Fault Recovery in Time-Synchronized Mission Critical ZigBee-Based Wireless Sensor Networks

Davide Scazzoli[1] · Atul Kumar[1] · Navuday Sharma[1] · Maurizio Magarini[1] · Giacomo Verticale[1]

Abstract Reliability and precise timestamping of events that occur are two of the most important requirements for mission critical wireless sensor networks. Accurate timestamping is obtained by synchronizing the nodes to each other while reliability can be obtained by eliminating single points of failure (SPF). In this paper, we address the SPF problem of a ZigBee-based wireless sensor network by means of using multiple coordinators with different personal area network identifiers (PAN IDs). We propose a solution where members of a network switch from one coordinator to another in case of failure by changing their respective PAN ID. We verify experimentally that our solution provides gains in terms of recovery speed and, therefore, synchronization accuracy with respect to a solution proposed in the literature.

Keywords Wireless sensor networks (WSNs) · Reliability · Time synchronization · Single Point of Failure (SPF) · Mission critical

✉ Maurizio Magarini
maurizio.magarini@polimi.it

Davide Scazzoli
davide.scazzoli@mail.polimi.it

Atul Kumar
atul.kumar@polimi.it

Navuday Sharma
navuday.sharma@polimi.it

Giacomo Verticale
giacomo.verticale@polimi.it

[1] Dipartimento di Elettronica, Informazione e Bioingegneria Politecnico di Milano, 20133 Milan, Italy

1 Introduction

Wireless Sensor Networks (WSNs) are characterized by the use of tiny, low cost and low energy devices, termed *nodes*, that are designed to provide connectivity in an environment without any preexisting infrastructure. Their main purpose is to collect and aggregate data acquired from sensors, installed on-board end nodes, that measure various physical quantities [1]. When developing and designing a WSN platform, that is thought for a specific application, devel-opers have to satisfy a rather wide range of requirements. The search for a balance between cost of sensor nodes and requirements represents a challenging task in each specific application. Therefore, one of the main goal in ongoing research activities is the improvement of WSNs technical characteristics to expand their field of application. This can be made possible, without significantly increased costs, by developing new technologies and protocols [1]. Concern-ing these two latter aspects, recently, we have witnessed increasing interest in the use of WSN for mission critical applications, where the requirement of reliable data trans-mission from source to destination is one of the most important challenges together with the possibility of pro-viding precise timestamping of events [2].

The flexibility of WSNs allows their use in a variety of innovative applications: from small body area networks, used to monitor health parameters [3], to large arrays of sensor nodes, used to monitor structural integrity [4] or dangerous physical phenomenons such as volcanic erup-tions [5]. Besides providing sustainability against ele-ments' failures, in order to correctly interpret data collected from multiple sensor nodes in different locations and time instants, it is necessary to maintain a common notion of time across all the nodes. This requirement is usually achieved by means of time synchronization protocols [6].

A popular technology for providing connectivity in WSNs is the IEEE 802.15.4 standard for *Low-Rate Wireless Personal Area Networks*, which has been specifically developed for low-power devices such as the nodes of a WSN [1]. Many standards are available that rely on IEEE 802.15.4 for the lower layers: ZigBee [7], WirelessHART [8], and 6loWPAN [9]. Each of them has its own advan-tages and disadvantages when used for implementing a WSN. Among these protocols we focus here on ZigBee, being it the most widely adopted standard in industrial applications mainly because its low cost hardware [10].

Low energy consumption in ZigBee networks is achieved thanks to the distinction of nodes in two cate-gories: Full Function Devices (FFDs) and Reduced Function Devices (RFDs). The FFD nodes are always on and are tasked with providing the infrastructure of the network, thus allowing extended sleep cycles for other devices. On the other hand, RFD nodes are sleepy end devices that establish a single link to one FFD, referred to as its parent [7]. The FFDs in a ZigBee network are divided in Coor-dinator and Routers. Each ZigBee network has a unique Coordinator, which is tasked with network formation and maintenance. Only one coordinator is allowed per network and once a network is formed no other coordinators may join it even if the original coordinator shuts down or has a failure. Different ZigBee coordinators operating in the same physical channel will be part of different networks, each one with its own unique Personal Area Network Identifier (PAN ID). ZigBee does not present a solution to deal with the loss of the coordinator, leading to a Single Point of Failure (SPF) problem [11]. Since this is a well known problem, a solution has been presented in [12] to address this issue. The proposed approach solves the issue by storing backups of the coordinator on a router node which, in case of failure, is reprogrammed to become a clone of the missing coordinator.

In this paper, we propose and assess a simpler alternative based on the changing of the PAN ID. Our solution gives advantages in terms of lower downtime, because it takes advantage from ZigBee's low network joining time of end node devices, which is in the order of a few milliseconds [13], and in terms of ease of implementation, since it uses only features provided by the standard. Moreover, we present a simple technique for achieving time synchronization at the sensor level and experimentally evaluate its performance, with special focus on the impact that the WSN recovery after a failure has on the synchronization of the sensor nodes within the network. Our experiments were performed using commercial off the shelf (COTS) devices, which are often ill suited to meet the requirements of mission critical applications. The purpose of this work is enabling devices that would otherwise be unsuitable for mission critical applications to at least meet requirements of link robustness. The specific requirements of mission critical applications can be very different and may include requirements that cannot be satisfied by ZigBee such as very low latency, in those cases different communication protocols may be able to overcome this issue. Link robustness, however, remains a common requirement to all critical applications where loss of connection will lead to loss of profit or worse. We, therefore, focus on this important aspect.

The rest of the paper is organized as follows. Section 2 gives a survey of the synchronization methods available for WSNs and of the solutions present in the literature to solve the ZigBee's SPF problem. Our proposed solution to the SPF problem is described in Sect. 3. Section 4 illustrates the specific synchronization protocol implemented in our prototype. Experimental results are given in Sect. 5 and, finally, Sect. 6 concludes the paper.

2 State of the Art in Synchronization and Solutions for Failure Recovery in Zigbee WSNs

2.1 Methods for WSN Synchronization

Three basic methodologies for WSN synchronization are available in the literature [14]:

– *Relative timing* each node monitors and compensates for offset and clock drifts with respect to other nodes. Synchronization is achieved by exchanging local timestamps between different nodes.
– *Relative ordering* a relative order of events and communications is achieved by comparing local clocks. With this method it is not necessary to compensate clock offset.
– *Global synchronization* if a global timescale such as GPS is available, all the nodes will synchronize.

Global synchronization is the best solution with respect to network overhead as each device can independently synchronize itself without using network resources. The main issue of *global synchronization* is the requirement of a GPS receiver in each node which is in direct contrast with requirements of low cost and low energy for WSNs. *Relative ordering*, on the other hand, is an attractive solution for its reduced burden on the nodes with respect to *global synchronization*. However, it is not a general solution as this method only establishes an order of events without offering actual clock synchronization. The most widespread synchronization method for WSNs is *relative timing* as it gives a good compromise between the other two alternatives. Several timing protocols use this methodology: Timing-sync Protocol for Sensor Networks (TPSN), Flooding Time Synchronization Protocol (FTSP), and Reference Broadcast Synchronization (RBS) [14]. These protocols differ in the way they estimate the offset and clock drift for the independent nodes [15]. A brief overview of these protocols is given below:

- TPSN is based on a server client paradigm where the client sends a request for synchronization information to a timing server. It is divided in two phases: level discovery and synchronization. The level discovery phase has the goal of creating a hierarchy that establishes which nodes are servers and which nodes are clients for each synchronization phase. Multihop networks will have multiple levels where clients of one level will act as servers for the level above them. In this architecture the root node will always be at the bottom level. The estimation of the offset is achieved by the one-to-one exchange of timestamps between servers and clients.
- FTSP is a variation of TPSN which was developed to deal with frequent topology changes. There are two main differences with respect to TPSN: the root node is not fixed but it is instead constantly re-elected and the synchronization information is flooded throughout the network.
- RBS differs from the other protocols as it is based on a receiver-to-receiver paradigm [16]. A reference beacon is broadcast by a third party and all receiving nodes exchange the relative time of reception for the beacon. The sent beacon does not contain any timing information and the synchronization is achieved by comparing the times of reception that the receivers exchange between one another.

As ZigBee provides a low transmission rate we implemented a TPSN protocol [17] that will be discussed in Sect. 4. In order to minimize protocol overhead, we focused on a static topology where the levels have been manually assigned and where the discovery phase was not implemented. We chose TPSN specifically over the other two methods that require broadcasts because they are poorly supported by the hardware we used to build our prototype. The general advantages and disadvantages of a broadcast approach versus a uni-cast approach are covered in Sect. 4.

2.2 Solutions for Failure Recovery in Zigbee WSN

In WSNs for mission critical applications the goal is to avoid that the failure of a device impacts on the functionalities of the rest of the network [18]. ZigBee has been designed for low power, low cost and easy to deploy wireless devices. It is mostly used in WSNs and other applications requiring low data rate transmissions and only provides limited solutions to manage failures. In the present work, we are interested to the case where the failure of a node can jeopardize the operation of the whole network, as it might happen for the failure of the ZigBee PAN coordinator.

A ZigBee network can be set up as a tree, star or mesh network. However, all these topologies suffer from the SPF problem. In a star topology the coordinator is always the central node and, therefore, if the coordinator fails the whole network fails. For tree topology, the coordinator must be the root node and, in the event of a coordinator failure, the network will experience a partial failure. On the other hand, mesh topology is the most robust and, depending on its configuration, the whole network may remain intact after a coordinator failure. However, even in this last case, a failure of the entire network is possible if, for example, indirect binding of devices is used because binding tables are getting lost after a failure of the coordinator. New releases of the standard, such as ZigBee PRO, aim at reducing this problem and introduce a distributed storing of the binding tables across the routers [19]. Even in ZigBee PRO, however, it is required to have a trust center that is unique and, often, coincides with the coordinator of the network. Should this trust center fail, the network can continue working for some time, but functionalities such as authentication for joining/rejoining devices will be lost. Furthermore the network will progressively fail as the security keys expire [19].

Since the solution adopted in ZigBee PRO solves only partially the problem, a solution based on ZigBee is proposed in [12] where backups of the coordinator are stored in the routers. In the case of a coordinator failure the routers can be reprogrammed to become a clone of the coordinator. This solution is transparent to the application layer and requires minimal hardware overhead. The downside is the long time required to reprogram and reset a device, which leaves a gap in the data if the coordinator is used as the network sink. Moreover, many commercial devices do not allow resetting device parameters, including the 64-bit device *unique identifier*, thus making the solution difficult to implement using COTS hardware.

3 Proposed Fast Recovery Procedure

The proposed fast recovery procedure relies on the use of redundant PAN coordinators in hot standby, ready to provide connectivity to other devices. Also, we introduce a procedure, implemented by the nodes belonging to the compromised network, to detect the failure and re-connect to a new coordinator. As is well known, ZigBee protocol is designed to have a low joining time. Our solution exploits this feature to minimize network downtime with respect to

[12]. For real-time applications, where data continuity is a critical requirement, this can be achieved by introducing a redundant management of hardware components as illustrated in Fig. 1. The Figure reports the case with 4 coordinators and 2 sensors that allows to recover up to 3 coordinator failures and 1 sensor failure.

It is worth observing that, with such a setup, the system can recover up to $N - 1$ coordinator failures and $M - 1$ sensor failures, where N and M are the number of employed coordinators and sensors, respectively. Also, this allows us to separate the redundancy of the coordinator from that of the sensor which, as shown in Fig. 1, can have very different requirements. Gateways or data sinks are more critical than individual sensors thus they often have more stringent redundancy requirements which translate in a higher number of redundant components like the example shown in Fig. 1.

Each sensor node implements an algorithm to monitor the connection status and discover coordinator failures by timeouts. After detection of a failure it begins the recovery procedures. A block diagram of the monitoring algorithm which automates this procedure is reported in Fig. 2.

The algorithm distinguishes between *Normal operation* and *Failure recovery*. A coordinator failure is detected by the lack of ACK replies to the data messages that are sent during normal operation. After a failure has been identified the recovery procedure begins switching trough PAN IDs to search for another coordinator. If no network joining notification is received, the algorithm will try again using a

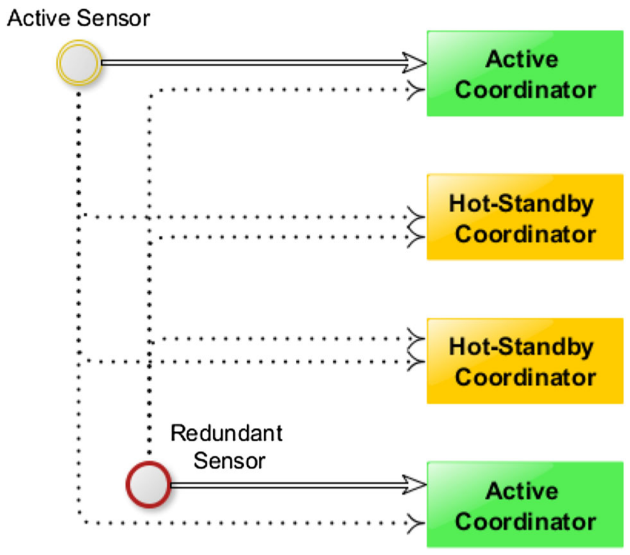


Fig. 1 Proposed scheme for maintaining data continuity through redundant sensor nodes during transition from active to backup coordinator. *Dotted lines* represent links formed only in case of failure. The scheme illustrates the case with N = 4 coordinators and M = 2 sensor nodes for a particular measure. The sensor nodes can have independent active and backup connections

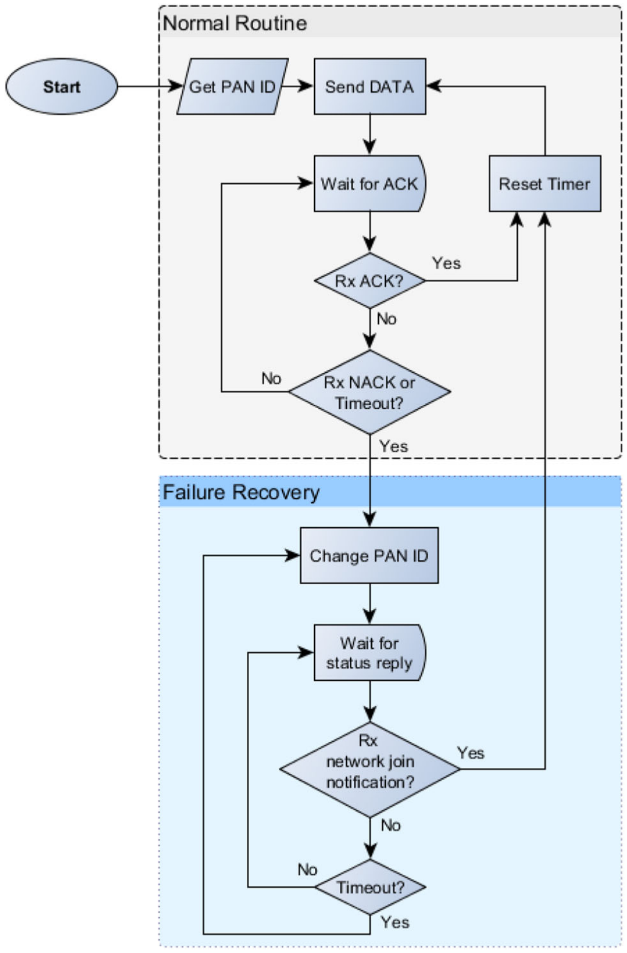


Fig. 2 Algorithm to automate PAN ID change on all nodes of the network. The PAN IDs of the available coordinator are assumed known

different PAN ID. The list of available PAN IDs for switching can be dynamically distributed by the coordinators when a new device joins the network. This procedure is executed on all network nodes that are affected by the failure of the coordinator. The proposed algorithm has been experimentally tested using commercial XBee S2 ZigBee devices mounted on Arduino boards. A full description of the experimental set-up will be given in Sect. 5.

The main difference with respect to the solution proposed in [12] is that our proposed method is based on switching to another network by changing the PAN ID rather than having a router reprogrammed to become the new coordinator. We have tested both the case of single and multiple device recovery at the same time. The measured performance is the downtime with respect to the solution proposed in [12] as well as the impact of network formation overhead when multiple routers join a coordinator at the same time. In case the event of multiple devices switching exactly at the same time conflicts will be resolved at the MAC layer. The downside of our approach is the increased hardware cost required to improve the downtime performance. If only end-devices need to re-associate then the downtime will be very low as expected by the ZigBee standard. However, a much higher down-time is observed if routers also need to switch network as will be seen in Sect. 5. However, the performance of this solution can be improved by adding Quality of Service (QoS) classes, where highest class will be the first to be allowed to reconnect while other ones will have to wait a certain time before attempting reconnection. It is easy, for example, to implement QoS in the form of a time that lower classes have to wait before initiating the failure recovery procedure, as illustrated in Fig. 3.

4 Synchronization Management in WSNs

As explained in Sect. 2.1, synchronization protocols rely on the exchange of timing information between nodes. Because of this they are susceptible to random delays in the delivery of the information which can impact on the synchronization accuracy. The following non-deterministic factors are responsible for the degradation in the accuracy of synchronization protocols:

- *Sender uncertainty* covers all the variable delays attributed to the sender. It can be subdivided in
  - *Send time* the time taken by the node to construct a packet and pass it to the MAC layer after its transmission has been decided.
  - *Access time* indicates how much the packet must wait at the MAC layer before being transmitted. It is highly variable and it depends heavily on network traffic.
  - *Transmission time* this is the delay introduced by the transmission at the physical layer. It is mainly deterministic, provided the connection speed does not change. There is still some variations given by possible interruptions during transmission.
- *Propagation delay* this delay is introduced by the transmission medium. However, since the distance between nodes in a WSN is often very small, in the order of a few hundred meters at most, this contribution is negligible.
- *Receiver uncertainty* covers the variable delays attributed to the receiver. It can be further subdivided in:
  - *Reception time* analogous to the transmission time, it is the time taken to receive the bits and pass them to the MAC layer.
  - *Receive time* the time taken to construct a packet from the received bits and pass it to the upper layers. The uncertainty is introduced by the variable delays introduced by the operating system.

We will now explain in detail how offset and skew are estimated within the synchronization protocol we used. Offset is estimated by exchanging the four timestamps from T_1 to T_4 between server and client, as illustrated in Fig. 4. The timestamps indicate the following specific events:

- T_1 indicates the local time for the client when a synchronization request is sent.
- T_2 indicates the local time for the server when it receives the synchronization request.
- T_3 indicates the local time for the server when a synchronization reply is sent.
- T_4 indicates the local time for the client when it receives the reply to its request.

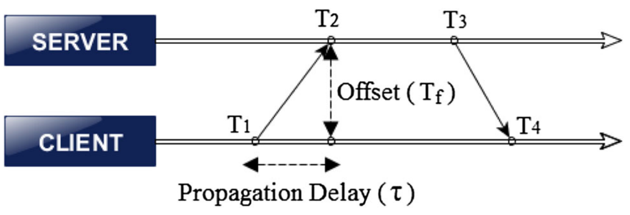


Fig. 4 Space–time diagram showing the four different time instants used in the protocol

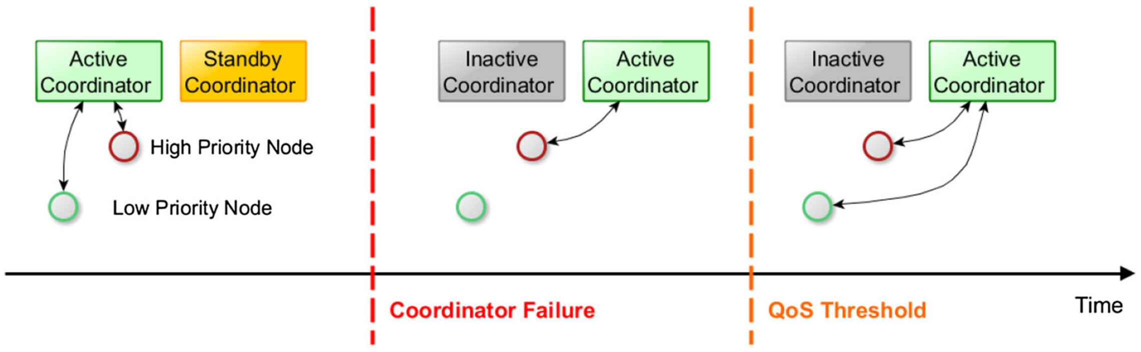


Fig. 3 Example of a QoS implementation for network switching based on time windows

With reference to Fig. 4, we have

$$T_2 = T_1 + T_f + \tau, \tag{1}$$

$$T_4 = T_3 - T_f + \tau, \tag{2}$$

where T_f is the offset between the local clocks and τ is the transmission delay. Assuming the delay τ is symmetric, from (1) and (2) we can write

$$T_f = \frac{(T_2 - T_1) - (T_4 - T_3)}{2}, \tag{3}$$

$$\tau = \frac{(T_2 - T_1) + (T_4 - T_3)}{2}. \tag{4}$$

A periodic compensation of the offset is simple to implement on most devices. However, the accuracy of the synchronization depends on the interval within which synchronization packets are exchanged. Using this approach the frequency offset between the various nodes quickly builds up and increases the synchronization error. The size of data used for synchronization is limited and its impact can be further reduced by attaching them to data frames. However, its burden remains proportional to the number of nodes to synchronize for TPSN and to the squared number of nodes to synchronize for RBS. In ZigBee networks, where the number of nodes can be in the order of thousands, this means that TPSN offers better scalability. It is therefore beneficial to implement not only clock offset correction but also frequency offset, or *skew*, estimation. With reference to Fig. 5, the skew can be calculated as

$$\text{skew} = \frac{T_f' - T_f}{\text{timing interval}}, \tag{5}$$

where T_f is obtained from the timestamps T_1 to T_4 and T_f' from the timestamps T_1' to T_4'. The advantages on protocol overhead given by this approach come at the cost of increased complexity and memory footprint. Particular issues arise when skew estimation is implemented on hardware constrained devices, such as the end-sensor nodes, where sizeable precision errors can arise from floating point multiplications. Hence, it is necessary to avoid directly multiplying the clock but instead track offset from a recent point in time.

While the Sender–Receiver approach of TPSN reduces protocol overhead with respect to RBS, estimation of the offset suffers from all the variable delays described at the beginning of this Section. We have therefore implemented a moving average filter in order to limit the impact of these errors on skew estimation. If from one side the implementation of skew estimation is resource consuming, on the other side the advantages in terms of reduced protocol overhead and precision are substantial. We have quantified the gain by taking measures from experimental results as described in the next Section. More details about the aspect of synchronization in WSNs can be found in [20] where the trade-offs of sender–receiver architectures like TPSN with respect to receiver–receiver ones like RBS are examined, together with the impacts of the different types of variable delays which affect these algorithms.

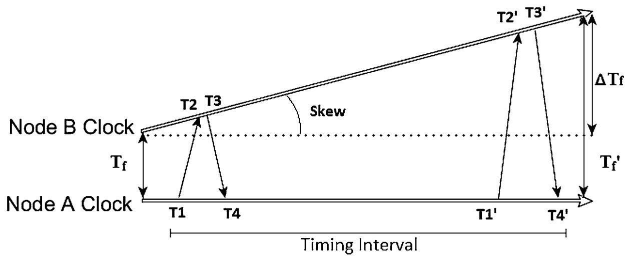


Fig. 5 Algorithm to calculate the timing skew and offset

5 Experimental Results and Measures

For the SPF problem of ZigBee networks we have performed several tests using COTS devices to measure the downtime of our method with respect to resetting a node, as proposed in [12]. Our experimental set-up consists of a Raspberry Pi board that controls two coordinators and several Arduino boards equipped with ZigBee S2 XBee transceivers [21]. These boards are also wired to the Raspberry Pi via their interrupt pins which are used for forcing simultaneous events such as timestamping in order to measure synchronization errors.

The experimental setup is displayed in Fig. 6. We have measured a downtime defined as the interval between the failure of the coordinator and the successful joining to the new coordinator. The delay introduced by the fault discovery algorithm was not considered as it is a common factor. We have logged the data by using the utility called

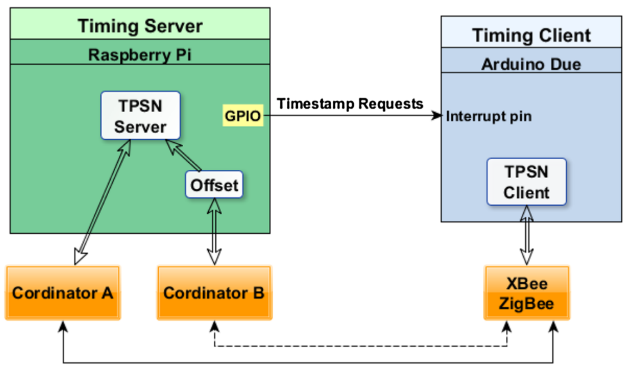


Fig. 6 A single client experimental set-up, multiple clients can be connected at the same time with the same configuration

XCTU which is freely provided by the module manufacturer [22].

Figure 7 shows results of our measures for a single router device for the two considered approaches. Data have been obtained by 100 runs for each method. Numerical results show that the proposed approach allows for a sig-nificant reduction in downtime with respect to the node resetting one. From a comparison of the results, the average downtime reduction is around 15 s. However, it is worth observing that the downtime of the proposed method depends on the number of routers that have to perform the switch. Since for end devices the switch happens very quickly, within a few milliseconds, the limiting factor is represented by the number of routers in the network because they must implement network formation.

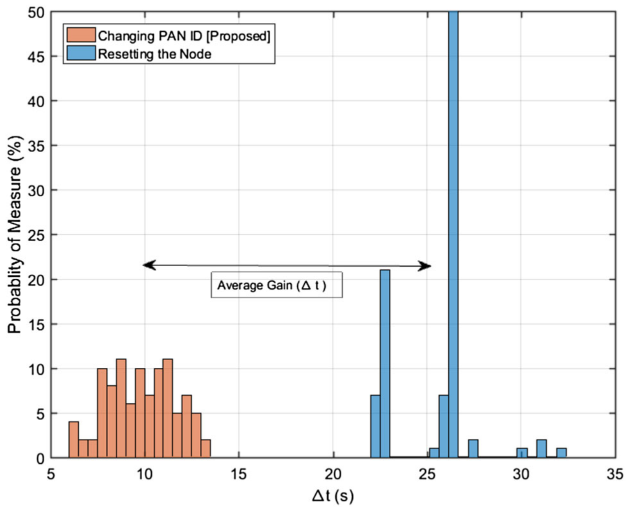
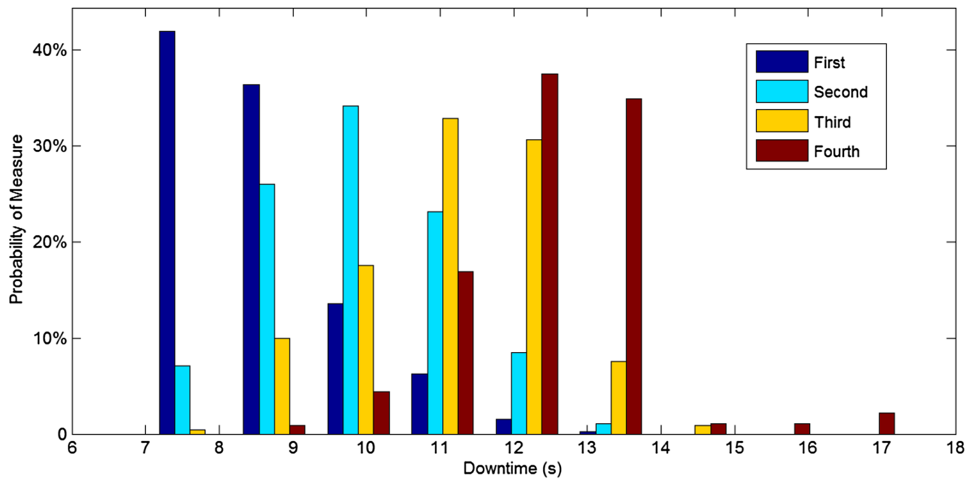


Fig. 7 Histogram of the observed downtime with the proposed scheme versus resetting the node for a single router

Results presented in [23] have been therefore expanded by performing a similar experiment where routers have been forced to reconnect at exactly the same time, using a wired interrupt. The measured downtime for each of them is shown in Fig. 8. From the numerical results we can see that each router, on average, adds between 1 and 2 s of downtime, that is caused by the overhead in establishing new network connections and bindings. This result means that the performance of our solution is dependent on the number of routers that must perform this switch and it can quickly worsen with respect to [12] for large numbers of routers. On the other hand we have experimentally verified that end devices are capable of joining a new network within a few milliseconds, as expected by the ZigBee standard, provided an active router or coordinator is in the range.

The experimental setup in Fig. 6 has been used to study also the synchronization problem. We have a TPSN server running on a Raspberry Pi and the TPSN clients running on Arduino boards connected to the server via the ZigBee XBee modules. In order to accurately measure the synchronization error we used interrupts on the Arduinos triggered by the Raspberry Pi's GPIO. Both the two devices were connected to a PC for data logging: the Arduino via UART and the Raspberry Pi via Ethernet. Our first experiment consisted in measuring the effects on the synchronization of the network outage in both the case of simple offset compensation and skew estimation. We let the clients synchronize to the server for some time in order to give enough samples for skew estimation. A number of 10 samples was used. After this, we simulated a network failure by unplugging the transceivers from the server and measuring the evolution of the synchronization error. The results for this experiment are shown in Fig. 9. At time $T = 0$ the connection is stopped and the evolution of the

Fig. 8 Histogram of the observed downtime with multiple routers switching at the same time



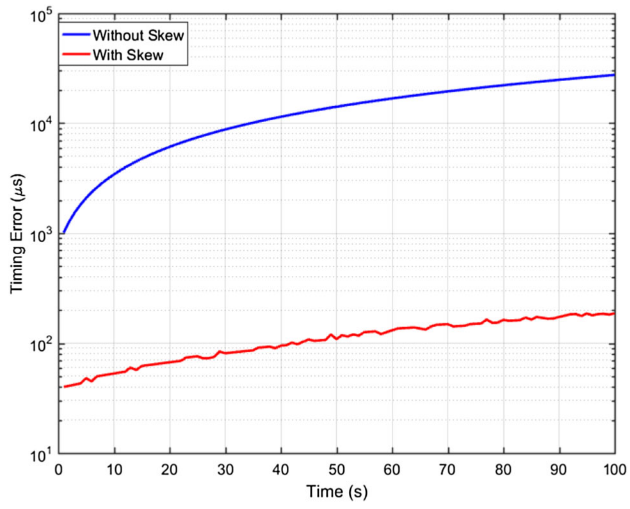


Fig. 9 Evolution over time of the synchronization error with and without skew compensation

error is measured trough timed interrupts triggered by the wired connection. We have verified that skew estimation significantly reduces the error of synchronization, particularly over extended periods of time. The accuracy of the synchronization itself was limited by the use of application layer timestamping as MAC layer one was not available on the used COTS devices. The curves do not start exactly at zero because we have some error in the offset estimation itself caused by the variable delays explained in Sect. 4. Furthermore, a small ripple is visible in the skew corrected line. This is caused by the limited accuracy of the timestamps available on the Arduino, which, have a coarse graduation of 4 µs. This error, however, is at least two orders of magnitude lower than that introduced in the offset estimation, which, as can be seen from the starting points in Fig. 9, is in the order of a few milliseconds. As our set-up consists of commercially available hardware and we made no further assumption on link reliability, the shown results can be expected in most deployment scenarios. It is worth noting that by comparing the different downtimes of the two methods reported in Fig. 7 with the evolution of the synchronization error reported in Fig. 9 we can see that there is an advantage in terms of synchronization accuracy. In particular, we can measure values ranging from 3 ms to 7 ms of reduction in synchronization error for the case of simple offset estimation and 20 µs to 40 µs for the case with skew estima-tion. Finally, we studied the evolution of the synchronization error during a recovery process using the setup described in Fig. 6. For skew estimation we used a moving average filter with 10 samples in order to average the errors in offset estimation. We performed a test without any offset between the two networks and one with a 10 ms offset and studied its effects on the offset and skew estimation. Synchronization messages are exchanged every 4 s. Hence, we allow some time for filling up the skew estimation filters and, then, we proceed to crash the network coordinator at $T = 30$ s. After a timeout the end devices start synchronizing with the new network where, in our setup, the synchronization is provided by the same device to eliminate any uncertainty in the offset. The results we have obtained are reported in Fig. 10, where it can be seen that in the case without offset the synchronization error follows the expected linear curve due to the clock drift and no impact is observed on the skew estimation. In contrast, for the case with offset its impact on the skew estimation as the slope of the skew line is highly skewed. This is corrected after 10 synchronization events as the error from the offset introduced by the network switch is discarded by the moving average filter.

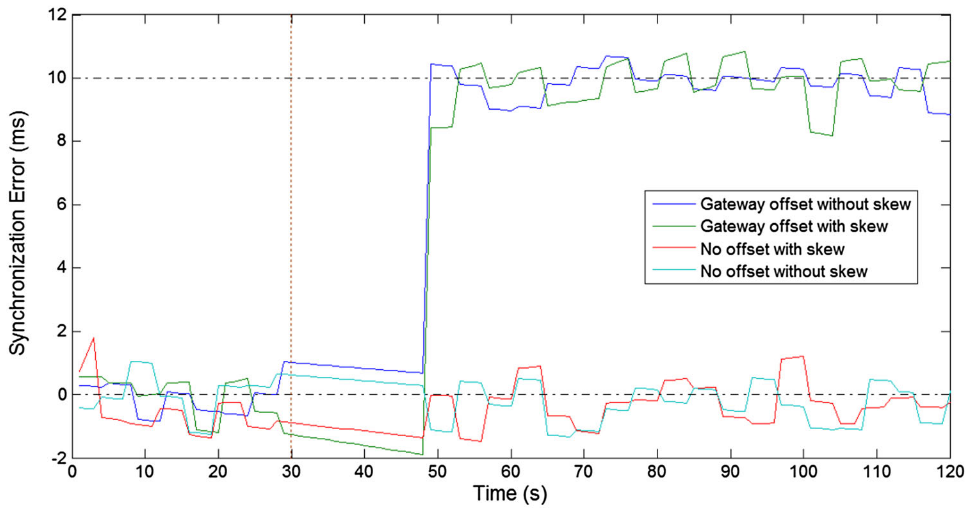


Fig. 10 Evolution over time of the synchronization error after a network switch in $T = 30$ s. Results show both the case of no offset and a 10 ms offset between networks

6 Conclusion

This paper focuses on the issue of time synchronization in wireless sensor networks (WSNs), the problem of the Single Point of Failure for ZigBee coordinators, and how the latter impacts on the former. We propose a new method for dealing with ZigBee coordinator failures based on using several coordinators with different identifiers. Our proposal consists in switching to a different network by changing the network identifier to achieve a lower downtime and we have experimentally verified a recovery which is up to 15 s faster than another solution present in the literature. The lower downtime further reduces the impact of a coordinator failure on the synchronization error. We tested the efficacy of our solution with an experimental set-up using commercial hardware such as the XBee ZigBee modules and Arduino boards. Further developments of this work involve improvements of the synchronization algorithm to more energy efficient solutions. The SPF solution can be extended to a more general case of creation of an ad-hoc WSN with mobile gateways such as, for example, those obtained using drones.

Acknowledgements This work has been partially funded by the EIT Digital ACTIVE Project.

References

1. V. Butenko, A. Nazarenko, V. Sarian, N. Sushchenko, and A. Lutokhin, "Applications of wireless sensor networks in next generation networks," Telecommunication Standardization Sector of ITU, (2014). Available at: https://www.itu.int/dms_pub/itu-t/opb/tut/T-TUT-NGN-2014-PDF-E.pdf
2. M. A. Mahmood, W. K. G. Seah and I. Welch, "Reliability in wireless sensor networks: a survey and challenges ahead," Computer Networks, vol. 79, pp. 166–187 (2015).
3. Lo, Benny PL, Surapa Thiemjarus, Rachel King, and Guang-Zhong Yang. "Body sensor networka wireless sensor platform for pervasive healthcare monitoring.", pp. 77-80, (2005).
4. Xu, Ning, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. "A wireless sensor network for structural monitoring." In Proceedings of the 2nd international conference on Embedded networked sensor systems, pp. 13-24. Acm, (2004).
5. G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees and M. Welsh, "Deploying a wireless sensor network on an active volcano, IEEE Internet Computing," vol. 10, pp. 18-25 (2006).
6. G. C. Gautam and N. C. Kaushal, "Quantitative and qualitative analysis of time synchronization protocols for wireless sensor networks," International Journal of Sensors, Wireless Communications and Control, vol. 4, pp. 2-19 (2014).
7. ZigBee RF4CE specification version 1.00. Website. http://www.zigbee.org.
8. HART Communication Foundation (HCF). WirelessHART Communication Standard. HART 7.0 Specifications, 2007.
9. "6LoWPAN working group," http://www.ietf.org/dyn/wg/charter/6lowpan-charter.html.
10. D.-M. Han and J.-H. Lim, "Smart home energy management system using IEEE 802.15.4 and ZigBee," IEEE Transactions on Consumer Electronics, vol. 56, pp. 1403-1410 (2010).
11. A. Willig and H. Karl, "Data transport reliability in wireless sensor networks - a survey of issues and solutions," J. Praxis der Informa-tionsverarbeitung und Kommunikation, vol. 28, pp. 86-92 (2005).
12. R. Klln and A. Zimmermann, "Transparent coordinator failure recovery for ZigBee networks," in Proc. Conference on Emerging Technologies & Factory Automation (EFTA), Mallorca, Spain, pp. 1-8 (2009).
13. N. Baker, "ZigBee and Bluetooth: strengths and weaknesses for industrial applications," Computing and Control Engineering 16.2, pp. 20-25 (2005).
14. B. Kaur and K. Amandeep, "A survey of time synchronization protocols for wireless sensor networks," International Journal of Computer Science and Mobile Computing, vol. 2, pp. 100-106 (2013).
15. A. Nayyer, M. Nayyer, L. K. Awasthi, "A comparative study of time synchronization protocols in wireless sensor network," International Journal of Computer Applications, vol. 36, pp. 13-19 (2011).
16. J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," ACM SIGOPS Operating Systems Review 36.SI, 147-163 (2002).
17. D. Djenouri and M. Bagaa, "Implementation of high precision synchronization protocols in wireless sensor networks," in Proc. of Wireless and Optical Communication Conference (WOCC), Newark, NJ, pp. 1-6 (2014).
18. F. Dobslaw, Z. Tingting, and G. Mikael "QoS assessment for mission-critical wireless sensor network applications," in Proc. of Conference on Local Computer Networks (LCN), Sydney, NSW, pp. 663-666 (2013).
19. Alliance, ZigBee. "ZigBee-2007 Layer PICS and Stack Profiles 6, "ZigBee Document 08006r03, Rev 3 (2008).
20. Ganeriwal, Saurabh, Ram Kumar, and Mani B. Srivastava, "Timing-sync protocol for sensor networks." Proceedings of the 1st international conference on Embedded networked sensor systems. ACM, (2003).
21. Digi International Inc. http://www.digi.com/
22. XCTU: Next generation configuration platform for XBee, Digi International Inc. Available at: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/xctu
23. D. Scazzoli, A. Kumar, N.Sharma, M.Magarini, G. & Verticale, "A novel technique for ZigBee coordinator failure recovery and its impact on timing synchronization." In Personal, Indoor, and Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium on (pp. 1-5). IEEE.