteristics make time synchronization an important part of communication in WSNs, and synchronization protocols are required.

In [6], we presented a new paradigm, the *application-driven WSN* paradigm, as a cross-layer solution aimed to help reducing the energy consumed by a network of sensors executing a set of applications. This paradigm assumes that each application defines its own network and set of nodes so that the exchange of information can be confined to the nodes associated to the application. The nodes share information about the applications they run and their duty cycles.

In [7], we proposed an extension to the *RPL* routing protocol, the *RPL-BMARQ*, with the purpose of making the network aware of the traffic generated by applications. The main objective of this extension was to construct directed acyclic graphs (DAGs), by using information shared by the application and network layers, allowing the nodes to select parents by considering the applications they run. In that work, we characterized the *energy consumption* and the *energy gain* and also *end-to-end delay* and *fairness*. For evaluation purposes, we selected four scenarios in which all the nodes joined the network at same time and performed simulations considering *regular RPL* and *RPL-BMARQ*. Later, we started to study the behavior of *RPL-BMARQ* considering that the nodes would not join simultaneously the WSN. At this end,

*Correspondence: bmarq@estgv.ipv.pt
[1]Departamento Engenharia Eletrotécnica, Escola Superior de Tecnologia e Gestão, Instituto Superior Politécnico de Viseu, Viseu, Portugal
[2]INESC TEC, Faculdade de Engenharia, Universidade do Porto, Porto, Portugal

we presented a draft of a possible node synchronization mechanism, and estimated the energy gains introduced by *RPL-BMARQ*.

In this paper, we consider a more realistic situation in which the nodes join the WSN at a non-predictable and different time. At this end, the sensor nodes must share some kind of time reference which allow them to be synchronized with respect to the life cycle of the applications they run. Therefore, in this paper, we propose a novel *synchronization mechanism* for *RPL-BMARQ*, which will help the nodes to wake up and to go asleep in a synchronized manner so that they can successfully send, receive, and forward packets, maintaining the energy consumption low.

The major contribution of this paper is then a mechanism for WSN which synchronizes the sensor nodes with respect to the applications life cycles they run, enabling these nodes to wake up and to go asleep in synchronism, while maintaining a packet reception ratio high. The novelty of our contribution comes from (1) the adaptation of the well-known *exponentially weighted moving average* technique to wireless mesh network scenarios and (2) using this mechanism to control the behavior of sensor nodes so that they become synchronized in relevant time instants which are defined by their application duty cycles.

The paper is organized in 6 sections. Section 2 presents the related work. Section 3 describes the application-driven WSN concept. Section 4 describes the rationale of the contribution–the synchronization mechanism. Section 5 evaluates the proposed mechanism, describes the methodology adopted for its validation, and discusses the results obtained. Finally, Section 6 draws the conclusions and presents future work.

2 Related work

In this section, we present and discuss related work in three main areas: *time synchronization*, *wake-up mechanisms for WSN*, and 6LowPAN/IPv6/ RPL evaluations.

2.1 Time synchronization

WSNs are constituted by sensor devices equipped with their own local clock for internal operations [5]. Events related to them, which include sensing, processing, and communication, are normally associated to timing information. In the particular case of WSN, there are many challenges related to time synchronization because these networks are distributed by nature and because of the constraints of the sensor nodes in terms of hardware and of software.

Akyildiz and Vuran [5] state that in order for the nodes to synchronize, they must exchange information about their clocks and use this information to synchronize their local clocks. By using wireless communications, WSNs create challenges for synchronization that result from the error-prone communication nature of the wireless channel which may cause packet losses due to low signal-to-noise plus interference ratios, or highly and variant non-deterministic delays caused by MAC access and packet retransmissions. These factors affect also the time synchronization messages. Therefore, some nodes may be unsynchronized. On the other hand, synchronization messages sent by nodes may lead other nodes to adapt to their unsynchronized local clocks. As a consequence, the network may be partitioned into different areas with different time that prevents synchronization of the entire network. Also, the wireless channel may introduce asymmetric delays between two nodes, which is important for synchronization because some synchronization solutions depend on consecutive message exchange and round-trip-time delays. Therefore, robust synchronization methods are needed.

We start to identify some factors that influence the synchronization of the nodes and that should be considered in the design of time synchronization mechanisms for WSN. As the *Network Time Protocol* (NTP) protocol [8] is a synchronization protocol normally used in IP networks, we provide an overview of it and also describe synchronization protocols for WSN related to our work.

2.1.1 Factors influencing time synchronization

According to [9], some of the factors influencing time synchronization in large systems constituted for example by personals computers, also apply to sensor networks, where temperature, phase noise, frequency noise, asymmetric delays, clock glitches, and sensors constraints are examples of these factors. In the case of the *temperature*, since sensor nodes are deployed in various places, temperature variations throughout the day may cause the clock to speed up or slow down. In the case of the *phase noise* factor, some of its causes are due to fluctuations in the hardware interface, response variation of the operating system to interrupts, and jitter in the network. The *frequency noise* results from the instability of the clock crystal. In the *asymmetric delay* factor, the delay of the path from one node to another node may be different from the return path which may result in an asymmetric delay and may cause an offset to the clock, which may go undetected. *Clock glitches* are abrupt jumps in time, caused by hardware or software anomalies such as frequency and time steps. Finally, WSN nodes are constrained by nature because of limited resources (e.g., low in energy consumption, low in processing power, or low in memory).

The transmission and reception of packets are the factors that cause more energy consumption in a sensor node. Therefore, a time synchronization protocol for sensor networks should help overcome the synchronization

problems introduced by the factors described above, avoid frequent message exchanges, and be self-configurable.

2.1.2 Network Time Protocol

The NTP [8] is the synchronization protocol more often used in the Internet. This protocol includes several synchronization mechanisms that have been also adapted for developed WSN synchronization protocols. Reference-broadcast synchronization (RBS) [10], timing-sync protocol for sensor networks (TPSN) [11], lightweight tree-based synchronization (LTS) [12], and TSync [13] are some examples of these protocols. NTP is used to adjust the clock of each network node. This synchronization is achieved by using a hierarchical structure of time servers. The root node is synchronized with the *Coordinated Universal Time* (UTC). In each level of this hierarchy, the time server nodes synchronize the clocks of their subnetwork peers. NTP uses a two-way handshake between two nodes to estimate the delay between these nodes and compute the relative offset accordingly (see Fig. 1, where node *s* will synchronize himself with node *r*). However, NTP assumes that the transmission delay between two nodes is the same in both directions. This is reasonable for the Internet, but some of the characteristics of WSN make this assumption inadequate. NTP is useful to discipline the oscillators of the sensor nodes, but using it to connect to time servers may be impossible because of sensor node failures, which are frequent in WSN. Using a single clock reference to synchronize all the nodes could be a problem due to the variations in network delays. Moreover, NTP requires intensive computing, requires a precise time server to synchronize the nodes, and does not consider the energy the nodes may spent to synchronize their clocks. All these problems may cause NTP to inaccurately measure delays and inaccurately estimate clock offsets.

2.1.3 Synchronization protocols for WSN

WSN poses unique challenges in the design of synchronization protocols, which calls for specific synchronization solutions. An example is the effect of the broadcast wireless channel. However, wireless communication introduce random delays between two nodes. Let us consider Fig. 2, which represents a handshake scheme. The delay between two nodes is characterized by four components: (i) the sending delay (t_{send}), (ii) the access delay (t_{acc}), (iii) the propagation time (t_{prop}), and (iv) the receiving delay (t_{recv}).

The handshake is initiated when node *s* issues a SYNC packet with the timestamp t_1^s. Between the time the synchronization protocol issues the synchronization command and the time during which the SYNC packet is prepared, there is a delay, t_{send}, resulting from the combination of operating system delays and transceiver delays on the node's hardware; t_{acc} corresponds to the additional delay introduced by the wireless channel after the packet has been prepared and transferred to the transceiver. This delay depends on the MAC protocol when the node waits for accessing the channel; as an example, MAC protocols using CSMA introduce a significant amount of access delay when the channel is very occupied. t_{prop} is the amount of time needed to transmit a SYNC packet to a receiver. Finally, t_{recv} is the time required for the transceiver of the receiver node *r* to receive the packet and process it. The transmission delay, t_{tx}, is a component of the receiving delay, which is important and characterized by the time needed for the SYNC packet to be completely received (see Fig. 2); it depends on the transmission rate and on the length of the SYNC packet. These components contribute to the overall communication delay, also referred as *critical path*. Delays are non-deterministic and create challenges when estimating clock offsets using the NTP's methods. Most of the synchronization protocols for WSN tend to minimize the effects of these delays, which are random. In what follows, four related existing synchronization protocols are described.

Reference-broadcast synchronization: In Fig. 3, a *sender-receiver handshake scheme* is shown which introduces a significant amount of non-deterministic delay [10]. The RBS protocol tries to minimize the overall communication delay in the synchronization process. It eliminates the effect of the broadcast node. Instead of synchronizing the receiver with the sender, RBS synchronizes

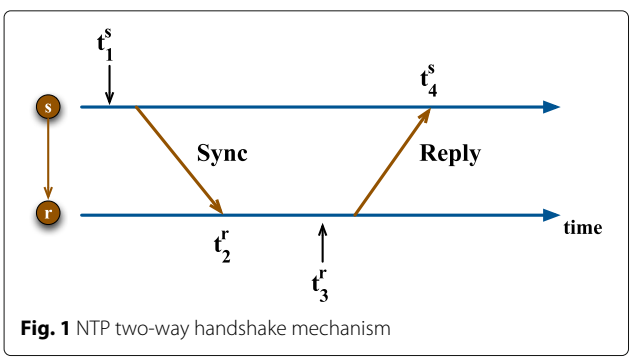

Fig. 1 NTP two-way handshake mechanism

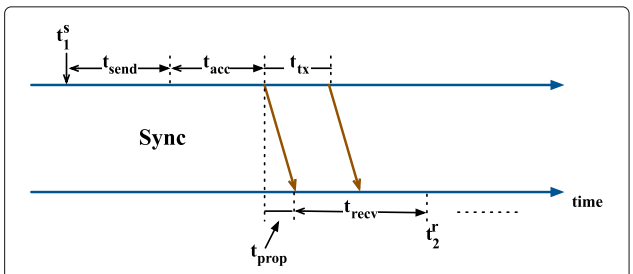

Fig. 2 Synchronization delay between two nodes

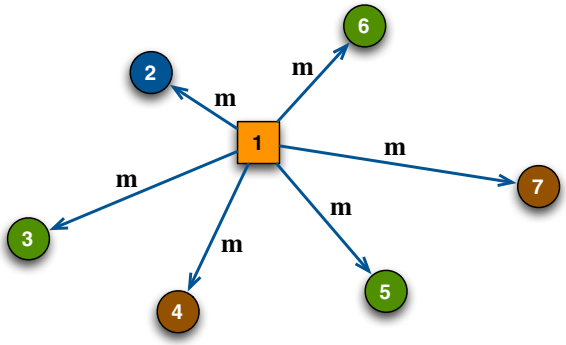

Fig. 3 Reference broadcast. Node 1 broadcasts *m* messages which are used by the other nodes for synchronization purposes

a set of receivers that are within the reference transmission of a sender. Considering that propagation times are negligible on wireless channels, as soon as a packet is transmitted, it is received at all sender's neighbors almost at the same time. Therefore, the synchronization may be improved if only the receivers are synchronized. As shown in Fig. 3, node 1 broadcasts *m* reference packets and each one of the receivers, within its broadcast range, records the time the packets are received. Then, the receiver nodes communicate with each other to estimate the offsets, just like the traditional synchronization. Figure 4a shows the *critical path* for traditional synchronization. Sending delays and the access delays should be accurately estimated to improve the synchronization. Reference-broadcast synchronization does not involve node 1 in the synchronization; only the receivers (nodes 2, 3, 4, 5, 6, and 7) synchronize among themselves based on a reference-broadcast message from node 1. As shown in Fig. 4b, this reduces the *critical path* duration. In fact, the possible origin of uncertainty in RBS is the time between when a broadcast packet is received and when it is completely processed.

A method used to determine with efficiency the clock offset of each node in relation to its neighbors is the *receiver-receiver synchronization* method. By exchanging messages with each neighbor, a node fills a table consisting of relative offsets. Therefore, the main goal of RBS is not to correct the clocks of the nodes but, every time a packet is received, to translate its timestamp to the node's clock using the relative offset information. This synchronization method can only provide synchronization in a broadcast area. In order to provide multi-hop synchronization, RBS uses nodes that receive two or more different reference-broadcast messages. These nodes are

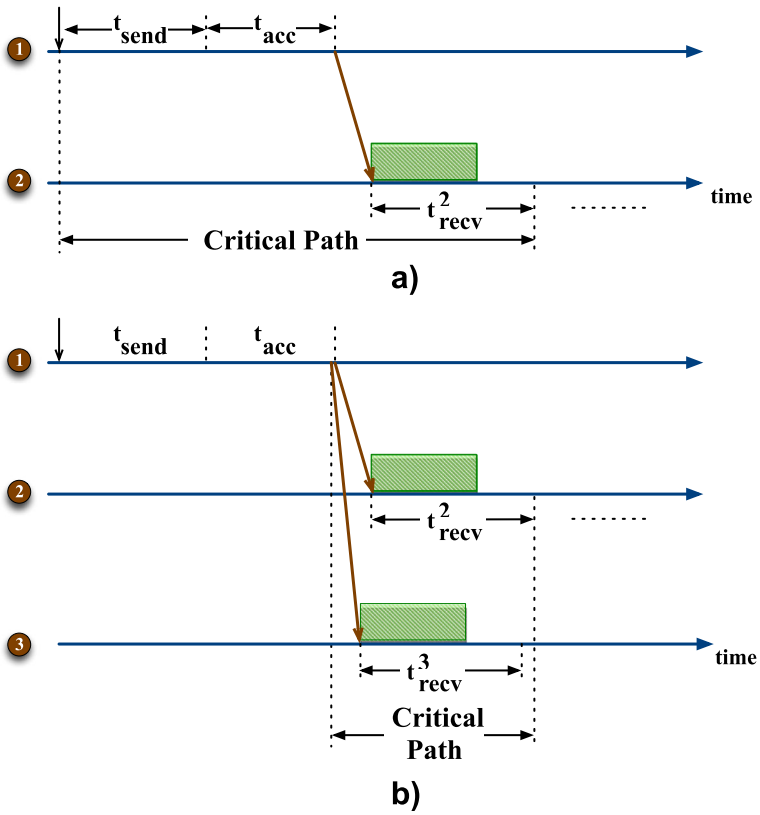

Fig. 4 Critical paths for: **a** a pair of nodes and **b** RBS

called *translation nodes*, and they are used to translate the time between different broadcast domains (see Fig. 5). As it can be observed, nodes A, B, and C are, respectively, the transmitter, the receiver, and the translation nodes. The transmitter node broadcasts its timing messages, the receiver node receives those messages, and then the nodes synchronize with each other.

Timing-sync protocol for sensor networks [11]: TPSN uses some of the NTP concepts: it uses a hierarchical structure to synchronize the entire WSN to a single time server. TPSN uses the root node to synchronize all or part of the network, consisting of two phases: (1) the *discovery phase*, where the structure of TPSN is built, starting from the root node and (2) the *synchronization phase*, where pairwise synchronization is performed across the network. In (1), the root node is assigned to level 0 and the other nodes in the network are assigned to levels according to their distance to the root node (see Fig. 6).

Firstly, the root node starts to construct the TPSN structure. To this end, it broadcasts a special packet called *level_discovery packet*. In this structure, the first level is assigned to the number 0, which is the level of the root node. The other nodes that receive this packet are the nodes that belong to level 1. Afterward, these nodes broadcast their *level_discovery* packet. Then, the neighbor nodes receiving those packets are labeled as level 2 nodes, and the process is repeated until all the nodes in the network are assigned to a level.

In (2), each node in the structure is synchronized with a node from a higher level. The root node sends another packet (the *time_sync* packet) which initializes the time synchronization process. Afterwards, the nodes in the next level start to synchronize with the root node by sending a *synchronization_pulse* to it, as shown in Fig. 7. In order to avoid collisions with other nodes, each node in level 1 waits for a random amount of time before transmitting the *time_sync* packet. After the reception of this packet, the root node sends an acknowledgment back to finish the synchronization process. In this way, nodes belonging to level 1 of the structure are synchronized with root node (see Fig. 7). This *time_sync* packet also serves as a *synchronization_pulse* to level 2 nodes. Upon a reception of this packet from a node in level 1, the nodes in level 2 wait for a random amount of time for the level 1 nodes to finish their synchronization. Then, they initialize the synchronization process by transmitting a *synchronization_pulse*. Acting like the root node in level 0, a level 1 node sends back an acknowledgment, the process continues until all the nodes at different levels are synchronized, and the entire network becomes synchronized.

In TPSN, the receiver synchronizes with the local clock of the sender according to the two-way message handshake, as shown in Fig. 7. For this reason, TPSN is based on a *sender-receiver synchronization method*. Hierarchical structures created by TPSN are similar to the structures created by NTP. Like in NTP, nodes may fail causing nodes to become unsynchronized. Also, nodes mobility can make the hierarchy useless, as they may move out of their levels. Therefore, nodes at level n cannot synchronize with nodes at level $n-1$, without requiring additional and periodical synchronization.

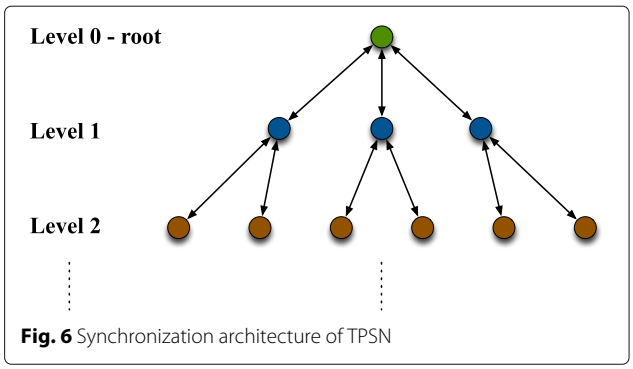

Fig. 6 Synchronization architecture of TPSN

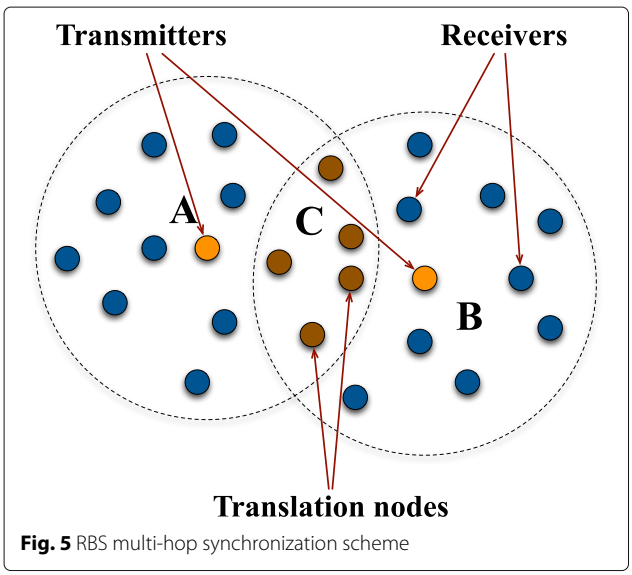

Fig. 5 RBS multi-hop synchronization scheme

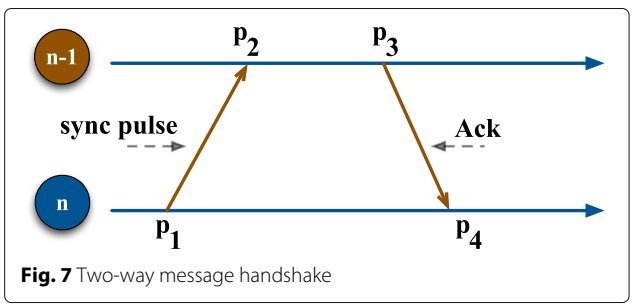

Fig. 7 Two-way message handshake

Lightweight tree-based synchronization [12]: LTS is similar to TPSN and follows two design approaches: centralized and distributed. The centralized design is based on the construction of a tree such that each node is synchronized to the root node. After the tree is constructed, the root initiates pairwise synchronization with its children nodes and the synchronization is propagated along the tree to the leaf nodes.

In the distributed design, LTS does not rely on the construction of a tree and synchronization can be initiated by any node in the network. Each node performs synchronization only when it has a packet to send. Therefore, each node is informed about its distance (in number of hops) to the reference node for synchronization, the desired accuracy, the clock drift, and a record of the time that has passed since they were synchronized. Then, the nodes adjust its synchronization rate accordingly. Nodes farther apart from the reference node perform synchronization more frequently because synchronization accuracy is inversely proportional to distance.

In general, LTS is based on message exchanges between two nodes to estimate the clock drift between their clocks. This synchronization scheme is named *pairwise synchronization scheme*, and it is extended for multi-hop synchronization.

In contrast to our centralized and asynchronous proposed synchronization mechanism, in [14], a synchronous protocol is proposed that provides a distributed strategy which guarantees convergence for any undirected connected communication graph. This strategy tries to control the nominal clock period and the clock offset based on the information received from neighbor nodes in order to achieve synchronization. Moreover, when an underlying communication graph is known, the authors purpose an optimal design strategy which can be used to study the effect of noise and external disturbances on the steady-state performance.

There are additional works proposing and analyzing time synchronization mechanisms. In [15], the authors use factor-graph methods for network clock estimation and propose two methods for message passing: *belief propagation (BP)* and *mean field (MF)*.

In [16], two joint synchronization and localization algorithms in both line of seeing (LOS) and in non-line of seeing (NLOS) environments are proposed. They applied Taylor expansions in order to represent factor graphs in closed Gaussian forms where the means and variances of beliefs of node estimates can be easily obtained by simple arithmetic operations.

In [17], the authors propose a global clock synchronization method by adopting a *packet-based* synchronization scheme. The proposed distributed algorithm requires communications only between neighboring sensors and computes a set of marginal distributions using the BP message passing [18]. The authors have observed that the state of clock offset at any sensor depends directly only on its neighboring sensors and that the algorithm synchronizes clocks with a consistent reference value instead of adjusting clocks to an average value.

In [19], WSN time synchronization follows two strategies: (i) *maximum time synchronization* (MTS) to simultaneously synchronize the skew and offset of each node when the communication delay is negligible and (ii) a *weighted maximum time synchronization* (WMTS) when the communication delay between the nodes is random. In contrast to our work, in which we synchronize a virtual clock, these authors attempt to synchronize the clock skew, in order to obtain acceptable synchronization accuracies. The main idea of MTS and WMTS is to drive all clocks to the maximum value among the network. In [19], random communication delays with normal distribution are considered, while we validated our solutions against Gaussian and exponentially distributed delays. This solution can be classified as distributed and asynchronous algorithm, whereas ours can be classified as centralized and asynchronous.

Since synchronization is a widely studied topic, in [20], a survey of clock synchronization for wireless sensor networks is published.

2.2 Wake-up mechanisms for WSN

WSN are energy limited so typically the nodes cannot keep radios active all the time, having to sleep and to wake up periodically [21]. Addressing this issue, there have been proposed several MAC protocols which were categorized as *synchronous* or *asynchronous* MAC protocols. Although asynchronous protocols are simpler, they tend to consume more energy. But in WSN, where energy must be saved, a different approach may be used. One possibility is to use synchronous methods. Using these protocols, some techniques are adopted to increase the nodes lifetime: (i) *duty cycling* and (ii) *scheduled rendezvous*.

Duty cycling: This is one mechanism widely used for energy-efficient MAC protocols in WSN. A MAC protocol that implements duty cycling uses appropriate sleep/wake-up mechanisms to conserve energy, and in [22], it is demonstrated that when sensor nodes remain in the sleep mode, they consume less energy than when in the idle mode. When there is no need for communication, the radio is put to sleep and, although applying duty cycling energy is conserved, it has some disadvantages. Putting sensors into sleep mode makes it difficult to the all network to function or at least certain part of it. As showed in [23], a few issues are needed to overcome such as deciding when to switch a device to low power mode or deciding "for how long should a device remain in the low power mode?" To solve these issues, efficient and flexible

duty cycling techniques have been proposed. The S-MAC [24] and the T-MAC [25] protocols are examples of them. These protocols transmit a *SYNC* packet to notify neighbors about their schedule and to synchronize the clocks of all nodes in the network. The method only compensates for clock offset and does not consider clock drift [21]. Moreover, the knowledgement of traffic patterns can also help to take decisions about waking up. This method is known as adaptive duty cycling. S-MAC [24] is one of the major energy-efficient MAC protocols that efficiently exploits the idea of adaptive duty cycling. It uses a periodic sleep-wake-up mechanism in order to lower power consumption. If a node has no packet to receive, it can waste a large amount of energy by just listening to the channel. Consequently, a node can save a significant amount of energy if it simply goes to sleep mode by switching off its radios [22]. T-MAC is an improvement over S-MAC duty cycling. In the T-MAC, listening period ends when no event has occurred for a time threshold *TA*. Though it improves on S-MAC, T-MAC has the disadvantage that it can face an early sleeping problem where a node can go to sleep even though its neighbor may still have messages for it. Synchronization is also an issue in duty cycling MAC protocols. In [26], that synchronous MACs such as S-MAC have low energy consumption for sending packets but are complicated due to the need of synchronization is argued. Conversely, asynchronous MACs, for example WiseMAC [27], is very simple, but it spends much energy in finding the neighbor's wake-up time. Moreover, synchronous methods can be characterized as one-way methods. Usually, the senders broadcast a reference message and receivers, upon the reception of the message, record the arrival time by their own clocks, and exchange this information among each other to compensate clock offset between them. In [21], a synchronous method is proposed in which clocks in the all network are not modified. Instead, the nodes are synchronized with their own clocks. Since the periodic broadcast event in the network is the same, although they have different measurement results for this period by their own clock unit independently, they are able to interact with each other at the same physical time. Without complicating the estimation process and without modifying the clock of a node, this synchronization method becomes simpler and more energy-efficient than the traditional synchronization one-way method.

Scheduled rendezvous: This type of MAC protocol requires a prescheduled *rendezvous* time at which neighboring nodes wake up simultaneously. In this method, a node wakes up periodically and sleeps until the next rendezvous time. A *scheduled rendezvous* scheme is shown in Fig. 8 [22].

The advantage of this scheme is that when a node is awake, it is guaranteed that all its neighbors are awake as well. Consequently, it is easier to send/receive packets. Broadcasting a message to all neighbors is also simpler in scheduled rendezvous schemes. RI-MAC [28] is a *receiver-initiated* asynchronous duty cycle MAC protocol for WSN. It uses a receiver-initiated data transmission in order to proficiently operate over a wide range of traffic loads. It attempts to minimize the time a sender and the receiver occupy the medium to find a rendezvous time for exchanging data, while still decoupling the sender and receiver's duty cycle schedules. A disadvantage of such MAC protocol is the requirement to maintain strict synchronization because clock drifting may deeply affect the rendezvous time.

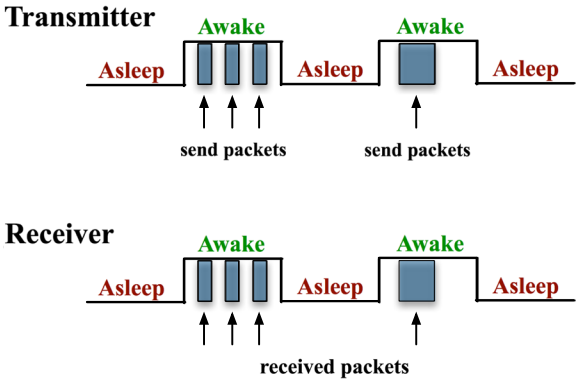

Fig. 8 A scheduled rendezvous scheme

2.3 6LowPAN/IPv6/RPL evaluations

In [29], a cross-layering design for RPL which provides enhanced link estimation and efficient management of neighbor tables is proposed. They used AMI as a case study and employed the *Cooja* emulator to evaluate their proposal. The authors analyzed RPL together with the underlying *X-MAC* and *ContikiMAC* and *Nullrdc* protocols from the reliability stand point by considering *packet loss*, *end-to-end delay*, and *energy consumption* and have implement a testbed using *ContikiOS* to validate their work. In [30], the performance of RPL used for multi-sink WSNs considering the *hop-count* and/or *ETX*, packet loss, and energy consumption metrics is evaluated. To validate the results from the performed simulations, the authors performed on a real-life testbed the same tests.

In both works [29, 30], the authors considered networks supporting single application where the nodes join the network at same time. The performance metrics they considered were packet loss, end-to-end packet delay, and energy consumption. In our work, the networks deployed support multiple applications and the sensor nodes join the network at different times what demands a node synchronization mechanism. In order to characterize the performance of our system, we used a set o metrics including

end-to-end packet delay, energy consumption, query success ratio, and fairness Index. Query success ratio (QSR) quantifies the success of a sink node with respect to the reception of all the expected reply packets upon the transmission of a query packet; this metric allows us to see if all the nodes receive the query packets and if they reply back. Therefore, it is easy to verify packet loss. The fairness index metric is used to investigate if the nodes have the same opportunity to reply back to the sink. We used ContikiOS/Cooja for the simulations and validated our work by implementing two testbeds.

3 Application-driven WSN

The *application-driven WSN* paradigm [6] assumes that each application defines its own network and set of nodes so that the exchange of information can be confined to the nodes associated to the application. The nodes share information about the applications they run and their duty cycles, and nodes are put asleep when there is no activity related to their applications. When nodes receive a query packet, they know exactly when they must wake up on the next period. The nodes alternate between wake and sleep states, and the amount of time spent in each phase is determined by the application duty cycle. When the wake-up time expires, the node switches to the sleep state, waking up again by the time computed by the synchronization mechanism proposed in Section 4 of this paper.

We assume that every node can participate in route discovery and packet forwarding. However, the nodes forwarding a given type of data will be primarily selected from the set of nodes running the same application to which the data is associated. For that purpose, each query packet includes information about the associated application (APPID), which is known by the nodes running that application. Our routing scheme tries to insure that data of an application is relayed mainly by the nodes running that application. When the sink node queries the other nodes running the same application, routing paths follow the directed acyclic graph (DAG) created. This DAG is created and maintained by a change to the RPL protocol scheme which uses mainly the nodes running that application; the nodes not associated to this application will not participate in the routing process, in a first attempt. In our proposal, the subset of nodes running the same application forms a "subnetwork" with multi-hop connectivity and application packets carry out also information about the application duty cycle (T_{CYCLE} and T_{ON}) that is used to create and maintain the DAGs in which not only the nodes running the same application but also the nodes having the same application duty cycle can be "grouped". Figure 9a shows a network topology supporting two different applications. Figure 9b shows the DAG created with standard RPL, and Fig. 9c shows the DAG created by our proposed solution. The wake-up mechanism is based on the applications time cycle information (T_{CYCLE} and T_{ON}), carried by every application query sent by the sink nodes. When a node receives a query packet, it knows exactly when it must wake up on the next period.

4 Application-driven synchronization mechanism

According to our application-driven concept, synchronization is achieved between the nodes that run the same applications or between the nodes that have the same application duty cycle, by considering their duty cycles. Therefore, the first time a node joins the network, it waits for an application query packet to adjust its virtual clock to the time carried by the query packet. We realize that this

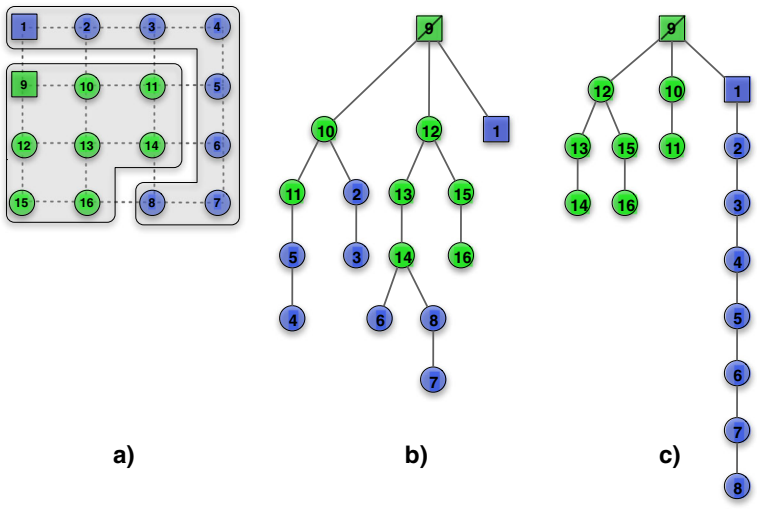

Fig. 9 Application-driven WSN concept. **a** Network topology. **b** RPL DAG. **c** BMARQ-RPL DAG

corresponds to setting the time's nodes to a value which does not consider network delays but, as demonstrated in the paper, this has no impact on our synchronization mechanism as the nodes dynamically adjust their sleeping offset (see $\beta \cdot |\delta_{k,n}|$ component in Eq. 2) and wake up and sleep almost at the same time during the network lifetime. As such, the synchronization algorithm takes advantage of the application query packets that are sent by the sink nodes once in every application duty cycle to maintain the sensor nodes synchronized. A network may support several applications but only the nodes running the same application or having the same duty cycle will synchronize between them. Therefore, a network supporting different applications may have different sets of nodes with different synchronizations and still be fully functional. Without having to send or to receive other type of packets for synchronization purposes, the nodes will rely only on the queries received to synchronize. In fact, this algorithm is centralized on a sink node, but its design is simple and adequate for our purposes. A distributed design would be more complex and imply the use of other types of packets for synchronization, often broadcasted through the network, which would have impact in energy consumption due to packet transmission and reception costs.

It is unlikely that all the sensor nodes would join a network at the same time. Having the nodes active all the time would deplete their batteries, so the nodes have to go sleep and to wake up periodically. All the nodes have to be awake almost at the same times in order to receive sink queries and to forward them to the other nodes. As a result, the nodes must be synchronized according to the application cycle they run. In order to synchronize all the nodes in the network, our proposed synchronization mechanism uses a synchronous method which includes two phases: *the synchronization setup phase* and *the synchronization maintenance phase*, described below.

4.1 The synchronization setup phase

When a sensor node joins the network, it remains in the wake state and waits for the reception of its first query packet sent by the sink node and forwarded by other nodes. Upon its reception, the node adjusts a virtual clock to the timestamp carried by the query. As it can be observed from Fig. 10, the query packet sent by a sensor *node n* towards a sensor *node* $n+1$ is the same query packet that *node n* received from the sink node. The timestamp carried by the query is extracted from the query packet. This phase is used to readjust the virtual clock; the periodicity of this readjustment depends on how often the nodes have to readjust their virtual clock. It is known that this phase corresponds to setting the time's nodes to a value which does not consider network delays.

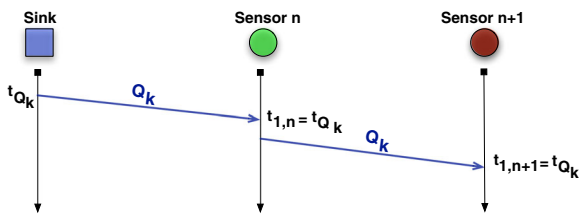

Fig. 10 Synchronization setup phase

In the example shown in Fig. 11a, sink node A issues a query ($Q_{k,j}$) before sink node B. The query packet is disseminated through the network as expected using the *RPL-BMARQ* routing solution [7]. Sensor nodes C and D, which run this sink's application, set their virtual clock to the timestamp carried out by the packet. Sensor node E, not running this application, also sets his virtual clock to the timestamp carried out by the query packet since it is the first query it receives. The same query packet ($Q_{k,j}$) is then forwarded to the other sensor nodes (nodes G, H, and K) which will also set their virtual clock to the same timestamp. Sensor node E will not forward the query packet $Q_{k,j}$ since it does not run this application and does not have neighbors running it. Similarly, node F, upon the first query packet ($Q_{k,i}$) reception from sink node B, and because it runs the same application, adjusts it virtual clock to the time carried out by the sink B query packet. As this sink has already adjusted its virtual clock using the sink A timestamp, sensor node F will have the same time as the other nodes. Again, the query $Q_{k,i}$ will be forwarded to the other sensor nodes (nodes I, J, and L) which will perform the same virtual clock adjustment. Figure 11b shows the same virtual clock adjustments, but in this case, it is sink node B that issues the first query packet and adjusts all the network node's virtual clocks.

4.2 The synchronization maintenance phase

Since all the nodes know the characteristics of the applications they run, after the reception of the first query packet, they expect to receive the second query packet by $t'_2 = t_1 + T_{\text{ON}} + T_{\text{OFF}}$. The time the nodes are sleeping (T_{OFF}) is defined as $T_{\text{Cycle}} - T_{\text{ON}}$ where T_{Cycle} is the application duty cycle time and T_{ON} is the time the nodes are awaked during each duty cycle. However, because network delays are variable, the nodes will receive this second query packet not in t'_2 but in t_2, as shown in Fig. 12. There is a difference between the expected value t'_2 and the real value t_2, $\delta_2 = t'_2 - t_2$. For example, if a node is expected to receive a query packet by $t'_2 = 100$ and receives it by $t_2 = 102$, then $\delta_2 = -2$. A negative value means that a query was received in delay, and a positive value means that the query was received in advance. Moreover, delays are the sum of all per-hop delays for each sensor

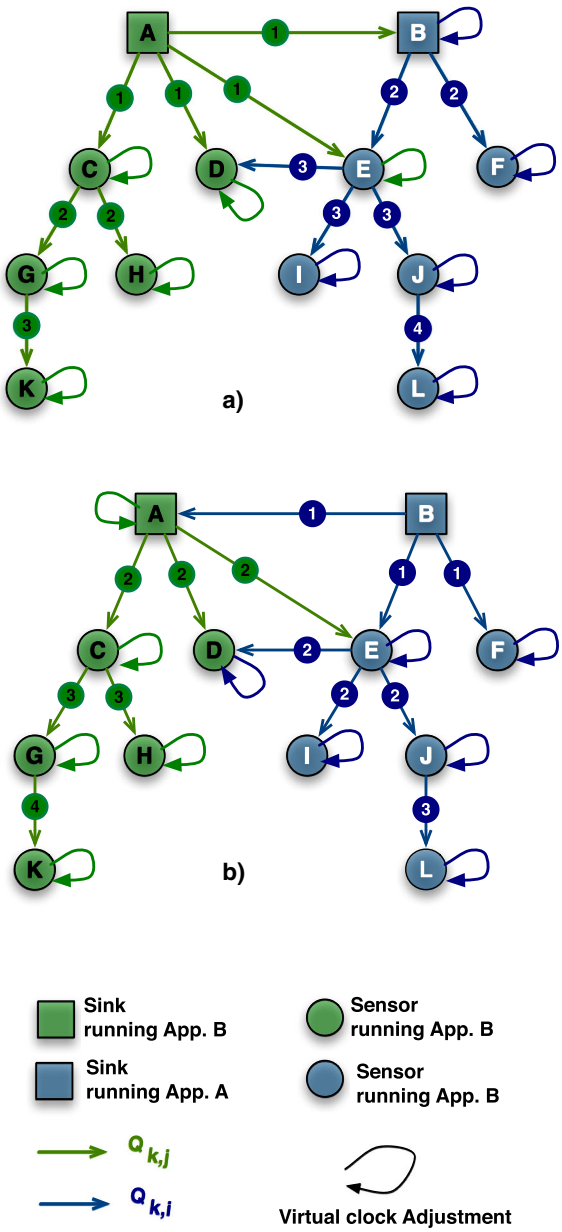

Fig. 11 Example of nodes synchronization

query packet reception and characterized by the sum of the processing and queueing delays in intermediate and destination sensor nodes, and the transmission delays and propagation delays in intermediate nodes. An in-depth characterization of these delays may be found in [31].

Our proposed mechanism estimates $\delta_{k,n}$ by using the exponentially weighted moving average (EWMA) technique (see Appendix). According to Fig. 12, the difference between the expected time to receive the next query and the time it is really received is computed by Eq. 1:

$$\begin{aligned} t'_{k,n} &= t_{k-1,n} + T_{\text{ON}} + T_{\text{OFF}} \\ \delta_{k,n} &= (1-\alpha)\cdot\delta_{k-1,n} + \alpha\cdot(t'_{k,n} - t_{k,n}); \\ 0 &< \alpha < 1 \end{aligned} \tag{1}$$

where $t'_{k,n}$ is the expected packet reception time and $t_{k,n}$ is the real packet reception time. $\delta_{k,n}$ is evaluated according to EWMA as in Eq. 3 with α reflecting the weight of the last observation. The $\delta_{k,n}$ value is dynamically adjusted every time a node wakes and receives a query packet, and it is used to control the time the node would sleep in the next cycle, given by Eq. 2.

$$T_{\text{Sleep}_{k,n}} = T_{\text{OFF}} - \beta\cdot|\delta_{k,n}| \tag{2}$$

In Eq. 2, the β factor is used to amplify the $\delta_{k,n}$ value to guarantee that the sensor node will wake some time before the next application cycle. $\beta\cdot|\delta_{k,n}|$ is the sleeping offset and represents the time the node will wake up before the start of the next application duty cycle. Algorithm 1 shows the pseudo-code of the application-driven synchronization mechanism with values given to α and β and to the virtual clock adjustment periodicity time (*adjust_periodicity_time*).

Algorithm 1: Pseudocode of the proposed synchronization mechanism

```
foreach (app.query_k received) do
    α = 0.125;
    β = 10;
    if (first(app.query)) then
        set_clock(query→ T_TX);
        adjust_periodicity_time = 3600 · 24 (eg. 24 hours);
        adjust_counter = adjust_periodicity_time / T_CYCLE;
    else
        if (app.query_id == node.app_id) then
            t'_{k,n} = t_{k−1,n} + T_ON + T_OFF;
            t_{k,n} = node.query_{T_RX};
            δ_{k,n} = (1 − α) · δ_{k−1,n} + α · (t'_{k,n} − t_{k,n});
            T_{Sleep_{k,n}} = app.T_OFF - β · |δ_{k,n}|;
            adjust_sleep_timer(T_{Sleep_{k,n}});
            adjust_counter = adjust_counter - 1;
        end
        if (adjust_counter == 0) then
            set_clock(query→ T_TX);
            adjust_counter = adjust_periodicity_time / T_CYCLE;
        end
    end
end
```

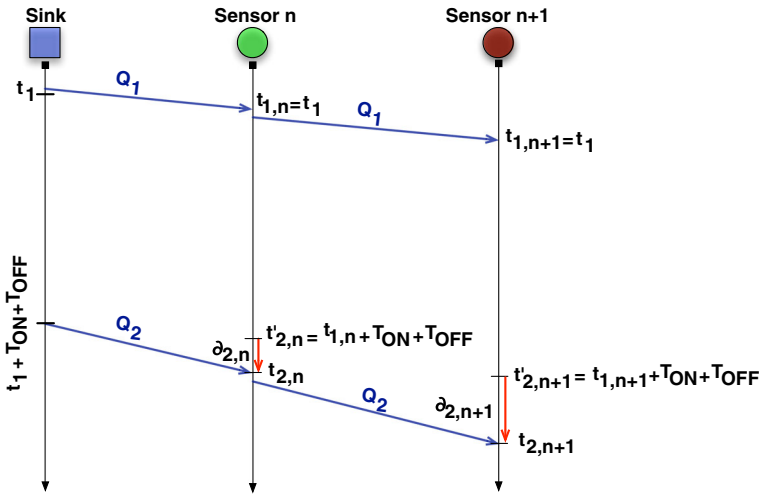

Fig. 12 Synchronization maintenance phase

5 Evaluation

In order to validate this mechanism, first we present a study on how the nodes can maintain their synchronization by estimating and evaluating the parameters presented in Eqs. 1 and 2, which corresponds to investigate in depth the synchronization maintenance phase. We also present and discuss results from the proposed synchronization mechanism using different values for α and β parameters, and query success ratio (QSR) results from simulations, and finally present and discuss some of the results obtained from two real testbeds. The QSR metric is defined as the ratio between the number of reply packets received by a sink node in response to a query packet and the number of replies the sink expects to receive.

5.1 Basic simulation of the synchronization mechanism

The node synchronization mechanism was evaluated considering the following probabilistic distribution of network delays: (1) uniform distribution, (2) Gaussian distribution, and (3) exponential distribution.

Figure 13 shows one sink node and three sensor nodes. The sink node transmits queries regularly. Each query time reception is affected by those different network delays, and the sensor nodes upon their reception will adjust their sleep time in order to try to wake up at same time on the next application duty cycle. For each node, different mean delays were considered: sensor node 1, 0.5 s; sensor node 2, 1 s; and sensor node 3, 2 s.

A Python program was written in order to randomly generate different network delay distributions. The program generates 10^5 queries, uses Eq. 1 to estimate the new expected query reception time by each node, and uses it to adjust the time each node must sleep (Eq. 2) in order to wake up on time for the next application cycle. Finally, the program computes how many time the nodes are waked up simultaneously. We consider that nodes are simultaneously awaked up if the three sensors are awaked for at least $\Delta = 80\% \cdot T_{\text{ON}}$. Let us also define $T_{\text{Sensors}_{\text{ON}}}$ as a random variable which captures the time during which the three sensors are simultaneously on the ON state, having values $T_{\text{Sensors}_{\text{ON}}} \epsilon [0s, T_{\text{ON}}s]$ (see Fig. 14). An occurrence of $T_{\text{Sensors}_{\text{ON}}}$ is computed as the time the first sensor goes asleep minus the time the last sensor wakes up.

In a first attempt, for α in Eq. 1, the value was set to 0.125, following current IETF recommendations for managing TCP timers [32], and for Eq. 2, the β value was empirically set to 10. All the sensor nodes wake every 15 min remaining waked for 1 min ($T_{\text{ON}} = 60$ s and $T_{\text{OFF}} = 840$ s).

Figure 15 shows results for the first situation evaluated—uniformly distributed network delays, with delays varying between $\pm 20\% \times 0.5$ s, $\pm 20\% \times 1.0$ s, and $\pm 20\% \times 2.0$ s. In Fig. 15a, one can see the histogram of randomly generated delays; Fig. 15b shows $T_{\text{Sensors}_{\text{ON}}}$'s histogram. Again, we can observe that

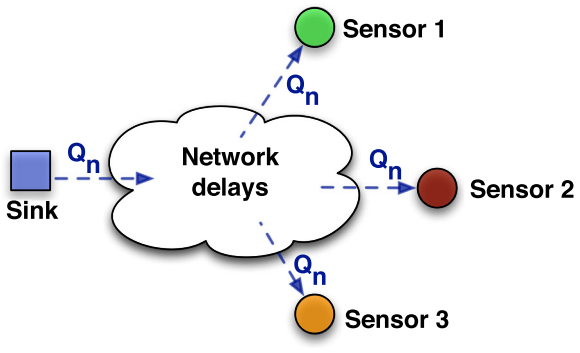

Fig. 13 WSN delay model

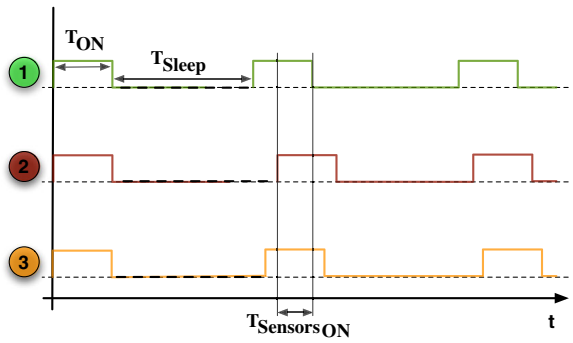

Fig. 14 Nodes adjustment of T_{ON} simultaneity

$P[T_{\text{Sensors}_{\text{ON}}} \geq \Delta] = 1$. In fact, it is verified that $T_{\text{Sensors}_{\text{ON}}} \epsilon [57.88, 59.66]$ s, and the mean value of $E[T_{\text{Sensors}_{\text{ON}}}] = 58.7$ s. As in the first situation, the nodes maintain synchronism in all the cycles.

Figure 16 shows results for the second situation evaluated—Gaussian distributed network delays, with delays having a standard deviation which is 20% of the mean values which are 0.5, 1.0, and 2.0s respectively. In

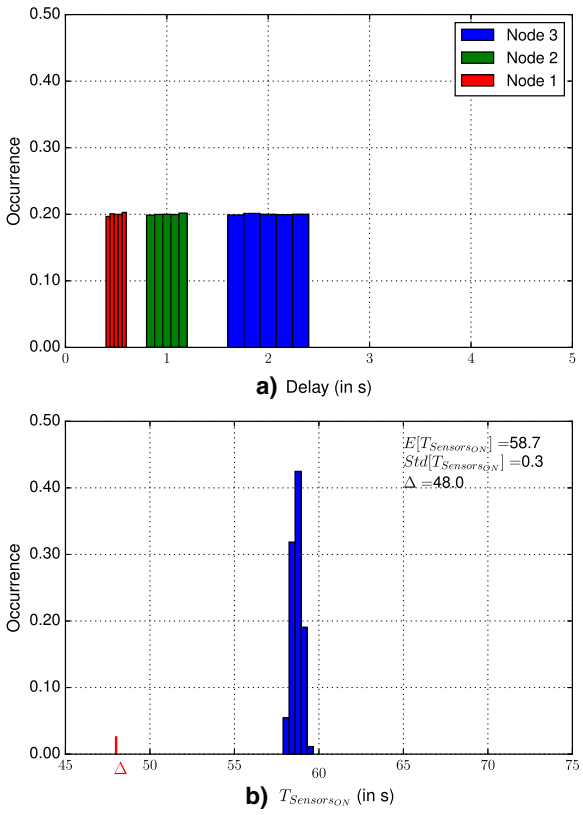

Fig. 15 Uniformly distributed network delays. **a** Delay histogram. **b** $T_{\text{Sensors}_{\text{ON}}}$'s histogram

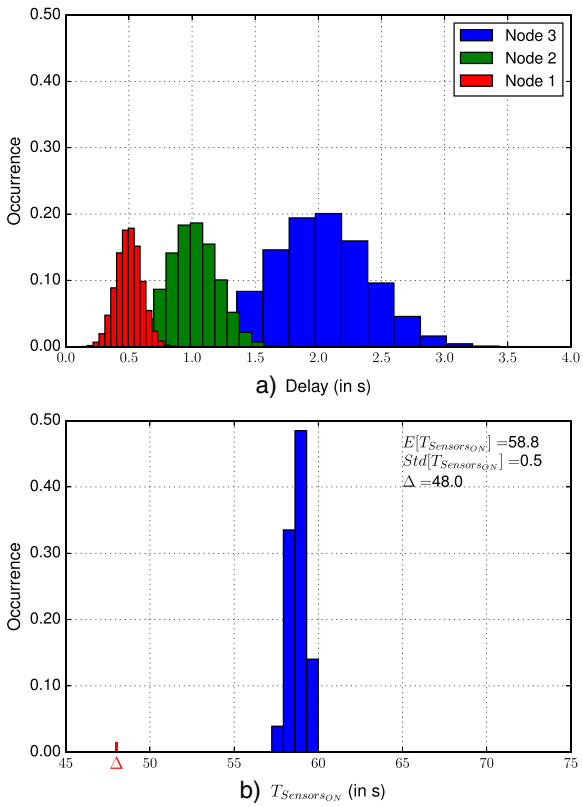

Fig. 16 Gaussian distributed network delays. **a** Delay histogram. **b** $T_{\text{Sensors}_{\text{ON}}}$'s histogram

Fig. 16a we can observe the histogram of randomly generated delays; Fig. 16b shows $T_{\text{Sensors}_{\text{ON}}}$'s histogram. As it can be observed, $\delta_{k,n}$ factor from Eq. 2 also affects the time each node must sleep ($T_{\text{Sleep}_{k,n}}$). Similar to the previous cases, $P[T_{\text{Sensors}_{\text{ON}}} \geq \Delta] = 1$, and the mean value is $E[T_{\text{Sensors}}] = 58.77$ s. In this situation the nodes will also maintain synchronism in every application cycle.

Figure 17 shows results from the last situation evaluated—exponentially distributed network delays, with mean delays targeting 0.5, 1.0, and 2.0 s, respectively. In Fig 17a, the histogram is shown and, as expected, there are variations; Fig. 17b shows $T_{\text{Sensors}_{\text{ON}}}$'s histogram. As can be observed, there are situations where the success condition is not satisfied. In this case, $E[T_{Sensors_{ON}}] = 57.52$ s and $T_{\text{Sensors}_{\text{ON}}} \epsilon [25.06, 59.99]$ meaning that the nodes will maintain synchronism by about 99% of the cycles.

Finally, Fig. 18 shows the box plot for the $\beta \cdot |\delta_{k,n}|$ component, which corresponds to the amount of time the nodes use to adjust sleep timers in order to wake up in synchronism in the next cycle. The worst value for the mean value of the $\beta \cdot |\delta_{k,n}|$ component is 1.11 s, and it corresponds to the exponential distribution, what means that a node will not sleep during T_{OFF} s but, in average, will sleep during

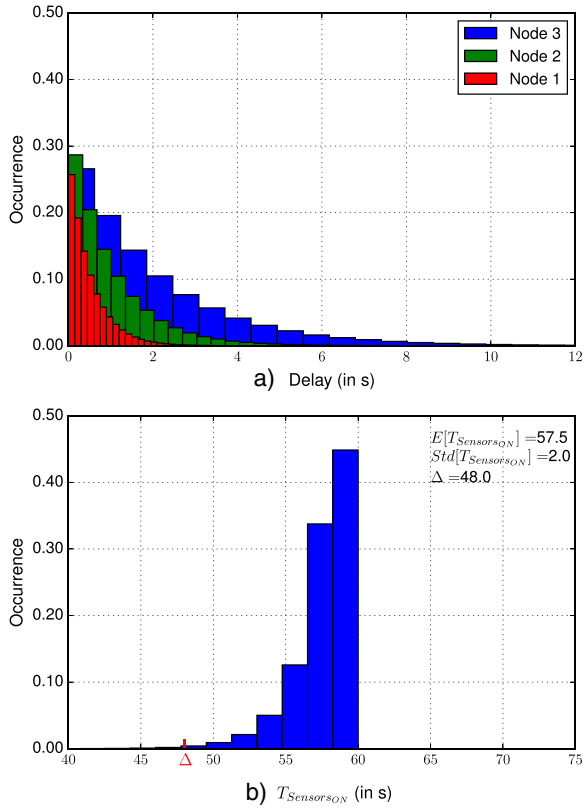

Fig. 17 Exponentially distributed network delays. **a** Delay histogram. **b** $T_{\text{Sensors}_{\text{ON}}}$'s histogram

$T_{\text{OFF}} - 1.11$s. Moreover, results showed that the probability $P[T_{\text{Sensors}_{\text{ON}}} \geq \Delta] = 1$ is observed in 99% of the occurrences, which means that all the considered nodes will be active at same time during at least $\Delta = 80\% \cdot T_{\text{ON}}$ in 99% of the application's duty cycles.

The box plot figures in this paper give the standard metrics: the 25th percentile, the 75th percentile, and the red line is the median value. The top and bottom of the whiskers show the maximum and minimum values, respectively. Finally, the black dashed line in the box represents the mean value.

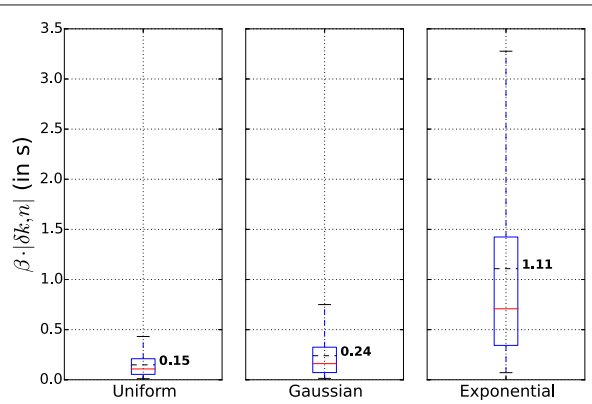

Fig. 18 Box plot for $\beta \cdot |\delta_{k,n}|$, the sleeping offset represented in Eq. 2

From this analysis, we may conclude that the synchronization mechanism may be adequate for our purposes. In order to increase the trust in these results, a sensibility analysis is also carried out, in order to understand how $T_{\text{Sensors}_{\text{ON}}}$ is affected by different values of α and β.

5.1.1 α and β values estimation

We performed studies using different values for the synchronization mechanism parameters α and β. We considered four sensor nodes and assumed a uniformly distributed delays varying in $\pm 20\% \times 0.5$ s, $\pm 20\% \times 1.0$s, $\pm 20\% \times 2.0$s. Figures 19, 20, 21, 22, 23, 24, 25, 26, and 27 show the results obtained when considering different values for the α and β parameters. Each of these figures present: (a) the $T_{\text{Sensors}_{\text{ON}}}$'s histogram; (b) the box plot for $T_{\text{Sensors}_{\text{ON}}}$ (in % of T_{ON}); and (c) the box plot for $\beta \cdot |\delta_{k,n}|$.

In the sensibility analysis shown below, we select two discrete set of values for α and β, $\alpha \in \{0.125, 0, 5, 0.875\}$ and $\beta \in \{1, 10, 50, 100\}$. We vary one parameter at time while maintaining the other constant.

α estimation: the weight given to the last sample in the calculation δ. Therefore, we want to investigate how it affects the synchronization mechanism by giving α different values, namely 0.125, 0.50, and 0.875.

β estimation since the $\delta_{k,n}$ value from Eq. 1 is small, we amplify it. The amplifying factor is the β parameter, and for it, we selected three values, $\beta \in \{10, 50, \text{and } 100\}$.

Figures 19, 20, 21, 22, 23, 24, 25, 26, and 27 show the results obtained for different combinations of the parameter's values. Table 1 summarizes it, showing: (a) the α and β values, (b) the $E[T_{\text{Sensors}_{\text{ON}}}]$, (c) average $E[T_{\text{Sensors}_{\text{ON}}}]$'s time in % of T_{ON}, and (d) $E[\beta \cdot |\delta_{k,n}|]$ component, the resulting sleeping offset.

For the selection of the α and β values, we considered the values that satisfy at the same time: (i) values of $T_{\text{Sensors}_{\text{ON}}}$ in % of T_{ON} above 80% and (ii) lowest $\beta \cdot |\delta_{k,n}|$ component value. Italicized values correspond to the ones that better satisfy our purposes.

5.1.2 Results discussion

This analysis of the results showed that not all the values chosen for α and β parameters satisfy our synchronization mechanism requirements. In fact, if we consider, respectively, $\alpha = 0.50$ and $\beta \in \{50; 100\}$, the mechanism will fail because the probability $P[T_{\text{Sensors}_{\text{ON}}} \geq \Delta] < 1$ (see, respectively, Figs. 23 and 24, what means that the sensor nodes will not be synchronized in all their duty cycles. The same applies if we consider $\alpha = 0.875$ and $\beta \in \{50; 100\}$, as shown in Figs. 26 and 27. From Figs. 26c and 27c,

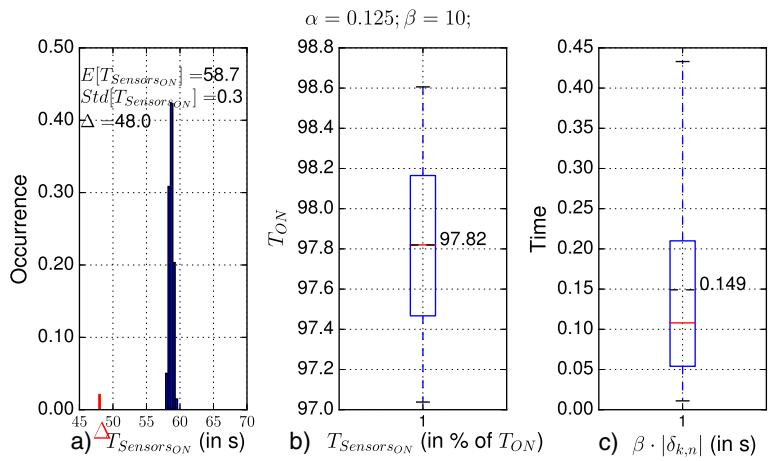

Fig. 19 Uniformly distributed network delays with $\alpha = 0.125$; $\beta = 10$. **a** $T_{\text{Sensors}_{\text{ON}}}$'s histogram. **b** Box plot for $T_{\text{Sensors}_{\text{ON}}}$ (% of T_{ON}). **c** Box plot for $\beta \cdot |\delta_{k,n}|$

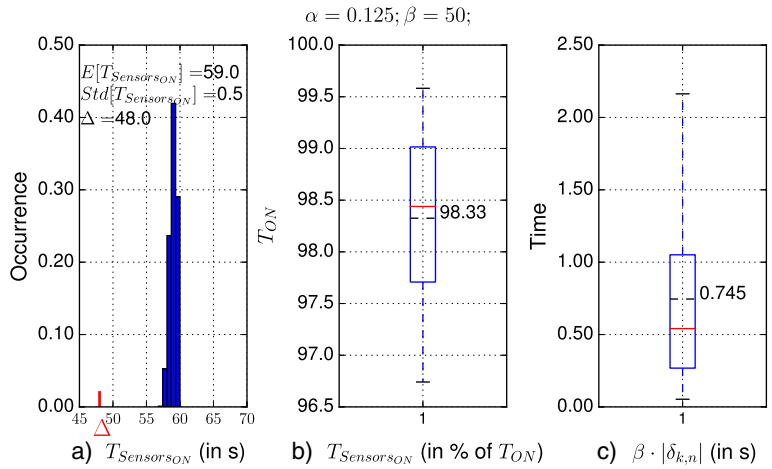

Fig. 20 Uniformly distributed network delays with $\alpha = 0.125$; $\beta = 50$. **a** $T_{\text{Sensors}_{\text{ON}}}$'s histogram. **b** Box plot for $T_{\text{Sensors}_{\text{ON}}}$ (% of T_{ON}). **c** Box plot for $\beta \cdot |\delta_{k,n}|$

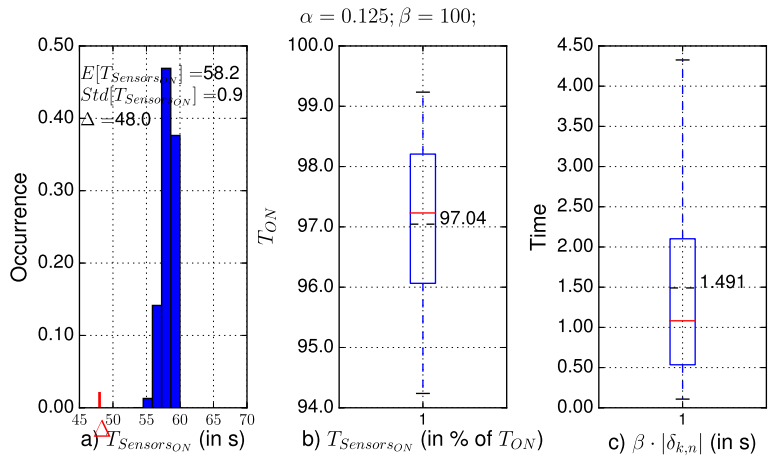

Fig. 21 Uniformly distributed network delays with $\alpha = 0.125$; $\beta = 100$. **a** $T_{\text{Sensors}_{\text{ON}}}$'s histogram. **b** Box plot for $T_{\text{Sensors}_{\text{ON}}}$ (% of T_{ON}). **c** Box plot for $\beta \cdot |\delta_{k,n}|$

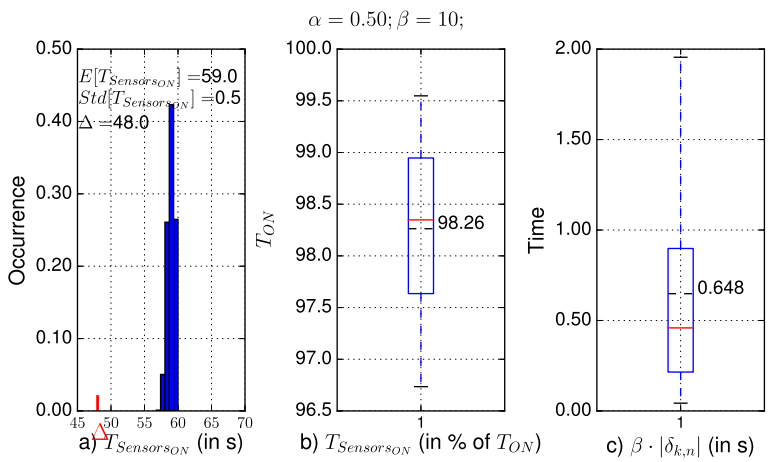

Fig. 22 Uniformly distributed network delays with $\alpha = 0.50$; $\beta = 10$. **a** $T_{\text{Sensors}_{\text{ON}}}$'s histogram. **b** Box plot for $T_{\text{Sensors}_{\text{ON}}}$ (% of T_{ON}). **c** Box plot for $\beta \cdot |\delta_{k,n}|$

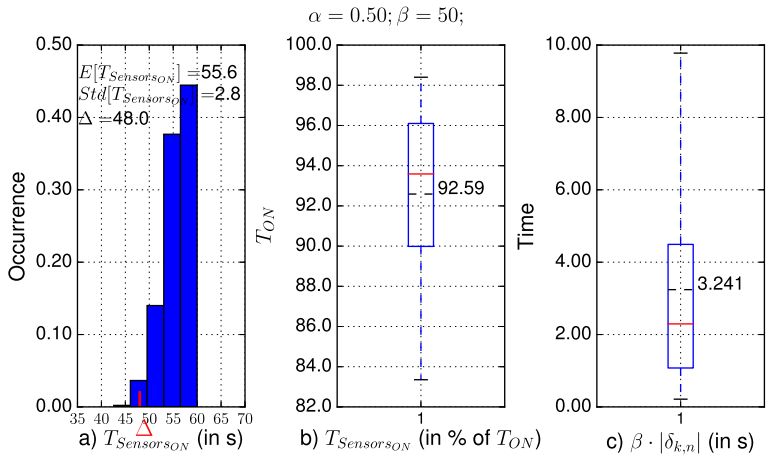

Fig. 23 Uniformly distributed network delays with $\alpha = 0.50$; $\beta = 50$. **a** $T_{\text{Sensors}_{\text{ON}}}$'s histogram. **b** Box plot for $T_{\text{Sensors}_{\text{ON}}}$ (% of T_{ON}). **c** Box plot for $\beta \cdot |\delta_{k,n}|$

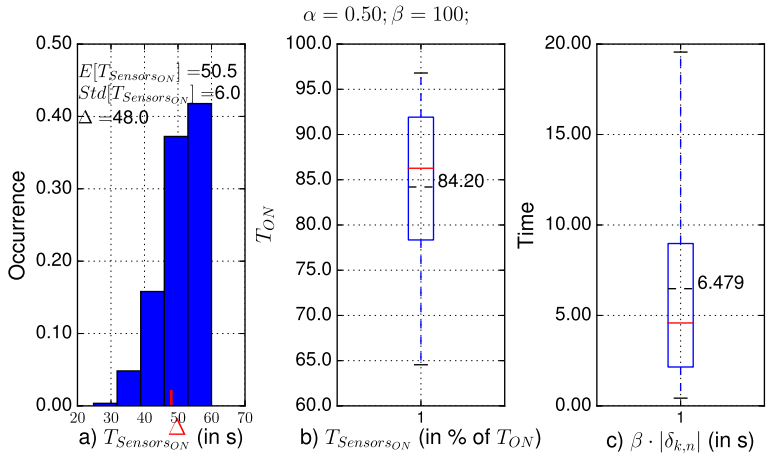

Fig. 24 Uniformly distributed network delays with $\alpha = 0.50$; $\beta = 100$. **a** $T_{\text{Sensors}_{\text{ON}}}$'s histogram. **b** Box plot for $T_{\text{Sensors}_{\text{ON}}}$ (% of T_{ON}). **c** Box plot for $\beta \cdot |\delta_{k,n}|$

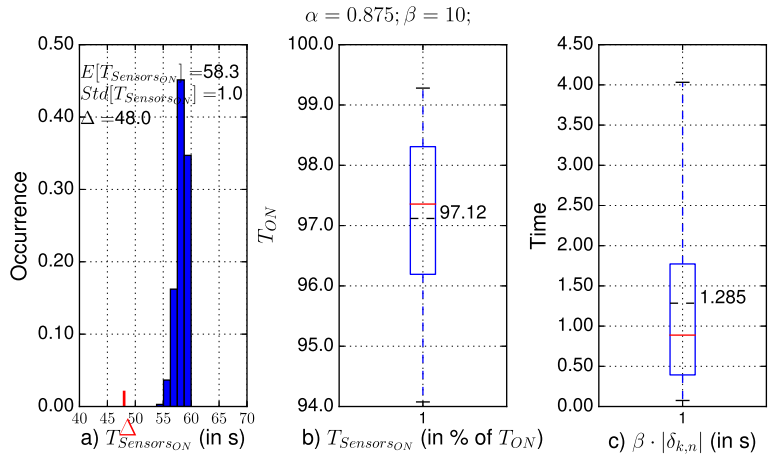

Fig. 25 Uniformly distributed network delays with $\alpha = 0.875$; $\beta = 10$. **a** $T_{\text{Sensors}_{\text{ON}}}$'s histogram. **b** Box plot for $T_{\text{Sensors}_{\text{ON}}}$ (% of T_{ON}). **c** Box plot for $\beta \cdot |\delta_{k,n}|$

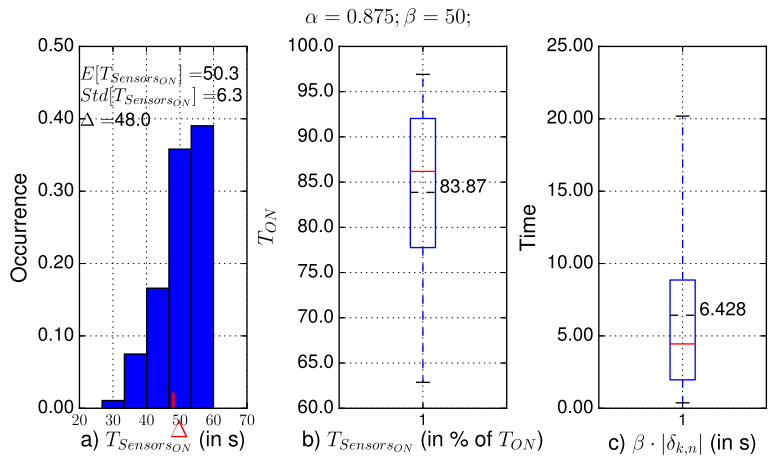

Fig. 26 Uniformly distributed network delays with $\alpha = 0.875$; $\beta = 50$. **a** $T_{\text{Sensors}_{\text{ON}}}$'s histogram. **b** Box plot for $T_{\text{Sensors}_{\text{ON}}}$ (% of T_{ON}). **c** Box plot for $\beta \cdot |\delta_{k,n}|$

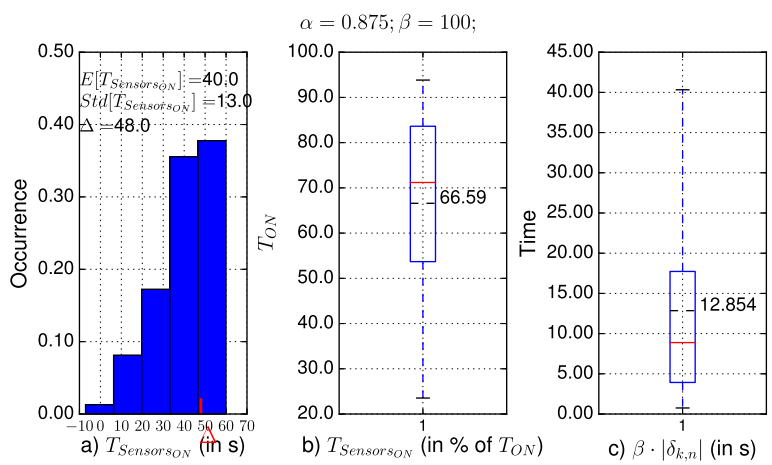

Fig. 27 Uniformly distributed network delays with $\alpha = 0.875$; $\beta = 100$. **a** $T_{\text{Sensors}_{\text{ON}}}$'s histogram. **b** Box plot for $T_{\text{Sensors}_{\text{ON}}}$ (% of T_{ON}). **c** Box plot for $\beta \cdot |\delta_{k,n}|$

Table 1 Summary of synchronization mechanism results as a function of α and β

Parameter		Results		
α	β	$E[T_{\text{Sensors}_{\text{ON}}}]$ (in sc)	$E[T_{\text{Sensors}_{\text{ON}}}]$ (in % of T_{ON})	$E[\beta \cdot \lvert\delta_{k,n}\rvert]$
0.125	*10*	*58.7*	*97.82*	*0.149*
	50	59.0	98.33	0.745
	100	58.2	97.04	1.491
	10	59.0	98.26	0.648
0.50	50	55.6	92.59	3.241
	100	50.5	84.20	6.479
	10	58.3	97.12	1.285
0.875	50	50.3	83.87	6.428
	100	40.0	66.59	12.854

we can observe that there are occurrences for $T_{\text{Sensors}_{\text{ON}}}$ below 80%, the threshold established for success, being in average equal to 97.82% of T_{ON}. Therefore, those values do not satisfy our selection criteria. From the other values evaluated, we may consider that $\alpha = 0.125$ and $\beta = 10$ are the values that better satisfy our purposes, for the scenarios considered. Comparing to other pair of values for α and β, these values present at the same time (i) greater $T_{\text{Sensors}_{\text{ON}}}$ value (58.7), which is almost the same theoretical value of T_{ON}; (ii) all the occurrences for $T_{\text{Sensors}_{\text{ON}}}$ in terms of T_{ON}% are above 97%, being in average equal to 97.82%; and (iii) the mean $\beta \cdot |\delta_{k,n}|$ component has, in average, the lowest value 0.149 what means that the sensor nodes have to wake up before the next duty cycle less time than in the other cases. This will have impact in energy consumption since the sensor nodes do not have to stay unnecessary time awaked.

5.2 Simulations

In [6] and in [7], two different applications were used in three different scenarios, being the nodes distributed as shown in Fig. 28. Simulations ran in *Contiki's Cooja* simulator [33]. All the nodes are within a distance of 25 m for a transmission range of 30 m and support one of the two applications. Each application is running in eight nodes, and each node runs a single application. In scenario 1, the nodes running App. A were selected in a way that a long path could be obtained; in scenario 2, both applications have the same node distribution; scenario 3 is used to investigate situations where at least one node from other application is required to relay data. Let us, for example, consider Fig. 28c. In this scenario, we can observe that node 9 routes/forwards packets of an application that it does not run. In the scenarios simulated, sink nodes are always awake, and sink node running App. B (node 9) was chosen as the network DAG root because of its application duty cycle. For the nodes running application A, $T_{\text{ON}} = 60$ s, $T_{\text{OFF}} = 3540$ s; for the nodes running application B, $T_{\text{ON}} = 60$ s, $T_{\text{OFF}} = 840$ s.

We simulated two situations: (i) a situation where all the nodes join the network at the same time, so that the proposed synchronization mechanism is not used as, in simulations with *COOJA*, clock drifting is the same for all the sensor nodes and (ii) the nodes will join the network at different times. The later implies the use of the synchronization mechanism described in Section 4 in order to keep the nodes synchronized with respect to the applications they run. The nodes join the network at different times which were randomly generated between 317 and 1102 s.

5.2.1 *Results and discussion*

In [34], the authors noticed timing inaccuracies in comparison to experiences made on TelosB motes hardware. Their simulations showed unexplained delays during packet transmission (TX) over the radio medium that were not observed during similar experiences on physical motes. According to their investigations, they discovered that the problem is with the emulation of MSP430-

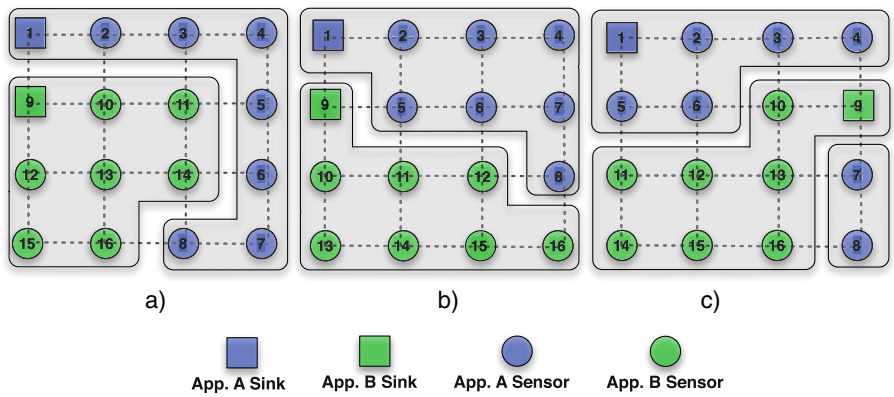

Fig. 28 Scenarios simulated. **a** Scenario 1. **b** Scenario 2. **c** Scenario 3

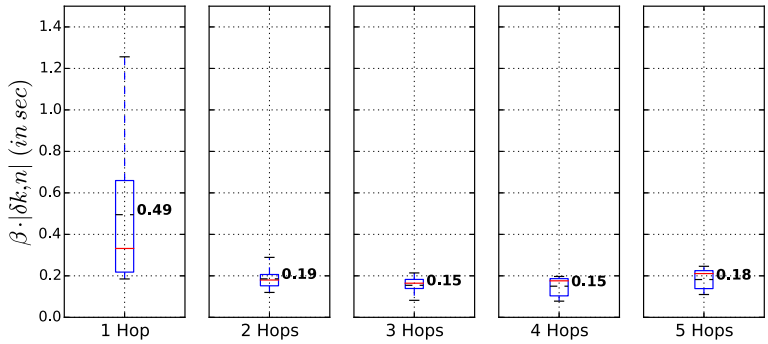

Fig. 29 Box plot for $\beta \cdot |\delta_{k,n}|$, with $\alpha = 0.125$ and $\beta = 10$, for each hop in scenario 1

powered, radio-enabled WSN motes by the MSPSim software package when loading packet data into the transmission buffer. The emulating mote performs this TX buffer loading at a different speed than the actual hardware. This may result in inexact simulations results. Nevertheless, the authors argue that, for the WSN application studied, time precision is not a key issue since the applications are not designed for real-time critical applications. The authors have selected the TelosB Hardware platform and ContikiOS/Cooja because there is no need to write the code twice since it is the same for physical motes and emulated motes, and the TelosB platform is the most used platform in the academia. This time inaccuracy has no impact in our synchronization mechanism. The EWMA technique used to control the synchronization of the sensor nodes also considers the resulting unexplained delays during packet transmission to estimate the arrival of the next query packet, and the simulation and testbed results show that the synchronization mechanism performs well when having different network delays.

In our solution, each time a node receives a query, it computes the time it must wake up before the start of the next application cycle in order to be able to receive and forward packets and to successfully reply back to the sink. The synchronization mechanism was configured with $\alpha = 0.125$ and $\beta = 10$.

In the simulations, 16 nodes have been used, half of them running each application. Each scenario was simulated ten times, and information was extracted in order to estimate delays, *QSR*, and the $E[\beta \cdot |\delta_{k,n}|]$ component. The results obtained are the following.

Delays We considered *delay* as the sum of all per-hop delays for each sensor query packet reception and characterized by the sum of the processing and queueing delays in intermediate and destination sensor nodes and the transmission delays and propagation delays in intermediate nodes.

Per hop $\beta \cdot |\delta_{k,n}|$ component: From the simulations, we have extracted information about the $\beta \cdot |\delta_{k,n}|$ component on a per-hop basis. Fig. 29 shows the box plot for this component in scenario 1. We can observe that, except for the first hop, this component presents per hop similar values, and sensor nodes would have to wake with an average sleeping offset of about 0.232 s. In the first hop, the sleeping offset has a grater value (0.49 s in average) because in this hop, we can observe some congestion, particularly between the sink node (node 1) and the sensor node 2.

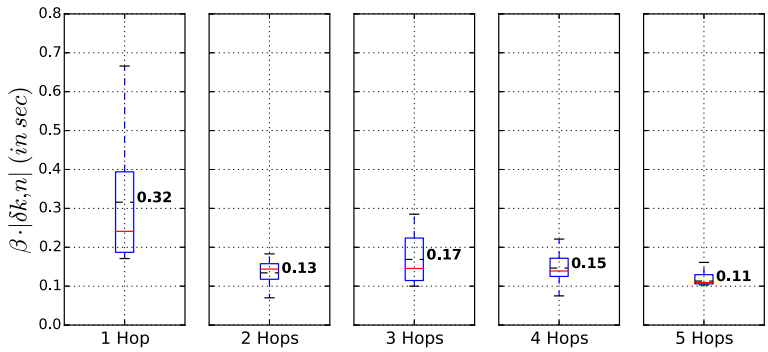

Fig. 30 Box plot for $\beta \cdot |\delta_{k,n}|$, with $\alpha = 0.125$ and $\beta = 10$, for each hop in scenario 2

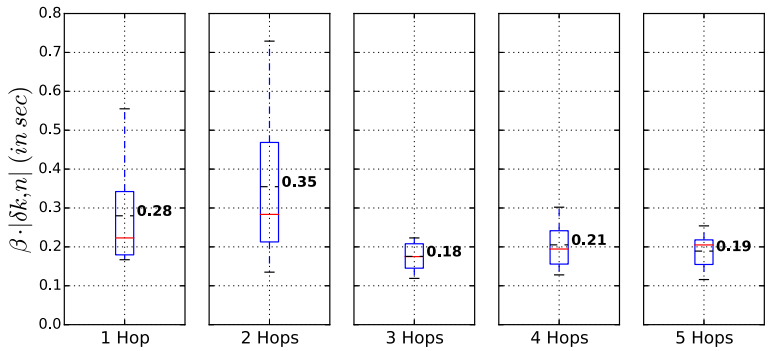

Fig. 31 Box plot for $\beta \cdot |\delta_{k,n}|$, with $\alpha = 0.125$ and $\beta = 10$, for each hop in scenario 3

Figure 30 shows the box plot for the $\beta \cdot |\delta_{k,n}|$ component in scenario 2. As in scenario 1, we can also see that this component presents similar values per hop, with an average sleeping offset of about 0.176 s.

Finally, Fig. 31 shows the box plot for the $\beta \cdot |\delta_{k,n}|$ component in scenario 3. As in the other two scenarios, we observed that this component presents similar values per hop, in an average of about 0.242 s. In the case of the sleeping offset for two hop nodes, it has in average a grater value (0.35 s). Analyzing this scenario's topology, and the traffic that may occur, we can observe some congestion around sensor nodes 3 and 13. For sensor node 3, it needs to forward replies from sensor nodes 4, 7, and 8. For sensor node 13, it also forwards replies from sensor nodes 11, 12, 14, 15, and 16. However, this sleeping offset value can be also considered as negligible as it has a small additional value.

In Fig. 32, the box plot for the expected sleeping offset value for each of the three scenarios studied is shown. As it can be verified, the nodes would sleep not the T_{OFF} time, but in average $T_{\mathrm{OFF}} - 0.716$ s. Moreover, we observed that, independent of the network topology, this component has almost the same values, what confirms that the synchronization mechanism proposed is adequate for our purposes. Moreover, comparing the results from Figs. 34 and 35, we observe that for scenarios 2 and 3, the maximum and minimum $\beta \cdot |\delta_{k,n}|$ values are different. In scenario 2, the nodes have more neighbors running the same application, what implies that each of them may need more time to access the wireless medium to forward a query. This is also reflected on the network delays and affects the $\beta \cdot |\delta_{k,n}|$ component.

QSR: Fig. 33 shows the box plot for *QSR*. In this figure,(1) the results using the standard *RPL* routing protocol, (2) the results using *RPL-BMARQ* solution proposed in [7] without the synchronization mechanism implemented, and (3) the results using the same *RPL-BMARQ* solution fully implemented are showed. As it can be observed, in average, 98.8% of the queries sent by sinks are replied by sensor nodes. With this success ratio, we can argue that the quality of the proposed synchronization mechanism is confirmed.

5.3 Testbed experiments

In order to confirm the results obtained from theoretical studies and simulations, we also tested our proposed solution in a real environment. For that purpose, two of the scenarios studied were selected (scenarios 1 and 3) and

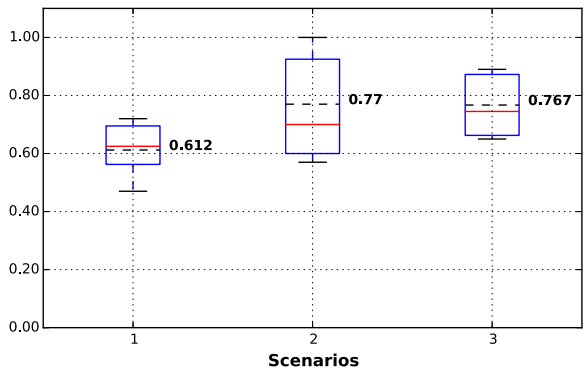

Fig. 32 Box plot for $\beta \cdot |\delta_{k,n}|$, with $\alpha = 0.125$ and $\beta = 10$, for each scenario (in s)

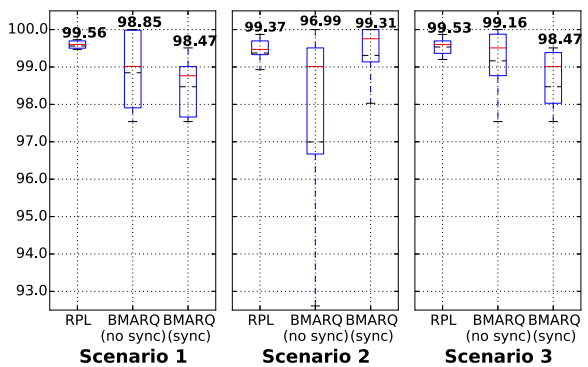

Fig. 33 Mean query success ratio—QSR for each scenario (in %)

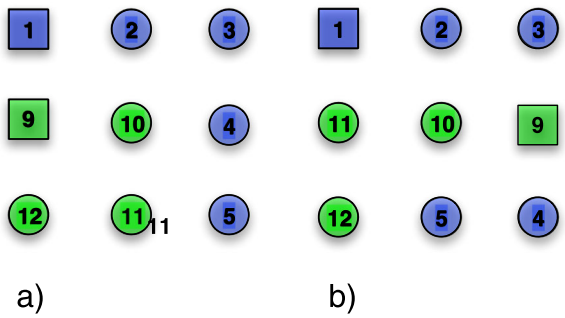

Fig. 34 Scenarios deployed. **a** Deployment 1. **b** Deployment 2

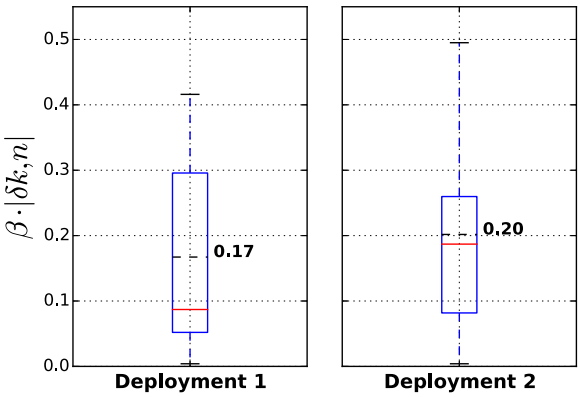

Fig. 36 Box plot for $\beta \cdot |\delta_{k,n}|$, with $\alpha = 0.125$ and $\beta = 10$, for each deployment (in s)

deployed. Since it was not possible to reproduce them at the same scale, the scenarios deployed correspond to a 3×3 square lattice topology, while keeping all the other functionalities. In order to obtain reliable terms of comparison, we have simulated these deployments using the same methods as in Section 5.2 and compared the simulated results with those obtained in testbeds. Figure 34 shows both topologies deployed and simulated, which were realized using *TelosB motes* [35], placed at distances of 5 m, and the radio transmission power was reduced to −7 dBm in order to reduce the node reception distance. Application A run in five nodes (1, 2, 3, 4, and 5), and application B runs in four nodes (9, 10, 11, and 12). Node 9 is, at the same time, the root of the DAGs and a sink. Node 1 is the other sink. The nodes ran *ContikiOS* (2.6) [36] which is an operating system for WSN which incorporates an implementation of the *IPv6* protocol stack and uses *RPL* as the default routing protocol.

5.3.1 Results and discussion

Each testbed experiment was carried out for 4 h. To log real-time data, two *Raspberry Pi* platforms were used, connected to both sink nodes via a serial connection. Inside each Raspberry Pi [37] platform was a python program running, responsible to get timestamp data from each sink with respect to query packets sent and reply packets received. In order to verify our proposed synchronization mechanism, we considered in this work (i) synchronization parameter's values $\alpha = 0.125$ and $\beta = 10$; (ii) packet reception time on the sink nodes side to estimate the expected reception time and to compute the sleeping offset component ($\beta \cdot |\delta_{k,n}|$); and (iii) *QSR* results. The main results obtained include the following:

$\beta \cdot |\delta_{k,n}|$ component: Figure 35 shows $\beta \cdot |\delta_{k,n}|$ component, the sleeping offset represented in Eq. 2 histogram for each deployment. As expected, it presents the same uniform distribution characteristics as the theoretical evaluation and the simulations performed. Moreover, we can see in Fig. 36 that this component presents in average a sleeping offset of 0.185 s.

QSR: Figure 37 shows simulation and real implementation results. As it can be seen, both present same values (100%), which means that also in real testbeds, the nodes reply to all the queries sent by sinks, going to sleep and waking up while being synchronized.

From the above results, we can conclude that there are no major differences between what was observed in

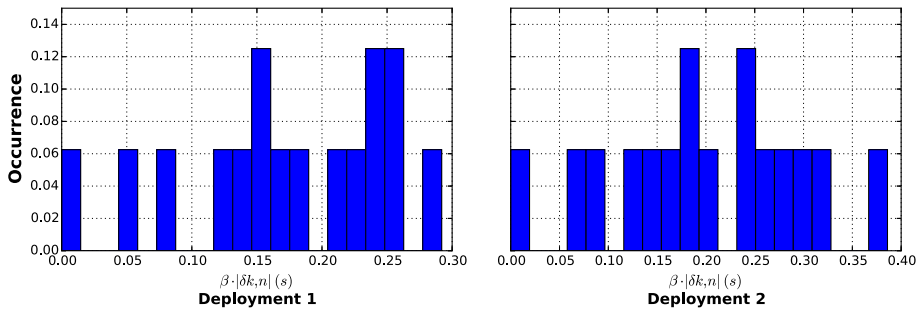

Fig. 35 $\beta \cdot |\delta_{k,n}|$, with $\alpha = 0.125$ and $\beta = 10$, histogram for each deployment (in %)

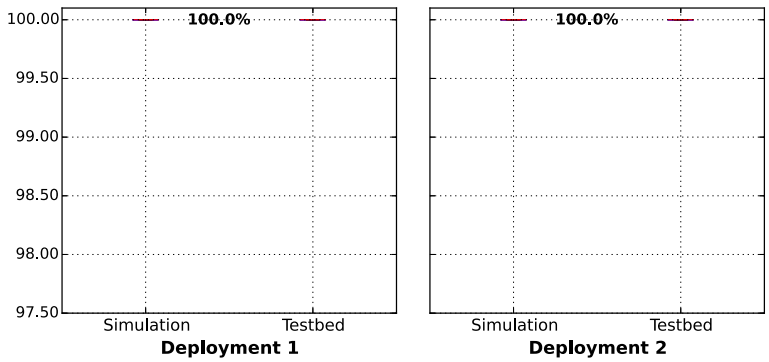

Fig. 37 Query success ratio (QSR) for the scenarios deployed (in %)

the theoretical studies and in the simulation environment and what was expected in the testbed environment. This confirms the usability and the quality of the synchronization mechanism proposed, when applied to *application-driven WSNs* with the characteristics described in this work.

6 Conclusions

This paper proposed an application-driven WSN synchronization mechanism using the EWMA technique to maintain synchronization of all the nodes in WSNs defined by the applications they run. The paper presents and discusses the performance of the synchronization mechanism for sensor devices using *IEEE 802.15.4* radios. The work presented reflects our analysis of the mechanism which assumes that the nodes are affected by different network delay distributions. The mechanism allows the nodes to go asleep and to wake up in synchronism. The mechanism was evaluated by means of simulations using *ContikiOS* and *Cooja*, and confirmed its functionalities. Finally, real testbed experiments confirmed our simulations results, showing that the mechanism also works in real applications.

Appendix

Exponentially weighted moving average

The exponentially weighted moving average (EWMA) [38] is a technique used for calculating a run-time average characterized by giving less and less weight to data as they get older and older. EWMA is easily plotted and may be also viewed as a forecast for the next observation. The EWMA equals the present predicted value plus *lambda* times the present observed error of prediction,

$$\text{EWMA} = \hat{y}_t + \lambda(y_t - \hat{y}_t) \tag{3}$$

where $\hat{y}_t$ is the predicted value at time t (the old EWMA), y_t is the observed value at time t, $y_t - \hat{y}_t$ is the observed error at time t, and λ is a constant ($0 < \lambda < 1$) that determines the depth of memory of the EWMA. Equation 3 can be written as

$$\hat{y}_{t+1} = \lambda y_t + (1 - \lambda)\hat{y}_t \tag{4}$$

EWMA statistics are currently used, for instance, by TCP to recover from undelivered segments; the mechanism is based on [39] and EWMA is used to estimate the timeout value that depends on the round trip time.

Acknowledgements

This work was financed by the Project "NORTE-07-0124-FEDER-000056" by the North Portugal Regional Operational Programme (ON.2 - O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF), and by national funds, through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT) within the fellowship "SFRH/BD/ 36221/2007". The authors would like to thank also the support from the Faculty of Engineering, University of Porto, to thank the support from the INESC TEC, and to thank the support from the School of Technology and Management of Viseu.

Competing interests

The authors declare that they have no competing interests.

Ethics approval and consent to participate

The authors declare that the work does not contain any studies with human participants or animals performed by any of the authors; the work has not been published before (except in the form of an abstract or as part of a published lecture, review, or thesis); the work is not under consideration elsewhere; copyright has not been breached in seeking its publication; and that the publication has been approved by all co-authors and responsible authorities at the institute or organization where the work has been carried out.

About the authors

Bruno Marques received in 2017 a PhD degree in Electrical and Computers Engineering from the University of Porto. He is an Adjunct Professor at the School of Technology and Management of Viseu, where he gives courses in industrial communications and computer networks. He also is an invited collaborator at the Centre for Telecommunications and Multimedia of the INESC TEC research institute.

Manuel Ricardo received in 2000 a PhD degree in Electrical and Computers Engineering from Porto University. He is an associate professor at the Faculty of Engineering of the University of Porto, where he gives courses in mobile communications and computer networks. He also leads the Centre for Telecommunications and Multimedia of the INESC TEC research institute.

Received: 29 March 2016 Accepted: 2 February 2017
Published online: 21 February 2017

References

1. IEEE-Computer-Society, IEEE Std 802.15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs) (2006). Revision of IEEE Std 802.15.4-2003
2. G Montenegro, N Kushalnagar, D Culler, Transmission of IPv6 Packets over IEEE 802.15.4 Networks (2007). IETF
3. CM Tang, Y Zhang, YP Wu, in *Instrumentation, Measurement, Computer, Communication and Control (IMCCC) 2012 Second International Conference on*. The P2P-RPL Routing Protocol Research and Implementation in Contiki Operating System, (2012), pp. 1472–1475. doi:10.1109/IMCCC.2012.345
4. T Winter, P Thubert, A Brandt, J Hui, R Kelsey, P Levis, K Pister, R Struik, J Vasseur, R Alexander, RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550 (Proposed Standard) (2012). http://www.ietf.org/rfc/rfc6550.txt
5. IF Akyildiz, MC Vuran, *Time Synchronization*. (John Wiley and Sons, Ltd, 2010), pp. 243–263. http://dx.doi.org/10.1002/9780470515181.ch11
6. BF Marques, MP Ricardo, in *Ad Hoc Networking Workshop (MED-HOC-NET) 2014 13th Annual Mediterranean*. Improving the energy efficiency of WSN by using application-layer topologies to constrain RPL-defined routing trees, (2014), pp. 126–133. doi:10.1109/MedHocNet.2014.6849114
7. B Marques, M Ricardo, in *Wireless Networks, The Journal of Mobile Communication, Computation and Information*. Energy-efficient node selection in application-driven WSN, (2016). doi:10.1007/s11276-016-1194-2 http://link.springer.com/article/10.1007%2Fs11276-016-1194-2
8. DL Mills. IEEE Trans. Commun. **39**, 1482 (1991)
9. W Su, I Akyildiz, Time-diffusion synchronization protocol for wireless sensor networks. IEEE/ACM Trans. Networking. **13**(2), 384 (2005). doi:10.1109/TNET.2004.842228
10. J Elson, L Girod, D Estrin, Fine-grained network time synchronization using reference broadcasts. SIGOPS Oper. Syst. Rev. **36**(SI), 2002. doi:10.1145/844128.844143. http://doi.acm.org/10.1145/844128.844143
11. S Ganeriwal, R Kumar, MB Srivastava, in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems. SenSys '03*. Timing-sync protocol for sensor networks (ACM, New York, 2003), pp. 138–149. doi:10.1145/958491.958508 http://doi.acm.org/10.1145/958491.958508
12. J van Greunen, J Rabaey, in *Proceedings of the 2Nd ACM International Conference on Wireless Sensor Networks and Applications. WSNA '03*. Information assurance in sensor networks (ACM, New York, 2003), pp. 11–19. doi:10.1145/941350.941353. http://doi.acm.org/10.1145/941350.941353
13. H Dai, R Han, TSync: a lightweight bidirectional time synchronization service for wireless sensor networks. SIGMOBILE Mob. Comput. Commun. Rev. **8**(1), 125 (2004). doi:10.1145/980159.980173 http://doi.acm.org/10.1145/980159.980173
14. R Carli, A Chiuso, L Schenato, S Zampieri, Optimal Synchronization for Networks of Noisy Double Integrators. IEEE Trans. Autom. Control. **56**(5) (2011). doi:10.1109/TAC.2011.2107051
15. B Etzlinger, H Wymeersch, A Springer, Cooperative Synchronization in Wireless Networks. IEEE Trans. Signal. Process. **62**(11), 2837 (2014). doi:10.1109/TSP.2014.2313531
16. W Yuan, N Wu, B Etzlinger, H Wang, J Kuang, Cooperative Joint Localization and Clock Synchronization Based on Gaussian Message Passing in Asynchronous Wireless Networks. IEEE Trans. Veh. Technol. **65**(9), 7258 (2016). doi:10.1109/TVT.2016.2518185
17. M Leng, YC Wu, Distributed Clock Synchronization for Wireless Sensor Networks Using Belief Propagation. IEEE Trans. Signal Process. **59**(11), 5404 (2011). doi:10.1109/TSP.2011.2162832
18. J Pearl. Artif. Intell. **29**(3), 241 (1986)
19. J He, P Cheng, L Shi, J Chen, Y Sun, Time Synchronization in WSNs: a Maximum-Value-Based Consensus Approach. IEEE Trans. Autom. Control. **59**(3), 660 (2014). doi:10.1109/TAC.2013.2286893
20. B Sundararaman, U Buy, AD Kshemkalyani. Ad Hoc Netw. **3**(3), 281 (2005)
21. T Ma, Z Xu, M Hempel, D Peng, H Sharif, Performance Analysis of a Novel Low-Complexity High-Precision Timing Synchronization Method for Wireless Sensor Networks. IEEE Trans. Wirel. Commun. **13**(9), 4758 (2014). doi:10.1109/TWC.2014.2331286
22. M Al Ameen, SMR Islam, K Kwak. Int. J. Distrib. Sensor Netw. **2010**, 16 (2010). http://dx.doi.org/10.1155/2010/163413%]163413
23. M Miller, N Vaidya, A MAC protocol to reduce sensor network energy consumption using a wakeup radio. IEEE Trans. Mob. Comput. **4**(3), 228 (2005). doi:10.1109/TMC.2005.31
24. W Ye, J Heidemann, D Estrin, in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. An energy-efficient, MAC protocol for wireless sensor networks, vol. 3, (2002), pp. 1567–1576. doi:10.1109/INFCOM.2002.1019408
25. T van Dam, K Langendoen, in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems. SenSys '03*. An adaptive energy-efficient, MAC protocol for wireless sensor networks (ACM, 2003), pp. 171–180. doi:10.1145/958491.958512. http://doi.acm.org/10.1145/958491.958512
26. W Pak, KT Cho, J Lee, S Bahk, in *Global Telecommunications Conference 2008. IEEE GLOBECOM 2008*. W-MAC: Supporting Ultra, Low Duty Cycle in Wireless Sensor Networks (IEEE, 2008), pp. 1–5. doi:10.1109/GLOCOM.2008.ECP.79
27. A El-hoiydi, Jd Decotignie, *in 9th International Symposium on Computers and Communications (ISCC '04)*, (2004), pp. 244–251
28. Y Sun, O Gurewitz, DB Johnson, in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems. SenSys '08*. RI-MAC: a receiver-initiated asynchronous duty cycle, MAC protocol for dynamic traffic loads in wireless sensor networks (ACM, New York, 2008), pp. 1–14. doi:10.1145/1460412.1460414. http://doi.acm.org/10.1145/1460412.1460414
29. E Ancillotti, R Bruno, M Conti, Reliable Data Delivery with the IETF Routing Protocol for Low-Power and Lossy Networks. IEEE Trans. Ind. Inform. **10**(3) (2014). doi:10.1109/TII.2014.2332117
30. D Carels, N Derdaele, ED Poorter, W Vandenberghe, I Moerman, P Demeester, Support of multiple sinks via a virtual root for the RPL routing protocol. EURASIP J. Wirel. Commun. Netw. **2014**(1), 91 (2014). http://dx.doi.org/10.1186/1687-1499-2014-91
31. P Pinto, A Pinto, M Ricardo, in *Wireless Days (WD) 2013 IFIP*. End-to-end delay estimation using RPL metrics in WSN, (2013), pp. 1–6. doi:10.1109/WD.2013.6686524
32. V Paxson, M Allman, HJ Chu, M Sargent, Computing TCP's Retransmission Timer (2011). http://tools.ietf.org/html/rfc6298
33. F Osterlind, A Dunkels, J Eriksson, N Finne, T Voigt, in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*. Cross-Level Sensor, Network Simulation with COOJA, (2006), pp. 641–648. doi:10.1109/LCN.2006.322172
34. K Roussel, YQ Song, O Zendra, in *EWSN 2016 - NextMote workshop*, ed. by K Roemer. ACM EWSN 2016- NextMote workshop (Junction Publishing, Graz, Austria, 2016), pp. 319–324. https://hal.inria.fr/hal-01240986
35. Crossbow TelosB. http://www.memsic.com/userfiles/files/Datasheets/WSN/6020-0094-02_B_TELOSB.pdf
36. A Dunkels, Contiki OS, open source, highly portable, multi-tasking operating system for memory-efficient networked embedded systems and wireless sensor networks (2013). http://www.contiki-os.org
37. Pi RaspBerry (2014). http://www.raspberrypi.org
38. JS Hunter. Qual. Technol. **18**, 203 (1986)
39. V Jacobson, Congestion avoidance and control. SIGCOMM Comput. Commun. Rev. **18**(4), 314 (1988). doi:10.1145/52325.52356. http://doi.acm.org/10.1145/52325.52356