

Energy-efficient node selection in application-driven WSN

Bruno Marques^{1,2}  · Manuel Ricardo²

Published online: 19 January 2016
© Springer Science+Business Media New York 2016

Abstract The growth of wireless networks has resulted in part from requirements for connecting people and advances in radio technologies. Wireless sensor networks are an example of these networks in which a large number of tiny devices interacting with their environments may be inter-networked together and accessible through the Internet. As these devices may be scattered in an unplanned way, a routing protocol is needed. The RPL protocol is the IETF proposed standard protocol for IPv6-based multi-hop WSN. *RPL* requires that communication paths go through a central router which may provide suboptimal paths, not considering the characteristics of the applications the nodes run. In this paper is proposed an *Application-Driven* extension to *RPL* which enables to increase the WSN lifetime by limiting the routing and forwarding functions of the network mainly to nodes running the same application. As nodes may join a network at a non predictable time, they must be synchronized with respect to their application duty cycles. Therefore, nodes have to wake up and sleep in a synchronized way. In this paper it is also proposed such synchronization mechanism. The results confirm that the proposed solutions provide lower energy consumption and lower number of packets exchanged than the conventional

RPL solution, while maintaining fairness and the packet reception ratio high.

Keywords Wireless sensor network (WSN) · Energy efficiency · Nodes synchronization · Cross-layer

1 Introduction

The growth of wireless networks has resulted in part from requirements for connecting people and advances in radio technologies. Wireless personal networks (WPANs) are an example of these networks, and wireless sensors networks (WSN) [2] are an example of WPANs. Several communications protocols have been defined making use of the IEEE 802.15.4 Physical and MAC layers [18], being the 6LoWPAN Network Layer adaptation protocol [14] an example which bridges the gap between low power devices and the IP world. Since its release, the design of routing protocols became increasingly important [40] and *RPL* [42] emerged as the IETF proposed standard protocol for IPv6-based multi-hop WSN.

Our work is focused on the design of an extension to the *RPL* routing protocol with the purpose of making the network aware of the traffic generated by applications. We assume that sensors form a large IPv6 network and that a sensor is enabled to run one or more applications. We also assume that the applications and the nodes to which they are associated are not always active, alternating between *on* and *off* states. By jointly considering the neighbors of each node, the applications that each node runs, and the forwarding capabilities of a node, we developed a communications solution which enables the data of every application and node to be transferred while keeping the overall energy consumed by the network low. Our solution,

✉ Bruno Marques
bmarq@estgv.ipv.pt

Manuel Ricardo
mricardo@inesctec.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

which we named *RPL-BMARQ*, assumes that every node will primarily select its parent from a set of nodes running the same application to which the data is associated. For that purpose, the application layer of each node shares information with other layers of the communication stack. Since the nodes may join the network at a non predictable and different times, they 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 a synchronization mechanism is also implemented in the proposed solution which will help the nodes to wake up and to go asleep in a synchronized manner so they can successfully send, receive, and forward packets.

1.1 Our contributions

This paper provides two contributions using *cross-layer* concepts. The first contribution is characterized as an extension to the RPL routing protocol, which uses information shared by the application and routing layers to construct *Directed Acyclic Graphs* (DAGs), allowing the nodes to select parents by considering the applications they run. The second contribution is a synchronization mechanism, which synchronizes nodes with respect to their application life cycles, enabling nodes to wake up and going asleep in synchronism. Both contributions help reducing the network energy consumption since they restrict radio communication activities and synchronize the nodes, while maintaining fairness and packet reception ratio high.

The paper is organized in six sections. Section 2 presents the related work. Section 3 describes the rationale of our solution. Section 4 characterizes the applications and scenarios selected for the study, describes the methodology adopted for validating the proposed solution, and discusses the results obtained. Section 5 presents and characterizes the scenarios selected for a real testbed implementation, and discusses the results obtained. Finally, Sect. 6 draws the conclusions and presents future work.

2 Related work

We classify the related work in four areas: node energy consumption, routing protocols, node synchronization, and cross-layering.

2.1 Energy consumption

Sensor nodes are usually battery powered and deployed in large areas in which changing or replacing batteries may be impractical or even unfeasible. Therefore, minimizing the power consumption in a node is a primary issue to be

considered, and the use of solutions for increasing the nodes lifetime is fundamental in WSN. It is well known that energy consumed by nodes in data sensing or processing functions may be negligible [35]. In contrast, data communication has a strong impact on the nodes battery mainly because of two aspects: (1) the radio transceiver implies a high power consumption when compared to the other components of the node; and (2), the communications phase is associated with phenomena such as collisions, overhearing, overemitting, and idle listening, which substantially reduce the nodes battery [7]. Furthermore, in a WSN, each node plays the dual role of data originator and data router. The failure of a few nodes can cause topological changes and might require rerouting of data packets and reorganization of the network. In this regard, power conservation and management take on additional significance. Therefore, some discussion and models on energy efficiency in WSN can be found on the literature, where technical approaches for prolonging the lifetime of battery-powered sensors have been the focus. These approaches include energy-aware protocol development and hardware optimizations, such as power consumption models for WSN devices and sleeping schedules to keep electronics inactive most of the time. Pantazis et al. [35] provide a good overview on this topic. They discuss a number of developed energy efficient routing protocols and models, and provide directions to select the most appropriate to be used in different WSN applications.

With respect to energy consumption, G. Anastasi et al. [3] present a survey on energy consumption based on the hardware of a typical sensor node. They divide the sensor node into four main components: a sensing subsystem including one or more sensors for data acquisition, a processing subsystem including a microcontroller and memory for local data processing, a radio subsystem for wireless data communication, and a power supply unit. This architecture and its associated power breakdown are used to propose a solution for reducing consumption in WSN. They also describe different approaches for energy management and conclude that the energy consumption of the radio is much higher than the energy consumption due to data sampling or data processing. Dunkels et al. [12], in the same approach, propose Eq. 1 which models the energy consumption based on the hardware components of a typical sensor node, E (J), as

$$E = (I_m \times t_m + I_l \times t_l + I_t \times t_t + I_r \times t_r + \sum_{i=1}^n I_{c_i} \times t_{c_i}) \times V \quad (1)$$

where I_m is the current consumed by the microprocessor in the time t_m during which the microprocessor is running, I_l and t_l are the current and time when the microprocessor is

in low power mode, I_t and t_t are the current and the time when the device communication is in transmit mode, I_r and t_r are the current and the time when the device communication is in receive mode, I_{c_i} and t_{c_i} are the current and the time consumed by other components (e.g. LEDs, ADCs, DACs), and V the sensor supply voltage. This equation is used in some operating systems such as Contiki [11] to estimate the energy consumption of a node when it reduces the energy consumption by powering off the microcontroller or other hardware components when they are not used.

One technique that tries to increase the nodes lifetime uses the *design of duty cycle schemes* to schedule the nodes radios states depending on network activity. According to G. Anastasi et al. [3] duty cycling can be achieved through two different and complementary approaches: *power management*, and *topology control*. In the case of the *power management* approach, it is possible to exploit node redundancy, and adaptively select only a minimum subset of nodes to remain active for maintaining connectivity; nodes that are not currently needed for ensuring connectivity can go to sleep and save energy. In the case of the *topology control* approach, finding the optimal subset of nodes that guarantee connectivity is the main objective. Therefore, the basic idea behind *topology control* is to exploit the network redundancy to prolong the network longevity. Moreover, active nodes do not need to maintain their radio continuously on. They can switch off the radio when there is no network activity, thus alternating between sleep and wakeup periods. These two approaches are complementary and implement *duty cycling*. The concept of *topology control* is associated with that of network redundancy. Large WSN typically have some degree of redundancy. In many cases network deployment is done at random. Therefore, it may be convenient to deploy a number of nodes greater than necessary to cope with possible node failures occurring during or after the deployment. *Topology control* protocols are thus aimed at dynamically adapting the network topology, based on the application needs, so as to allow network operations while minimizing the number of active nodes. There are some criteria to decide which nodes to activate/deactivate, and when. So, *topology control* protocols are classified in two categories: *location driven* protocols, and *connectivity driven* protocols. *Location driven* protocols define which node to turn on and when, based on the location of sensor nodes which is assumed to be known. *Connectivity driven* protocols, dynamically activate/deactivate sensor nodes so that network connectivity, or complete sensing coverage [23], are fulfilled. A detailed survey on topology control in wireless ad hoc and sensor networks is available in [38]. The concept of *power management* is associated with *sleep/wakeup* schemes which can be defined for a the radio

subsystem of the sensor node, without relying on topology or connectivity aspects. *On-demand* protocols are examples. The basic idea is that a node should wakeup only when another node wants to communicate with it. The main problem associated is how to inform the sleeping node that some other node is willing to communicate with it. A possible solution consists in using a *scheduled rendezvous* approach. The basic idea is that each node should wake up at the same time as its neighbors, according to a wakeup schedule, and remain active for a short time interval to communicate with their neighbors. Then, they go sleep until the next rendezvous time. The major advantage of such scheme is that when a node is awake it is guaranteed that all its neighbors are awake as well. This allows sending link-local multicast messages to all neighbors. On the opposite, scheduled rendezvous schemes require nodes to be synchronized in order to wake up at the same time. This scheme, as others such as *On-demand* and *Asynchronous* are surveyed and discussed in [3].

Other techniques that try to increase the nodes lifetime use *data-driven approaches*, such as the *data aggregation* scheme. This scheme tries to address the case of unneeded samples, aimed at reducing the energy spent by the sensing subsystem. Some of these schemes can also reduce the energy spent for communication as well, as they reduce the amount of data to be delivered to the sink node. Data aggregation is achieved at intermediate nodes between the sources and the sink to reduce the amount of data traversing the network towards the sink. The most appropriate data aggregation technique depends on the specific application and must be tailored to it. Therefore, being application-specific, in [13] we can find a comprehensive and up-to-date survey about them.

As a conclusion, *Duty cycle schemes* and *data-driven approaches* help reducing the nodes energy consumption by placing nodes sleeping as much time as possible, and reducing the amount of data to be transferred between the nodes. Equation 1 captures well the energy consumed by a node as it explicitly addresses the times associated to the transmission and reception of packets.

2.2 Routing

In the recent past, the ZigBee Alliance introduced a communication stack for wireless sensor networks meeting the typical requirements of low data-rate lossy links interconnecting low-power devices. Based on IEEE 802.15.4, ZigBee [33] specifies the application and network layers. The application layer framework consists of a set of application objects. The ZigBee network layer defines how the network is formed and how the network address is assigned to each participating node. Two routing schemes are available in ZigBee networks: mesh routing, and tree

routing. The mesh routing is similar to the AODV [19] routing algorithm, while the tree routing scheme resembles the cluster tree routing algorithm described in [15]. However, ZigBee was not able to easily plug that kind of networks into the IP-based Internet [1]. As a matter of fact, the *Internet Engineering Task Force* (IETF) 6LoWPAN addresses control and sensor networks working over wireless technologies [10], having designed a standard which defines how IP communications are performed over low-power WPAN [14] and it uses IPv6. 6LoWPAN [31] addresses the challenge of enabling wireless IPv6 communications over wireless sensor networks. A question which often arises is the following: is not the IP protocol too big for low-bandwidth networks? According to [16], resource conservation is the key factor for low-power wireless sensor applications. The resources that matter to achieve deployment ubiquity, long-lived power autonomy, and cost effective devices include: low protocol overhead over the wireless links, low program and data memory requirements, and low power usage in intermittent and infrequent sensor operation. On all these dimensions 6LoWPAN achieves efficiencies comparable to those obtained by non-IP based architectures such as ZigBee, while offering the benefits of end-to-end communication to a huge range of devices. Recently, with the increasing trend towards the *Internet-of-Things*, the ZigBee alliance designed the *ZigBee IPv6-based stack* [39] for 802.15.4 networks. This changed the “traditional” ZigBee stack to use 6LoWPAN, to use RPL as routing protocol, and to use UDP.

RPL [42] is a IPv6 routing protocol that organizes nodes along a *Destination Oriented Directed Acyclic Graph* [29] (DODAG), normally rooted at a border router node or at a sink node. As described in [43], the DODAG root initiates the DODAG formation by periodically sending *DODAG Information Object* (DIO) messages which it advertises by using link-local multicast addresses [4]. DIO messages carry information such as the DODAG root’s identity, the routing metrics in use, and the rank of the originating node *rank* in the DODAG. A node joins the DODAG by taking into consideration these factors, and determines its own rank in the DODAG based on the information advertised by its neighbors in their DIOs. The node chooses as parent in the DODAG the neighbor having the smallest rank. Once a node has joined the DODAG, it sets a path to the root through its parent and starts generating its own DIO messages. RPL provides paths from nodes to a root while requiring these nodes to store little forwarding and routing information. In order to keep the size of forwarding and routing tables small, RPL does not provide by default paths from the border router back to the nodes. To overcome this issue, a RPL router that requires a path from itself to a node must send a *Destination Advertisement Object* (DAO)

message all the way which will lead to the installation of the path to the node. Moreover RPL provides paths that are often much longer than the shortest available paths and the constraint to route only along a DODAG may potentially cause traffic congestion near the DODAG root [4]. Also, the constraint for every possible destination in the DODAG to originate a DAO may be a problem because it is a proactive destination-initiated process which involves sending and receiving many control messages [4].

2.3 Node synchronization

WSN are energy-limited so usually the nodes cannot keep radios active during all the time, having to sleep and to wake up periodically [26]. Addressing this issue, there were proposed several MAC protocols which were categorized into *synchronous* MAC protocols and *asynchronous* MAC protocols. Although asynchronous protocols are simple, they tend to consume more energy because they use a long preamble to ensure that receivers can get the packet. In WSN where energy must be saved, a different approach may be used. One possible way is to use *synchronous* methods. In the literature we can find several proposed synchronous methods, namely the S-MAC [44] and the T-MAC [41]. These protocols transmit a *SYNC* packet to notify neighbors of 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 [26]. Synchronous methods can be characterized as one-way method. Normally, 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 [26] is proposed a synchronous method 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 for all the detecting clocks, 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 is simpler and more energy-efficient than the traditional synchronization method [26].

2.4 Cross-layer

According to [22], cross-layer optimization in WSN has been addressed by multiple studies in different scenarios. The main idea is to design communications layers such that they can share and react to information from other layers. In recent years cross-layer design was used to increase the

efficiency of WSN communication systems. In [30] cross-layering for WSN is surveyed. WSNs consider the *application*, *network*, *medium access control*, and *physical* layers. Routing protocols attract special attention by their impact on the network lifetime. In the cross-layer design proposed in [32], a new adaptive MAC (A-MAC) requires a change in the used routing protocol. New metrics that depend on the duty cycle (the ratio between a sensor active and sleep times) influence the routing decisions. A-MAC changes the nodes duty cycle dynamically to achieve a predetermined network lifetime.

3 RPL-BMARQ

WSN, being constituted by sensor nodes which are known to be energy constrained, depends on their nodes lifetime. Thus, in order to reduce nodes energy consumption, routing strategies capable of finding energy-efficient paths are demanded. We assume that routing protocols must find routes in which the nodes may be kept asleep the maximum amount of time they can. For that purpose, we use the concept of application duty cycle time, characterized by states wake and sleep, and their times. We also assume that WSN forms a mesh network, and that nodes may run multiple applications. Two main questions then arise: (1) How to use mainly the nodes running the application associated to the data being transferred by the network, so that the nodes associated with other applications can continue sleeping? (2) How to synchronize nodes so that they can wake and going asleep simultaneously?

We define *Application-Driven WSN* (ADWSN) 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 exchanged information can be confined to the nodes associated with the application. The nodes share information about the applications they run, and also their duty-cycles. Our solution, *RPL-BMARQ*, stands for RPL By Multi-Application ReQuest. It tries to insure that data of an application is relayed mainly by the nodes running that application. When sink nodes query the other nodes, routing paths should involve preferentially nodes running the same application. 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 DAG created. This DAG is created and maintained by a change in the RPL protocol scheme which will choose

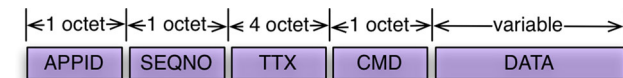
mainly the nodes running that application as parent; the nodes not associated to this application will not be selected as parent, in a first attempt. The nodes are put asleep when there is no activity related to their applications. When nodes receive a query packet they know exactly when they must wake up on the next period. In our solution the nodes alternate between the wake and sleep states. The amount of time of each phase is determined by the applications duty cycle. When a node is awake it performs activities including waiting for a sink query and forwarding packets to neighbors. When the wake up time expires, the node switches to the sleep state, waking up again by the time computed by the proposed synchronization mechanism.

3.1 Cross-layer information

RPL-BMARQ uses application layer information in order to create DAGs and to synchronize the nodes using a synchronous method mechanism. Each *ICMPv6 RPL DIO* message has information about: (1) the application the node runs and application duty-cycle, i.e. the time cycle of the application and the time the nodes are expected to be woken; (2) the number of neighbors; (3) the number of neighbors running this application. This information is used by a node to maintain its *neighbor table* (see Table 1), to maintain routing tables and to create and maintain DAGs. From a *neighbor table* a node knows its neighbors IPv6 addresses (global and link-local), what kind of node they are (DAG root, sink, sensor, or other), what applications they run, and their correspondent duty-cycles. Figure 1 shows the *DAG metric container object* used by *RPL-BMARQ* which needs to be included as an extension to any *RPL objective function metric container object*. A *DAG metric container object* [42] is a *RPL control message option* used to compute the rank of a node, and to help the selection of the best parent. In *RPL-BMARQ*, this *metric object* includes information from the application layer with respect to: the identification of the application that the node runs (*APPID*), which may also be used to identify groups of applications with the same duty cycles so that multiple applications running with the same duty cycle can be included in the same DAG; the total application cycle time (*TCYCLE*); the total time the node is expected to be waked (*TON*); the number of neighbors that the node has (*NBR1*); and the number of neighbors which are running his application (*NBR2*). The *metric container object* is mandatory for the *RPL-BMARQ* solution because it carries all necessary information to create and maintain DAGs and *neighbor tables*. Each time a query packet is sent by sinks the packet is “disseminated” into the network according to the *RPL-BMARQ* routing mechanism. This packet (see Fig. 2) is constituted by the *APPID* field; by the *SEQNO* field; by the *TTX* field; by the *CMD* field; and by the *DATA* field.

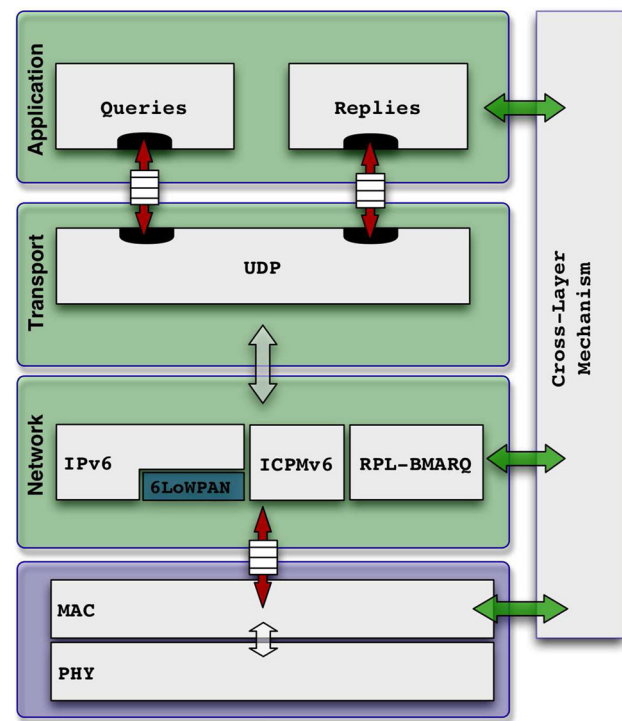
Table 1 RPL-BMARQ neighbor information

IPADDR	LLADDR	TYPE	APPID	TCYCLE	TON	NBR1	NBR2
--------	--------	------	-------	--------	-----	------	------

**Fig. 1** RPL-BMARQ metric container object**Fig. 2** RPL-BMARQ application layer packet

The *APPID* field identifies the application to which the packet corresponds; the *SEQNO* field is used to sequence a packet; the *TTX* field carries the timestamp (this time corresponds to the originator clock time); the *CMD* field specifies the type of the message (query, reply, or other); finally, the *DATA* field contains application data. This information is used not only to know if a packet is to be forwarded in the network layer, but also if it must be replied at the application layer. The *APPID*, *SEQNO* and *TTX* fields are also used by the synchronization mechanism to maintain the nodes synchronized, as described in Sect. 3.3.

Figure 3 shows RPL-BMARQ communications stack. Each time a query is sent by a sink node, the node constructs a message packet with the structure shown in Fig. 2 where in the *APPID* field it identifies the application it runs, in the *SEQNO* field it includes the application data packet query sequence, in the *TTX* field is included the timestamp the query was sent. The *CMD* field contains information which identifies the data packet as a query. This structure is sent through a particular UDP socket and included in the *data* field of a *transport layer* segment. Then, this segment is sent to the *network layer* using an IPv6 link-local address *FE80::* as destination. Since nodes can receive more than one query from neighbors, an incoming query buffer is used by the RPL-BMARQ routing mechanism which will help to decide if the received query is to be forwarded again to other neighbor nodes, to decide if the query is to be discarded in case of already have been received, and to decide if the query is to be replied back. The node consults its neighbor table to see if it has

**Fig. 3** RPL-BMARQ communications stack

neighbors running the application from which the query was received. If there exist at least one neighbor, the query is forwarded again using the same link-local address (except if the query was sent by this neighbor); if not, the packet is discarded. Also, if this node runs the application from which the query was received, the datagram is passed up to the *transport layer* which will use the same earlier UDP socket to send the query packet to the *application level*. At this level, upon the reception of the query message, the node processes it according to the information asked and sends back a reply by constructing a similar message packet (Fig. 2). This message is sent to the *transport layer* using another UDP socket, which will construct a new segment to be used by the *network layer* to send a datagram to the sink global IPv6 address. It has to be noted that queries are sent to a link-local IPv6 multicast address, whereas replies are sent to global IPv6 unicast addresses.

3.2 DAG creation

The DAG creation scheme will use mainly the nodes running the same application; the nodes not associated to the application will not participate in DAG creation process, in a first attempt [28]. Algorithm 1 shows how to create DAGs in our solution. A root node starts to create the DAG by sending DIO messages. The nodes surrounding the root use the information carried in these DIO messages, compute their rank and join the DAG, in the same way as in regular RPL. The computed rank is jointly sent with the identification and duty-cycle of the application, in DIO messages. This information is used by other nodes to update their *neighbor tables*, compute their own rank, and advertise their presence by sending new DIO messages. All node's *neighbor tables* record the neighbors IP address, the application that they run and its duty-cycle. This information is used to help the node to join the DAG, by looking into its *neighbor table*. If the node runs the same application of its neighbor and if the latter has a lower rank, the former may choose it as parent, joining the DAG through it. If the node has neighbors which do not run its application, but have in turn at least one neighbor running this application, one of them can be selected as parent. As example, if the neighbor has two neighbors, one root and the other sink, the root node will be always selected. Therefore, the node will not change parent depending on packets arrival. Otherwise, a node will always select a sink node neighbor as parent. If a node has only sensor nodes as neighbors it will select for parent the sensor running the same application which has lower rank. Moreover, if the sensor has neighbors not running the same application but in turn they have other neighbors which run its application, the former will select as parent a neighbor with lower rank. In other situations, the mechanism switches to regular RPL. This mechanism is introduced inside a *RPL Objective function* to create the DAGs accordingly.

3.3 Synchronization mechanism

It is unlikely that all the sensor nodes would join a network at the same time. Having the nodes active during all the time would deplete their batteries, so nodes have to go sleep and to wake up periodically. All the nodes need to be awake at same times in order to receive sink queries and to forward them to the other nodes. As a result, nodes must be synchronized according to the application cycle they run. 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.

```

if (neighbor == is_root) then
  add_parent(neighbor);
else
  if (neighbor == is_sink && neighbor.rank <
    node->neighbors.rank) then
    add_parent(neighbor);
  else
    if (neighbor.app_id == node.app_id &&
      neighbor.rank < node->neighbors.rank) then
      add_parent(neighbor);
    else
      if (neighbor->neighbor.app_id ==
        node->neighbor.app_id && neighbor.rank <
        node->neighbors.rank) then
        add_parent(neighbor);
        reconfigure(neighbor(app_id,
          app_duty-cycle));
      else
        if (neighbor.rank <
          node->neighbors.rank) then
          add_parent(neighbor);
        end
      end
    end
  end
end
end

```

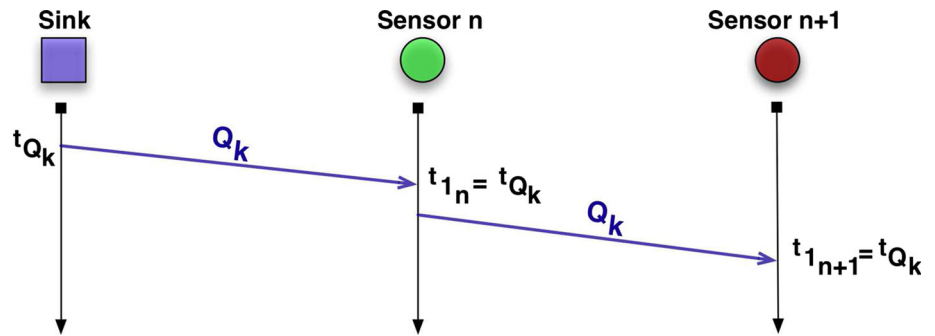
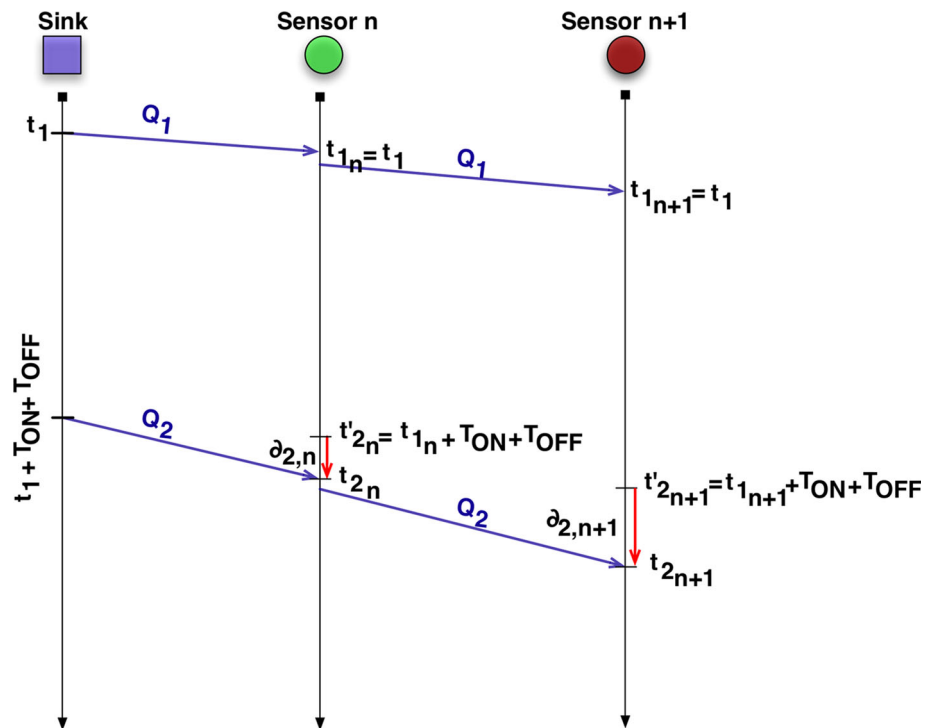
Algorithm 1: Pseudocode of RPL-BMARQ DAG creation executed by a node

3.3.1 The synchronization setup phase

When a 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 (see Fig. 4). This information is extracted from the query packet itself (*TTX* field). It is known that this corresponds to setting the time's nodes to a time value which does not consider the transmission delay nor the propagation delay, but this fact has no impact on the mechanism.

3.3.2 The synchronization maintenance phase

Since all the nodes know the characteristics of the applications they run, after the reception of their first query packet they would expect to receive the second query packet by $t'_2 = t_1 + T_{ON} + T_{OFF}$. 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. 5. 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 $\delta_2 = -2$; if the same node expected to receive the query packet by $t'_2 = 100$ and receives it by $t_2 = 98$, 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. In general, the

Fig. 4 Synchronization setup phase**Fig. 5** Synchronization maintenance phase

difference between the expected time to receive the next query and the time it is really received is computed by Eq. 2

$$t'_k = t_{k-1} + TON + TOFF$$

$$\delta_k = (1 - \alpha) \cdot \delta_{k-1} + \alpha \cdot (t'_k - t_k) \quad (2)$$

where t'_k is the expected packet reception time, and t_k is the real packet reception time. δ_k is evaluated according to an exponential moving average with α reflecting the weight of last observation. The δ_k 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. 3.

$$T_{Sleep_k} = T_{OFF} - \beta \cdot |\delta_k| \quad (3)$$

In this Eq. 3, the β factor is used to amplify the δ_k value in order to guarantee that the nodes will be awake in time in the next application cycle to successfully receive and forward packets, maintaining them synchronized. Following current IETF recommendations for managing TCP timers [36] we selected $\beta = 10$. Algorithm 2 shows how *RPL-BMARQ synchronization mechanism* is implemented.


```

while (bootstrap) do
  wait(app.query);
end
foreach (app.queryk received) do
  if (first(app.query) then
    set_clock(query → TTX);
  else
    if (app.query_id == node.app_id) then
      t'k = tk-1 + TON + TOFF;
      tk = node.queryTRX;
      δk = (1 - α) · δk-1 + α · (t'k - tk);
      TOFF = app.Toff - β · |δk|;
      adjust_sleep_timer(TOFF);
    end
  end
end
end

```

Algorithm 2: Pseudocode of the proposed synchronization mechanism

4 RPL-BMARQ evaluation

4.1 Applications characterization

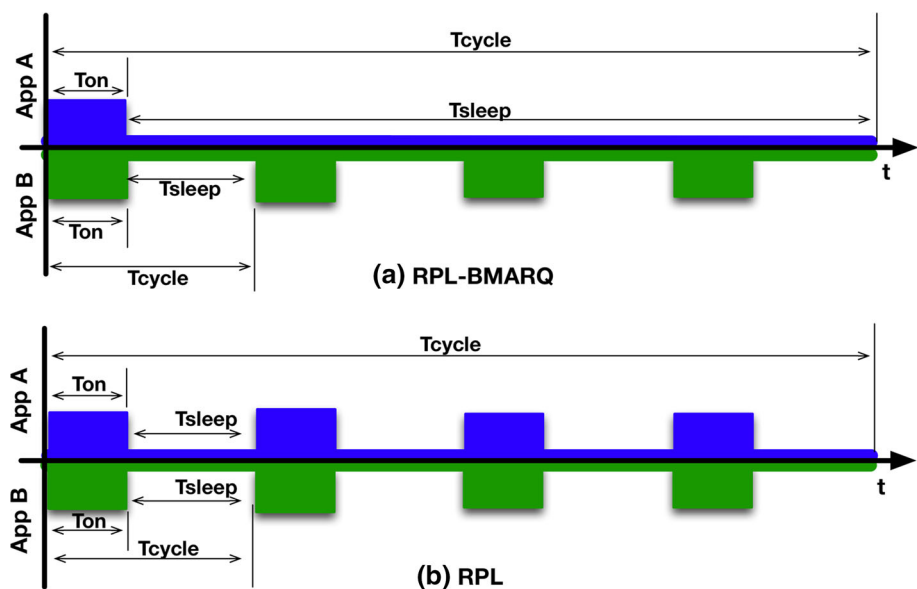
Two different applications are used in our experiments. These applications and their topology are characterized by the following aspects: static, organized, pre-planned, no mobility, 16 nodes deployed in a square lattice topology, all sensor nodes battery-powered, except for the sink nodes. Additionally we consider multi-hop communications, the traffic pattern is point-to-multipoint when data is queried, and point-to-point when queries are replied by nodes. The first application (App. A) has a duty-cycle of 1 h; every sensor node running this application wakes every hour remaining awake during one minute for receiving the query and send data back to the sink. The second application (App. B) has a duty-cycle of 15 min; the sensor nodes also

wakes for one minute to sense and to send data to the sink and to communicate. As shown in Fig. 6a, the period of App. A is 4 times the period of App. B. All sensor nodes are expected to be awake when data is queried and replied, and sleeping when there is no activity.

4.2 Scenarios studied and evaluated

In order to evaluate the *RPL-BMARQ* solution, a square lattice of 4×4 nodes was used. The nodes are distributed as shown in Fig. 7, where the four scenarios evaluated are also shown. 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. Sink nodes placements were chosen in order to allow long routing paths, since long paths consume more energy. 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; in these scenarios we aim to investigate the influence of the application duty-cycle in energy consumption. *Scenarios 3 and 4* are used to investigate situations where at least one node from other application is required to relay data. In the scenarios simulated, sink nodes are always awake, and sink node running App. B (node 9) was chosen as DAG root because of its application duty-cycle. Since our solution constrains the paths to the nodes associated to the application, a node needs to be woken up only when its applications run and not for generic routing and forwarding purposes. In contrast, RPL was used as shown in Fig. 6b; in this case despite the applications having different periods, all the nodes would have to wake up every 15 min in order to

Fig. 6 Applications activity cycle



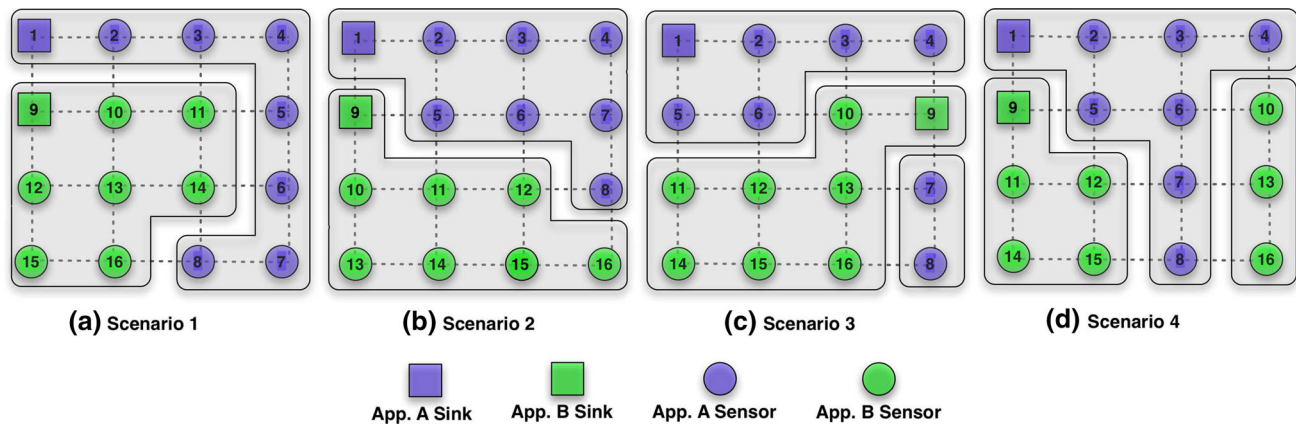


Fig. 7 Nodes deployment in different square lattice mesh topologies

process routing messages. In our solution, a query is “multicast” only to the nodes associated to the application and not the entire WSN. When the application nodes reply, only the nodes running the application will send and forward *unicast* packets. So, using the *RPL-BMARQ* solution, the routing paths are chosen not only according to RPL objective functions, but also considering the nodes belonging to the application for which the paths are required, and the total number of neighbors a node has, at least in the first attempt. Figure 7c shows a case of node deployment where some of the nodes of App. A (nodes 7 and 8) are unable to receive sink queries because they are isolated. Figure 7d shows a *very particular* node deployment, where nodes 10, 13, and 16 running application B are unable to receive sink queries because they are isolated. In this case, node 8 would be selected to participate in the routing and forwarding process, being selected by node 16 as parent. This last scenario raises a routing issue and it is discussed separately in Sect. 4.5.

In order to evaluate the performance gains of our solution, first we present a theoretical study to estimate the magnitude of energy consumption improvements introduced by our solution. Then, we present simulation results of four scenarios each simulated 10 times and, finally, we present the results from two real testbed implementations.

4.3 Theoretical evaluation

4.3.1 Packet energy consumption

Considering that a node is implemented as a CrossBow TelosB [9] sensor hardware, and the energy is consumed by its CPU and RF transceiver, the total energy consumed can be described as follows:

$$E = E_{on} + E_{TX_{Bcast}} + E_{RX_{Bcast}} + E_{TX_{Ucast}} + E_{RX_{Ucast}} + E_{Idle} + E_{Sleep} \quad (4)$$

Table 2 TelosB specification

	Nominal
Current in transmit (0 dBm) mode (mA)	19.5
Current in receive mode (mA)	21.8
Current in MCU on, radio off (mA)	1.8
Current in MCU on, radio on-idle mode (μ A)	365
Current in sleep mode (μ A)	5.1
Power supply (V)	3.6
Transmit bit rate (kbit/s)	250
Transmit symbol rate (ksymbol/s)	62.5

where E_{on} is the energy consumed during the time that node is waked, $E_{TX_{Bcast}}$ is the energy consumed when sending “broadcast” packets, $E_{RX_{Bcast}}$ is the energy consumed when receiving “broadcasted” packets, $E_{TX_{Ucast}}$ is the energy consumed when sending unicast packets, $E_{RX_{Ucast}}$ is the energy consumed when receiving unicast packets, E_{Idle} is the total energy consumed when the node is in the idle state (the state where a node has its radio on and waiting to send or to receive a data packet), and E_{Sleep} is the total energy consumed when the node is sleeping. The energy consumed by a node in idle state is computed by $E_{Idle} = I_{Idle}(A) \times V \times t_{Idle}(s)$, considering $I_{Idle} = 365 \mu A$, $V = 3.6 V$, and t_{Idle} the time the node is idle, which depends on the communications scenario. The energy consumed by a sleeping node is computed as $E_{Sleep} = I_{Sleep}(A) \times V \times t_{Sleep}(s)$, considering $I_{Sleep} = 5.1 \mu A$, $V = 3.6 V$, and t_{Sleep} the total time the node is sleeping. The values are extracted from Table 2, extracted from [8].

For theoretical evaluation purposes, we assume the simplest case of having no collisions and all the packets being correctly received. We also assume that a unicast packet is acknowledged at the MAC Layer, whilst a

“broadcast” packet is not. The energy consumed per packet considering the information of Table 2, and the IEEE 802.15.4 specification [17], can be computed as follows.

Transmission of “broadcast” packet: non-beacon enabled IEEE 802.15.4 networks use an unslotted CSMA-CA channel access mechanism [21, 24]. We assume that each time a device needs to transmit, it waits for a random number of unit backoff periods in the range $\{0, 2^{BE} - 1\}$ before performing the Clear Channel Assessment CCA. If the channel is found to be idle, the device transmits. If the channel is found to be busy, the device waits another random period before trying to access the channel again. Assuming the channel is found to be free, and also assuming that the backoff exponent BE is set to $macMinBE$ which has the default value of 3, and the access time can be computed as

$$\begin{aligned} T_{CA} &= InitialBackoffPeriod + CCA \\ &= (2^3 - 1) \times aUnitBackoffPeriod + CCA \\ &= 7 \times 320 \mu s + 128 \mu s \\ &= 2.37 ms \end{aligned} \quad (5)$$

The CCA detection time is defined as 8 symbol periods. $aUnitBackoffPeriod$ is defined as 20 symbol periods, where 1 symbol corresponds to 16 μs .

As shown in Fig. 8, the energy consumed is computed as $E_{TX_{Bcast}} = E_i + E_{P_{TX}}$; E_i is the energy consumed during the Channel Access period (CA) which is $T_{CA} \times P_{Idle}$; P_{Idle} is the power consumed by the node in the idle mode which is 1.31 mW. $E_{P_{TX}}$ is the energy consumed during the time required to send the packet of size S (in octets). $E_{P_{TX}} = S \times T_{octet} \times P_{TX}$; T_{octet} is the time required to send one octet, which is 32 μs , and P_{TX} is the power consumed in the transmission of the same octet, which is 70.2 mW.

Reception of “broadcast” packet: when a node receives a “broadcast” packet of size S , the energy consumed is $E_{RX_{Bcast}} = E_{P_{RX}} = S \times T_{octet} \times P_{RX}$ where T_{octet} is the time required to receive one octet, which has the same value as the time required to send one octet. $P_{RX} = 78.5$ mW is the

power consumed by receiving the same octet, as shown in Fig. 9.

Transmission of unicast packet: when a node sends a unicast packet, the amount of energy consumed is computed as the energy required to transmit the packet plus the energy consumed during the reception of the acknowledge frame. The transmission of an acknowledgment frame in a non-beacon enabled network commences $aTurnaroundTime$ symbols after the reception of the data frame, where $aTurnaroundTime$ is equal to 192 μs . This gives the device enough time to switch between transmit and receive mode.

As shown in Fig. 10, the total energy consumed is $E_{TX_{Ucast}} = E_i + E_{P_{TX}} + E_{T_{Ack}} + E_{M_{Ack}}$; E_i is the energy consumed during the Channel Access period; $E_{P_{TX}}$ is the energy consumed during the time required to transmit the packet of size S ; $E_{T_{Ack}}$ is the energy consumed while waiting for the reception of the acknowledgment, which is 0.252 μJ , and corresponds to $aTurnaroundTime$ times the power consumed in the idle mode which is 1.31 mW; $E_{M_{Ack}}$ is the energy consumed during the time required to receive the complete MAC acknowledgment frame which has a size of 11 bytes, and corresponds to 27.6 μJ .

Reception of unicast packet: the energy consumed to receive a unicast packet of size S is $E_{RX_{Ucast}} = E_{P_{RX}} + E_{T_{Ack}} + E_{M_{Ack}}$. $E_{P_{RX}}$ is the energy consumed during the time needed to receive the packet of size S ; $E_{T_{Ack}}$ is the energy consumed during the T_{Ack} time to wait before sending the acknowledge packet (this value is the same as the waiting time before receiving the acknowledge packet); $E_{M_{Ack}}$ is the energy consumed during the time of the transmission of the acknowledge MAC frame - T_{Mack} times the power consumed in the transmission mode which is 70.2 mW, corresponding to 24.7 μJ , (see Fig. 11).

4.3.2 Energy gain estimation

We define the energy gain of our routing solution as:

$$\frac{E_{RPL} - E_{BMARK}}{E_{RPL}} \times 100 \quad (6)$$

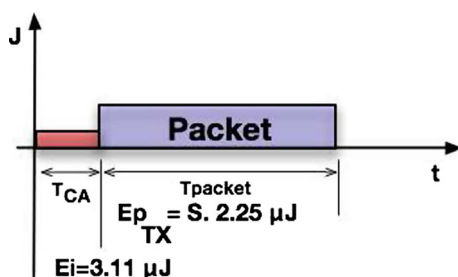


Fig. 8 Energy consumed by a node when transmitting a “broadcast” packet of size S octets

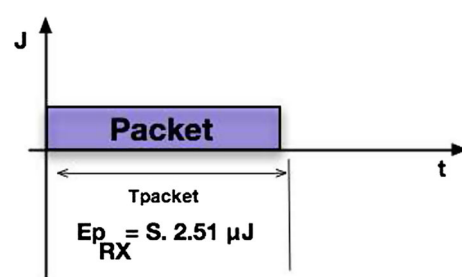


Fig. 9 Energy consumed by a node when receiving a “broadcast” packet of size S octets

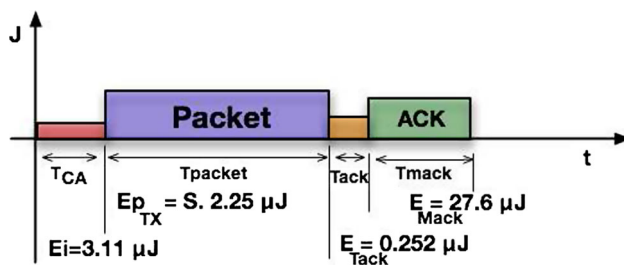


Fig. 10 Energy consumed by a node when transmitting a unicast packet of size S octets

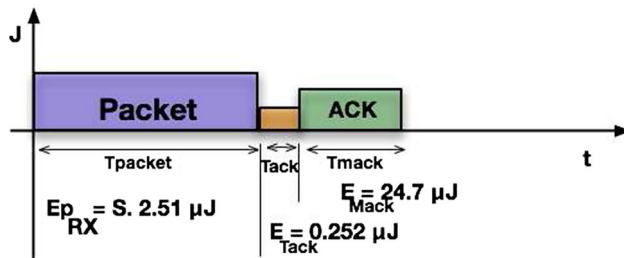


Fig. 11 Energy consumed by a node when receiving a unicast packet of size S octets

where E_{RPL} is the total energy consumed by the nodes for the RPL routing solution, and E_{BMARQ} is the total energy consumed by the nodes for our routing solution. The values are given in %.

4.3.3 DAGs used

Figure 12 shows the DAGs selected for theoretical evaluation. For *RPL-BMARQ* solution we have manually selected the DAGs so that the nodes may send and received packets as expected. For *RPL* the DAGs were selected in order to allow for shortest paths considering the hop count metric.

4.3.4 Results and discussion

We characterized the energy consumed by the nodes running simultaneously both applications, we take into account the number of “broadcast” (more exactly, *link-local multicast*) and unicast packets, and the time the nodes are wake, sleeping, or in idle mode during 1 h. We also compare our solution against the generic RPL routing solution. The analysis was performed based on packets of 127 octets (application data size of 81 octets, plus the IEEE 802.15.4 MAC layer header and PHY layer header size of 46 octets). The packet size chosen reflects worst cases in the analysis performed. MAC layer collisions were not considered. For simplicity we assumed that all packets were sent and received with no errors and no

retransmissions. The calculus was made using a C program that implements both solutions.

Energy: Figure 13 shows the total energy consumed by each node. As can be seen, our solution always consumes less energy. The total of energy consumed is computed as the sum of the energies consumed by individual nodes. The energy consumed by each node is given by Eq. 4, and Fig. 14 shows the total of energy gains using de RPL-BMARQ solution in each scenario. The gain is computed using Eq. 6. Figure 15 shows the energy consumption by each solution in the scenarios studied. As it can be seen, nodes using the *Standard RPL* solution always consume more energy. In the selected scenarios the mean energy consumed for the RPL-BMARQ solution considering the 4 scenarios, is 5.95 J, and for the RPL solution is 8.76 J. The mean gain, considering the 4 scenarios, is 32.7 %. Concerning to the time the nodes are sleeping, results show that for the RPL-BMARQ solution, nodes sleep more time than for the RPL solution. For the selected scenarios, and for the RPL-BMARQ solution, the sum of time the nodes are sleeping is 57,000 s, while for the RPL solution is 56,640 s. In the idle mode energy is also consumed.

Application packets transmitted and received: Figure 16 shows the distribution of the “broadcast” packets generated by each node in the four scenarios. As shown, the number of “broadcast” packets sent using our solution is lower than the number of packets sent using the RPL solution. In scenarios 3 and 4, the *RPL-BMARQ* solution needs node 9 and node 8, respectively, to forward packets. Those nodes forward 5 “broadcast” packets in scenarios 3 and 4. The total number of “broadcast” packets sent in each scenario corresponds to the packets sent when sink nodes issue queries. In our solution, only the nodes belonging to the application forward the packets, thus network flooding is bounded to the nodes of the application. In the case of the RPL solution, when a sink issues a query the packets are “broadcasted” to the entire network. Since there are two applications running, the network is “broadcasted” twice. For example in scenario 1 our solution sends 40 packets, while the RPL solution sends 80 packets. Figure 17 shows the distribution of “broadcast” packets received by each node. As shown, the number of “broadcast” packets received with our solution is lower than the number of packets received using the RPL solution. For the RPL solution, in the scenarios considered, 4 nodes are used more often (nodes 5, 6, 7, and 12). They are placed in the middle of the topology and receive more packets than the others, since they have more neighbors. With the RPL-BMARQ solution, the distribution of the nodes inside the topology has influence. Looking at scenario 1, node 10 receives more packets than the others nodes (16 packets). This node runs application B, thus it is used 4 times per hour. It also receives more packets since it

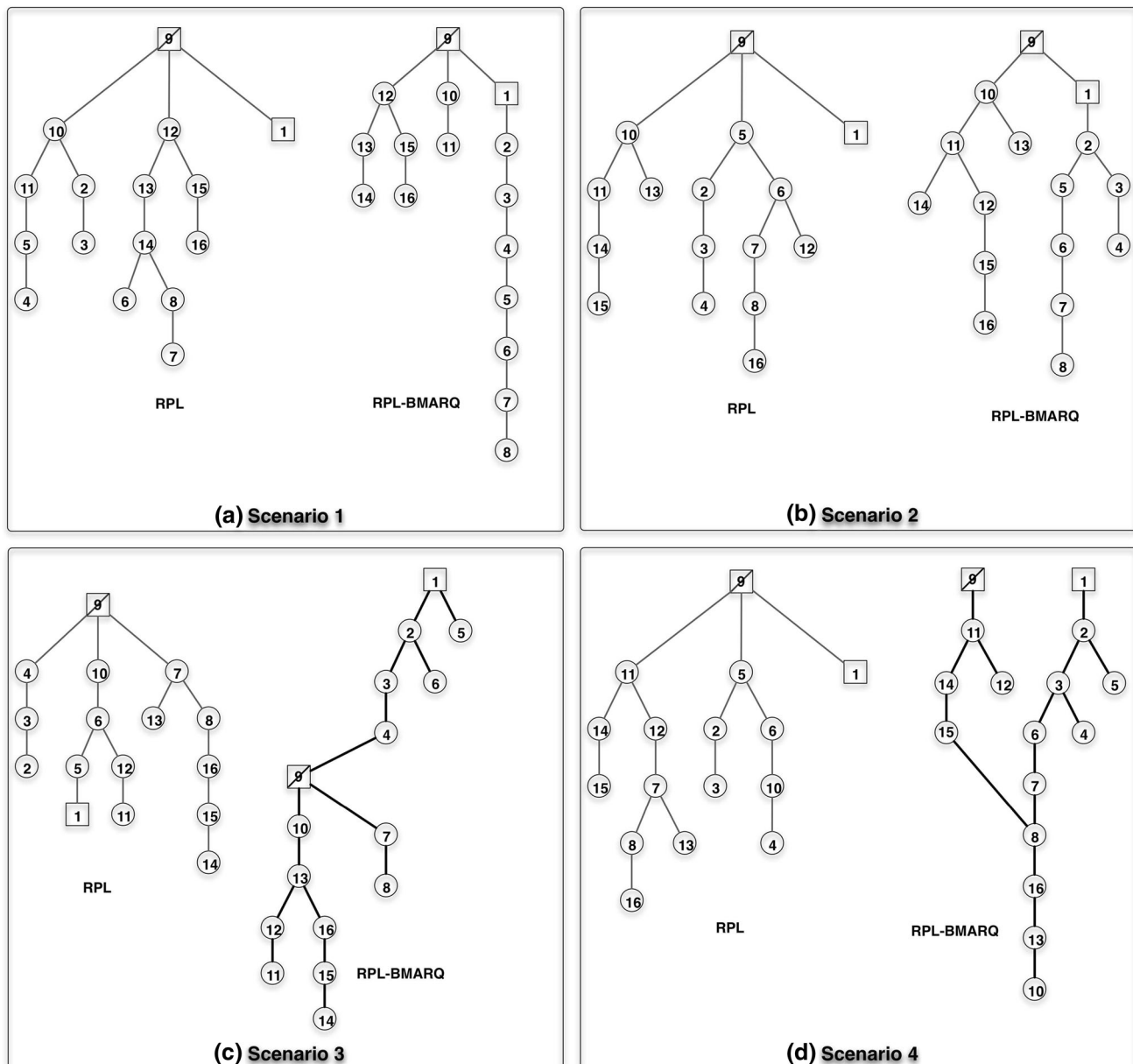


Fig. 12 DAGs used in theoretical evaluation

has more neighbors. In the case of scenario 4, there are 2 nodes (11, 15) which receive respectively 12 and 8 packets, because node 8 relays packets from nodes running application B.

When analyzing Figs. 16 and 17, we also conclude that with the RPL-BMARQ solution application B generates more “broadcast” packets than application A, since queries are four fold. With RPL, all nodes are waked in order to receive and to send “broadcast” packets, so the node distribution does not influence the number of “broadcast” packets. Moreover with RPL, queries are issued 5 times per hour (one from application A, and 4 from application B).

Figures 16 and 17 also show “broadcast” “hotspots”. Figure 18 shows the distribution of unicast packets sent by each node in the four scenarios considered. We can verify that the total number of unicast packets sent in each scenario is lower for RPL-BMARQ than for RPL, except for scenario 4, where RPL-BMARQ needs to send more unicast packets. In this scenario nodes need to forward more packets in order to reply to sink queries. Since RPL-BMARQ uses mainly the nodes belonging to the applications, node 11 is more often used. In the scenarios considered, and analyzing the DAGs used (see Fig. 12) the paths selected by the RPL-BMARQ solution are longer

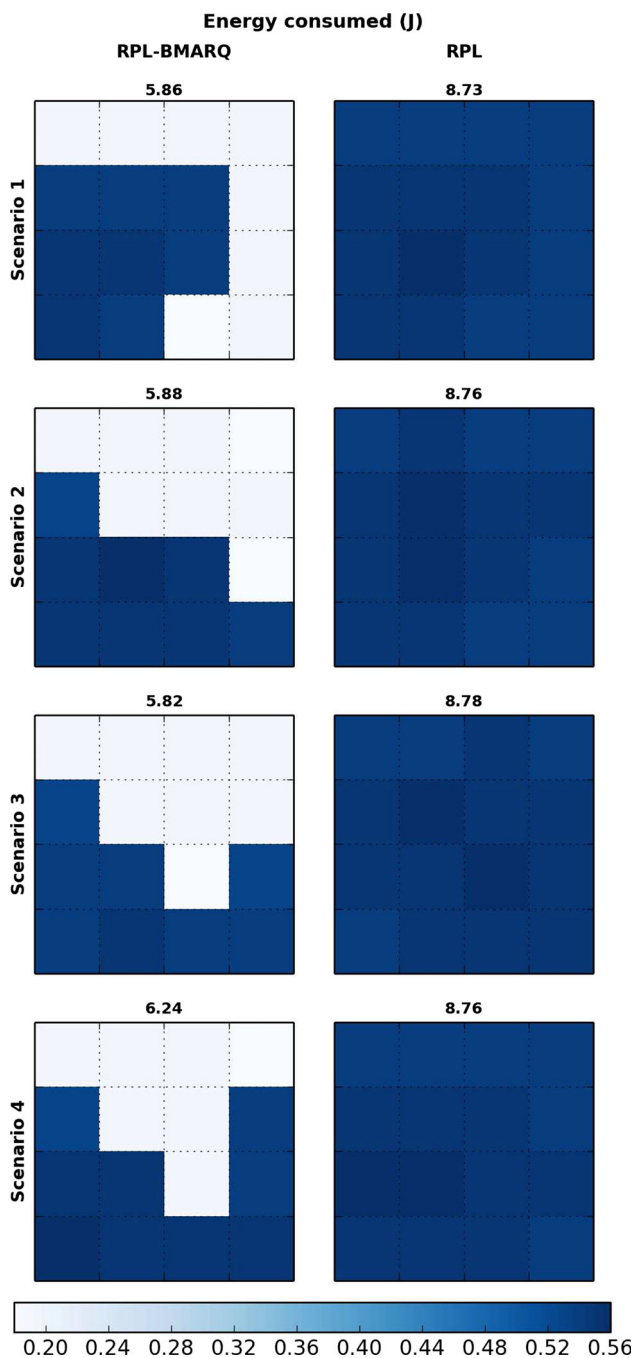


Fig. 13 Total of energy consumed in each scenario for the RPL-BMARQ and the RPL solutions

than those selected by the RPL solution, which makes no distinction between nodes. In the case of scenario 4, node 8 is also used to forward packets from nodes running other application. In this situation, node 8 transmits 13 packets. Since for the RPL solution all nodes must be awake, in scenario 4, node 11 is the node with more activity, and it transmits 26 packets per query. The number of the unicast packets received in our solution is smaller than in RPL, as

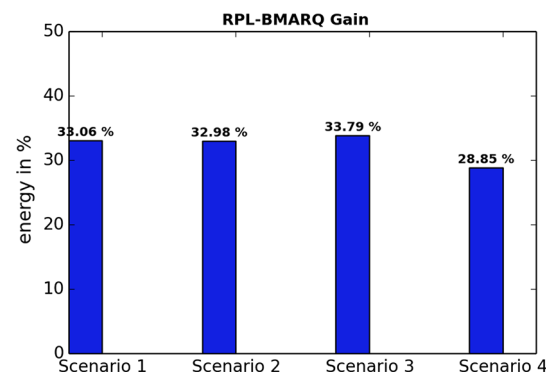


Fig. 14 RPL-BMARQ energy gains in each scenario (in %)

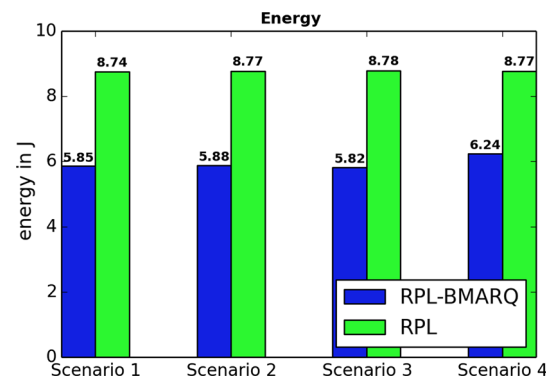


Fig. 15 Energy consumption in each scenario (in J)

shown in Fig. 19). As example, in scenario 1 the total number of unicast packets received for the RPL-BMARQ solution is 191, while for the RPL solution is 290. The results are summarized in Table 3. They show that, for the scenarios studied, our solution provides significant energy gains.

Analysis for large networks: the above theoretical results were obtained for small topologies. But one could ask how the proposed solution could lead to non-optimal path when large networks are involved. Let us take as example a large network, characterized by a square lattice topology of 100 nodes and consider two limit node distributions: (1) 50 nodes running application A and 50 nodes running application B (Fig. 20a), and (2) 10 nodes running application A and 90 nodes running application B (Fig. 20b). These topologies represent large WSNs with different node distributions by applications (ratios 1/1 and 1/9) and, simultaneously, the usage of several nodes of running one application being used to relay packets generated by nodes running the other application. To perform this study we used the same method employed in Sect. 4.3.4. Results are summarized in Table 4. As it can be observed, in both topologies the energy consumed by *RPL-BMARQ* is always lower than the energy consumed by *RPL*.

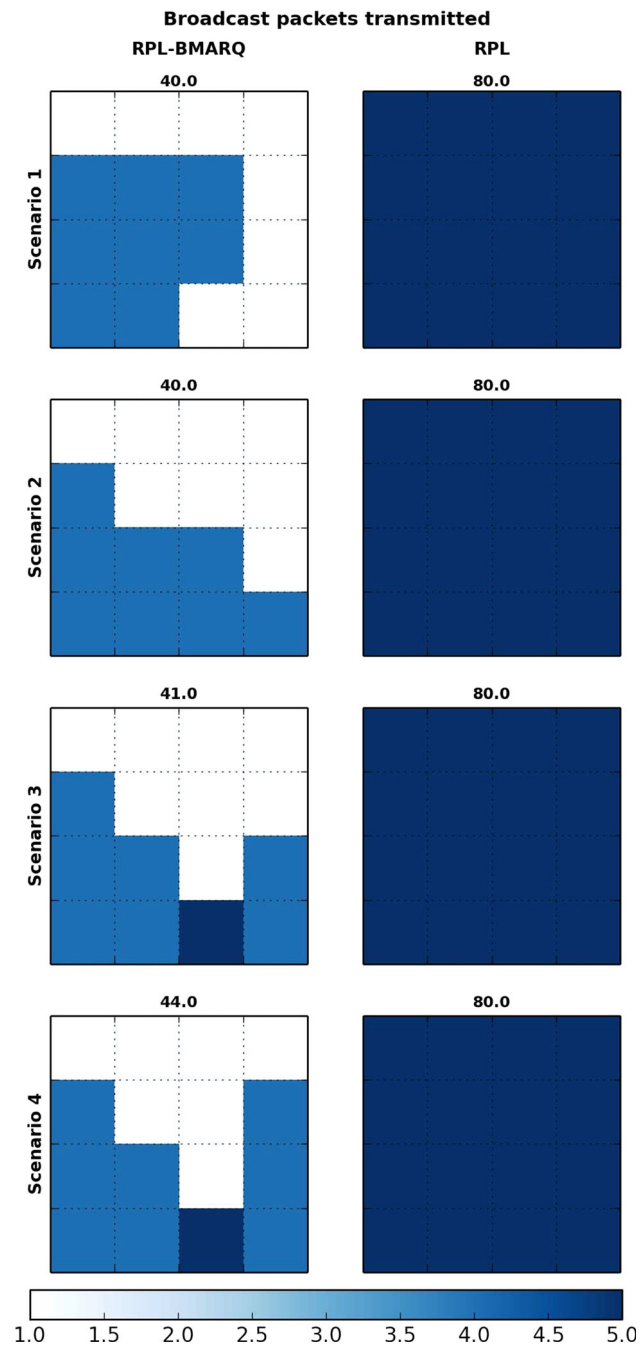


Fig. 16 Total number of “broadcast” packets generated in each scenario for the RPL-BMARQ and the RPL solutions

In Fig. 20a, b we can observe that our solution presents gains respectively of 30 and 90 %. This difference comes from the sensors nodes distributions: in (a) the application distribution ratio is the same (same number of sensor nodes for each application); in (b) 90 % of the sensor nodes run one application which is four times greater than the duty cycle of the application running in the other 10 % of the nodes.

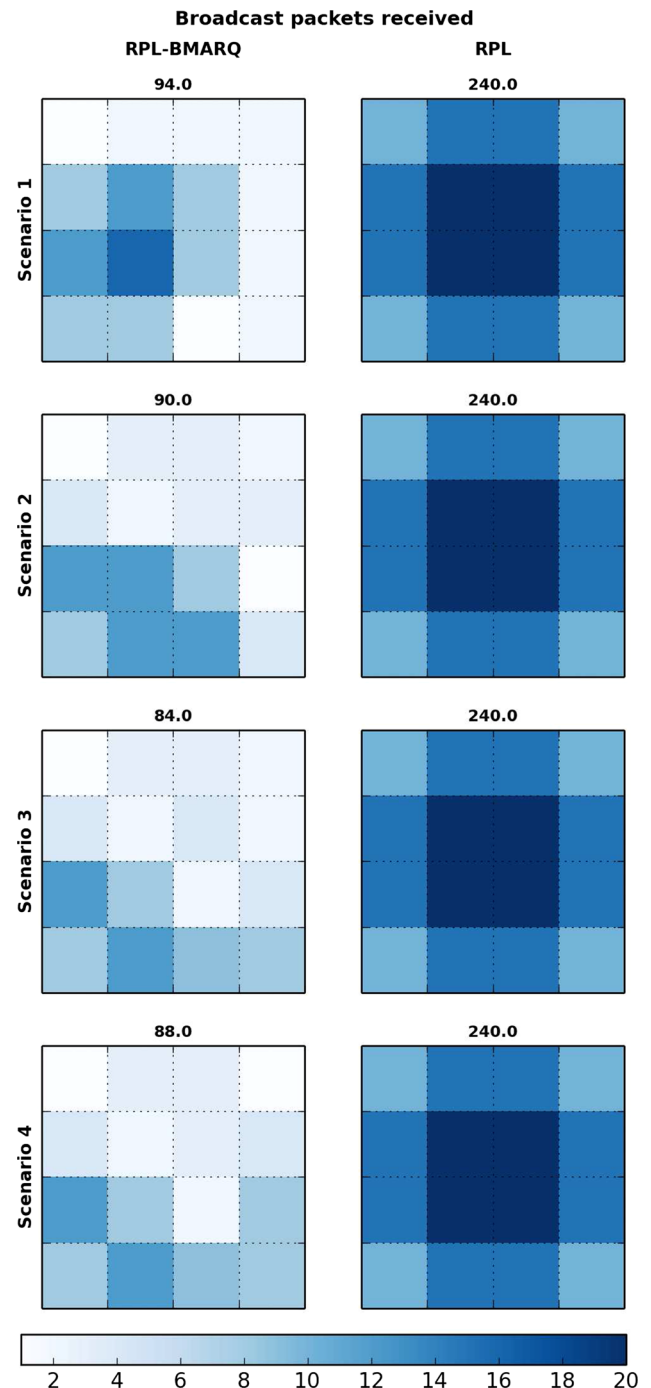


Fig. 17 Total number of “broadcast” packets received in each scenario for the RPL-BMARQ and the RPL solutions

Analyzing the total number of layer 3 packets sent and received, we observed that the *RPL-BMARQ* solution presents lower values than *RPL*. In (a) the total number of broadcast packets received by *RPL-BMARQ* and *RPL* are respectively 848 and 930; the total number of unicast packets received by *RPL-BMARQ* and *RPL* are respectively 5, 339 and 5, 998. With respect to topology (b) the

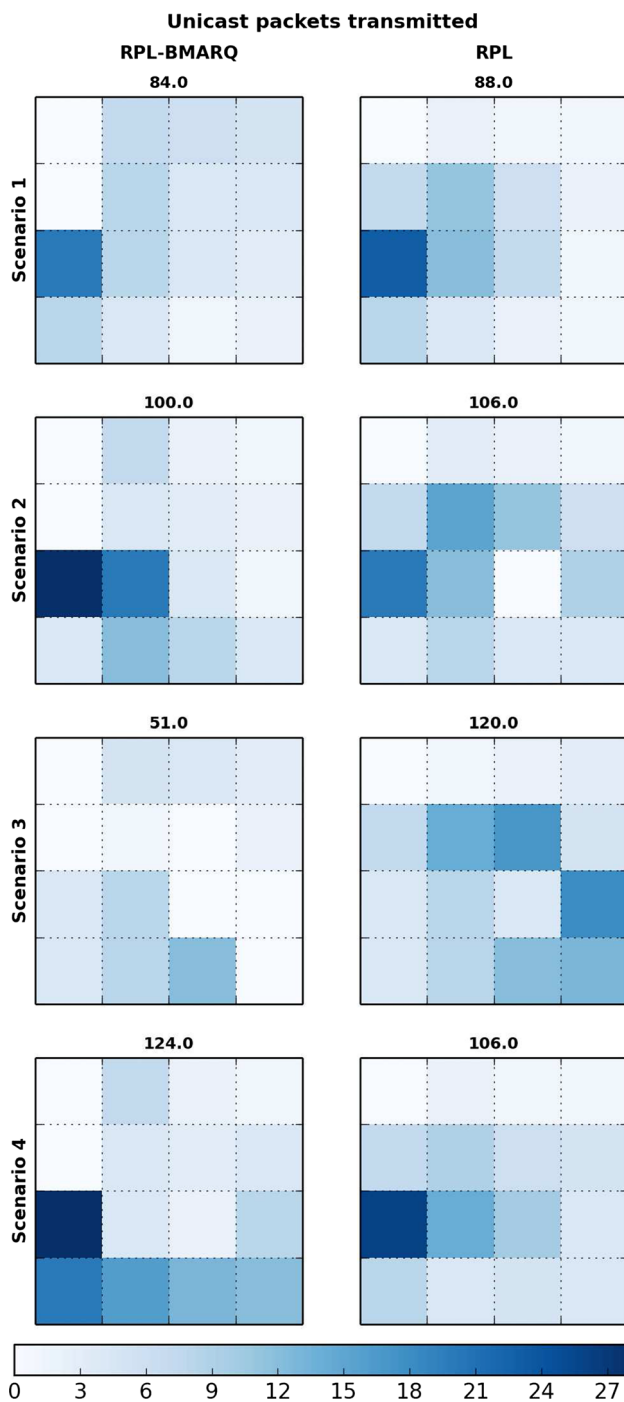


Fig. 18 Total number of unicast packets transmitted in each scenario for the RPL-BMARQ and the RPL solutions

total number of *broadcast* packets sent by *RPL-BMARQ* and *RPL* are respectively 379 and 500; the total number of *broadcast* packets received by *RPL-BMARQ* and *RPL* is are respectively 1, 324 and 1, 800; The total number of *unicast* packets sent by both solutions is the same (5, 607)

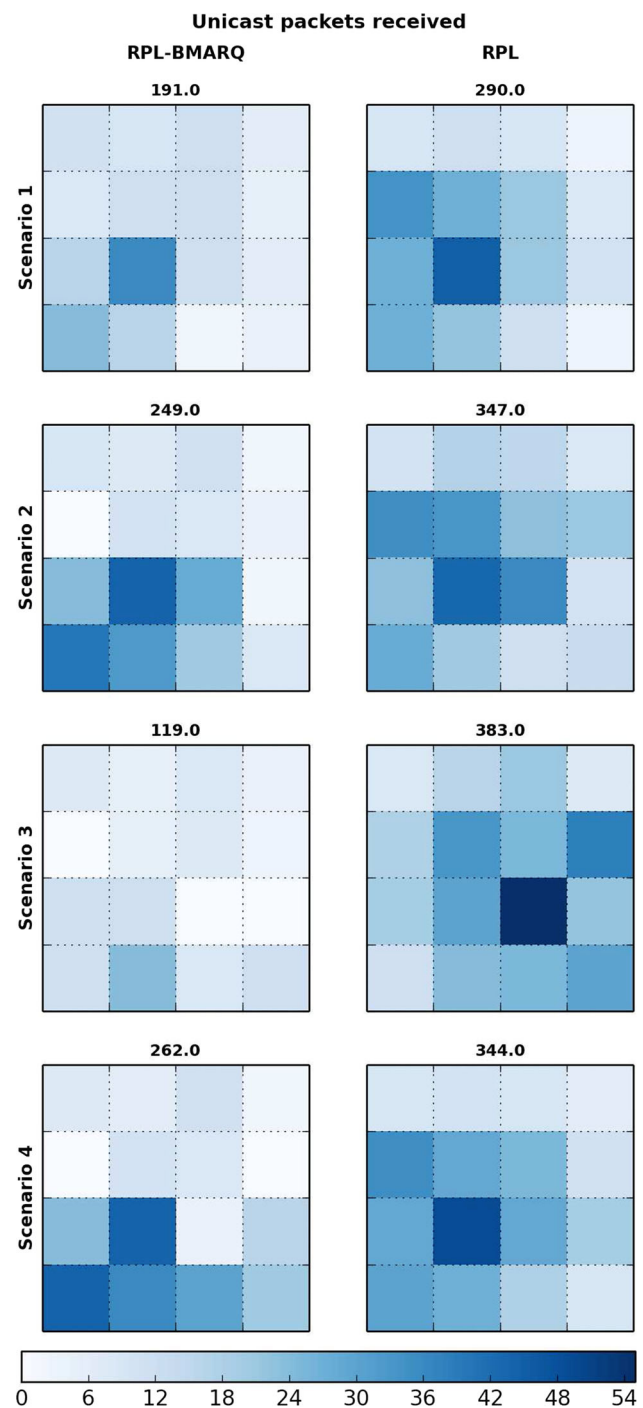


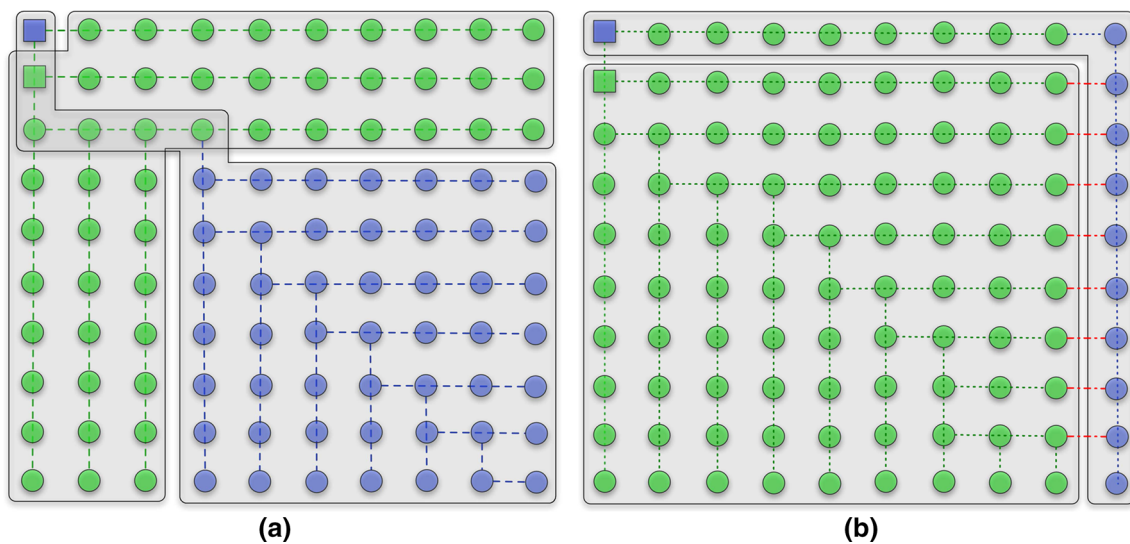
Fig. 19 Total number of unicast packets received in each scenario for the RPL-BMARQ and the RPL solutions

and the total number of *unicast* packets received by *RPL-BMARQ* is lower (20, 953) than *RPL* (21, 109).

From these results we can conclude that for large WSNs the number of hops does not affect our solution as it still provides significant energy gains.

Table 3 Theoretical results

	Scenario 1				Scenario 2				Scenario 3				Scenario 4			
	BMARQ		RPL		BMARQ		RPL		BMARQ		RPL		BMARQ		RPL	
	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx
Broadcast (packets)	40	94	80	240	40	90	80	240	41	84	80	240	44	88	80	240
Unicast (packets)	84	191	90	88	100	249	106	347	51	119	120	383	124	262	106	344
Time waked (s)	600		960		600		960		600		960		645		960	
Time sleeping (s)	57,000		56,640		57,000		56,640		57,000		56,640		56,955		56,640	
Time idle (s)	597.43		955.88		596.93		955.35		598.21		955.01		641.66		955.37	
Energy consumed (J)	5.86		8.73		5.88		8.76		5.82		8.78		6.24		8.76	
Energy gain (%)	33.0				33.0				33.8				28.9			

**Fig. 20** Impact on large networks: **a** 50 nodes running application A and 50 nodes running application B; **b** 10 nodes running application A and 90 nodes running application B**Table 4** Results obtained for large networks

	Topology (a)				Topology (b)			
	BMARQ		RPL		BMARQ		RPL	
	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx
Broadcast (packets)	255	848	255	930	379	1324	500	1800
Unicast (packets)	1707	5339	1707	5998	5607	20,953	5607	21,109
Time waked (s)	14,700		23,520		22,200		324,000	
Time sleeping (s)	338,100		336,000		203,400		336,000	
Energy consumed (J)	28.24		40.05		42.42		441.66	
Energy Gain (%)	29.5				90.4			

4.4 Simulations

In order to study the behavior of the RPL and the *RPL-BMARQ* solutions, both have been implemented in

ContikiOS [11], which is an operating system used for wireless sensor networks. *ContikiOS* 2.6 was chosen because it includes an IPv6 stack with *6LoWPAN* support, as well as *ContikiRPL*, which is a basic RPL

implementation. The communication between the nodes is achieved using the *CSMA/CA* access scheme with *NullRDC*, which means that the *Mac Layer*, by it-self, does not put the nodes in the *on* and *off* states. The simulations were performed using *Cooja* [34], a simulator for *ContikiOS*, which allows developers to test code and systems before running it on a target hardware. The hardware platform selected was *TelosB* [9] which uses IEEE 802.15.4 radios. The 2.4 GHz radio model chosen was the *Unit Disk Graph Medium* (UDGM). UDGM models the transmission range between the nodes as an ideal disk; the nodes outside it do not receive packets, while the nodes within the transmission distance receive all the messages. UDGM confirms the functionality and behavior of our solution. Packets have a IPv6 Payload of 82 bytes, except for signaling. Table 5 shows the code-size in bytes for sink and sensor nodes for *RPL* and for *RPL-BMARQ* solutions. As it can be verified, code-size overhead resulting from *RPL-BMARQ* implementation is about 7.1 % for sinks, and 6.4 % for sensor nodes. In contrast, code in RAM size is reduced by about 9.6 % for sinks and 10 % for sensor nodes.

4.4.1 Results and discussion

We have simulated *RPL-BMARQ* solution in two situations. The first corresponds to a situation where all the nodes join the network at same time, so that the designed nodes synchronization mechanism is not used, being represented in figures as *BMARQ (no sync)*. In the second situation, the nodes will join the network at different time, and it is represented in the figures as *BMARQ (sync)*. The later implies the use of the synchronization mechanism in order to keep the nodes synchronized with respect to the applications they run. The nodes join the network at different times which was randomly generated, and the expected random time value for a node to join a network is defined as

$$E[t_c] = \frac{1}{2} \cdot \text{Max}(T_{\text{Cycle}_A}, T_{\text{Cycle}_B}), \quad (7)$$

$$t_c \in [0, \text{Max}(T_{\text{Cycle}_A}, T_{\text{Cycle}_B})].$$

Table 5 Code-size for RPL and RPL-BMARQ solutions

	RPL			RPL-BMARQ		
	ROM	RAM		ROM	RAM	
Sink	42,280	186	7400	45,492	186	6688
Sensor	42,260	186	7400	45,170	186	6662

Shown is ROM (.text) and RAM (.bst + .data) in bytes

For the evaluated scenarios, T_{Cycle_A} equals 1 h and T_{Cycle_B} 15 min. So, the average delay for a node to join the network is 1800 s. Table 6 shows the time required to boot the network in these conditions.

DAGs created: Figure 21 shows the DAGs generated during simulations. For each scenario the simulations ran constructed the same DAGs. In this figure it can be verified that *RPL-BMARQ* solution constructs the DAGs as expected. A particular attention must be given to scenario 4 where node 16 choses node 8 as parent which does not run its application. In this situation, node 8 should not send packets from nodes 10, 13 and 16 to its parent (node 7), but use other link, sending them directly to node 15 which will forward them so that the node 9 (sink) can receive the packets. This aspect raised some issues and therefore it is discussed separately.

Energy: we consider energy consumption related only to communication aspects, e.g. packet transmission, reception, *radio “idle”* the state where a node has its *radio on* and is waiting to send or to receive a data “packet”, and *radio interferences*. We aim to investigate how much energy is consumed by the nodes in these states, as shown by Eq. 8.

$$E = E_{TX} + E_{RX} + E_{Idle} + E_{Int} \quad (8)$$

E_{TX} is the total energy consumed when sending packets, E_{RX} is the total energy consumed when receiving packets,

Table 6 Nodes association time randomly generated for each scenario

Scen.	Node type	App.	Node	Boot (in s)
1	Sensor	A	2, 3, 4	561
			5, 6, 7, 8	1027
		B	10, 12, 15	940
			11, 13, 14, 16	1102
2		A	2, 3, 5, 6	561
			4, 7, 8	1027
		B	10, 11, 13, 14	940
			12, 15, 16	1102
3		A	2, 3, 4, 5	561
			6, 7, 8	1027
		B	10, 11, 12, 13	940
			14, 15, 16	1102
4		A	2, 3, 5	561
			4, 6, 7, 8	1027
		B	11, 12, 14, 15	940
			10, 13, 16	1102
All	Sink	A	1	399
		B	9	317

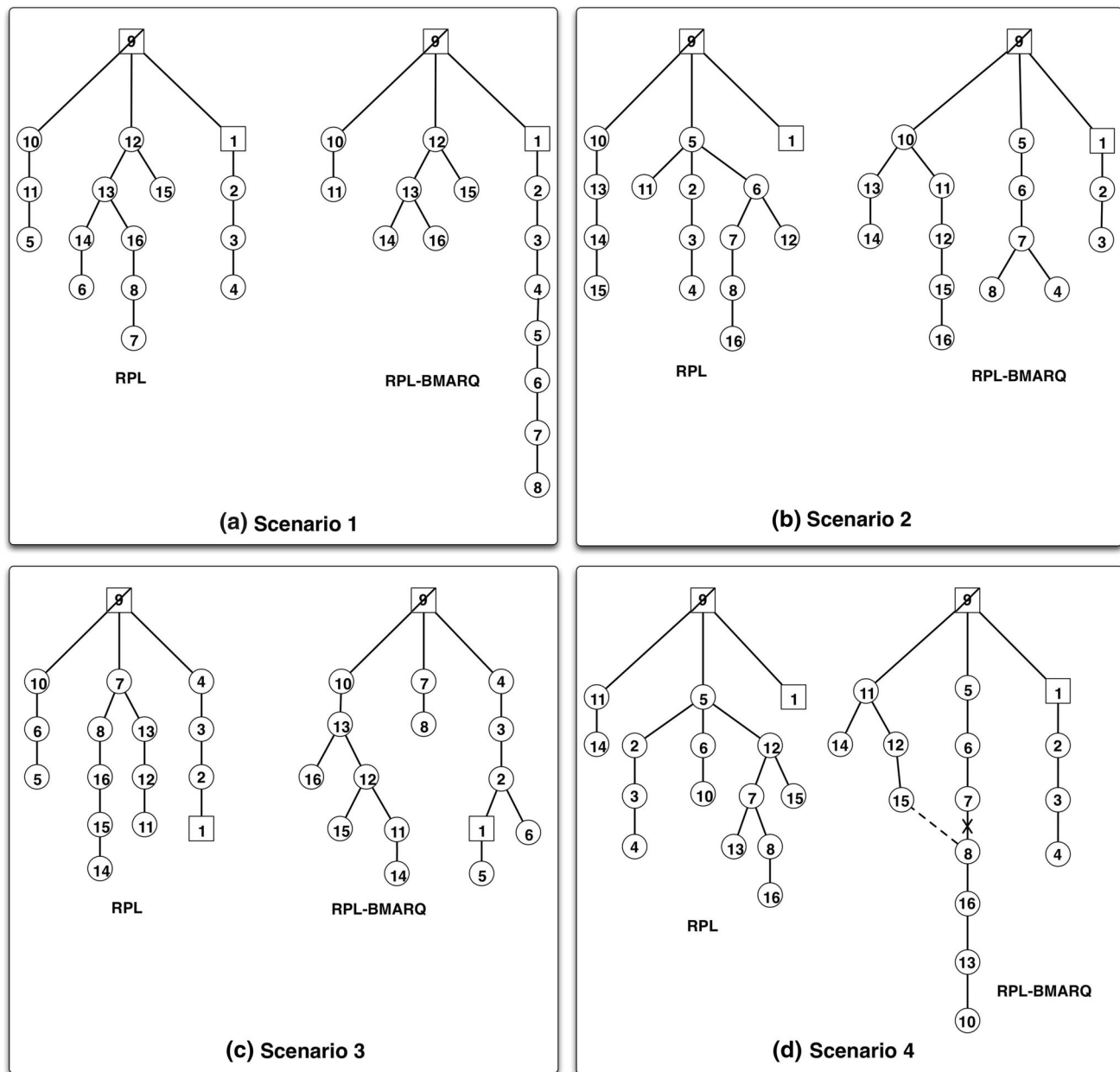


Fig. 21 DAGs generated during simulations

E_{Idle} is the total energy consumed by a node when it has the radio in the *idle state*, and E_{Int} is the total energy consumed by a node when it suffers radio interferences from its neighbors. Generically, we compute energy E_{state} as $I_{state} \times V \times t_{state}$, considering I_{state} the current consumed by the node in the *state*, V the voltage supplied to the node, and t_{state} the total time the node is in that *state*. The I_{state} and V values depend on existing platforms (e.g. TelosB [9]), and a generic energy model defining the total energy consumed by the nodes is detailed in [27]. From the results obtained, we extracted data related to communications time which is shown in Fig. 22. This figure shows the total time,

for 95 % confidence intervals, in which nodes radios were in their different considered states. As one can observe, the permanence time in each state is lower for *RPL-BMARQ* (both *no sync* and *sync* implementations) than for *RPL*. Using *RPL-BMARQ*, the time where radios are in the idle state is less than that using *RPL*, which reflects the major contribution of *RPL-BMARQ* design. Applying the information of Table 2, which was extracted from [9], to Eq. 8, we can compute the total energy consumed by the nodes (see Fig. 23). Results show that the total energy consumed by the nodes considering those states, and using the *RPL-BMARQ* solution using both implementations is very low,

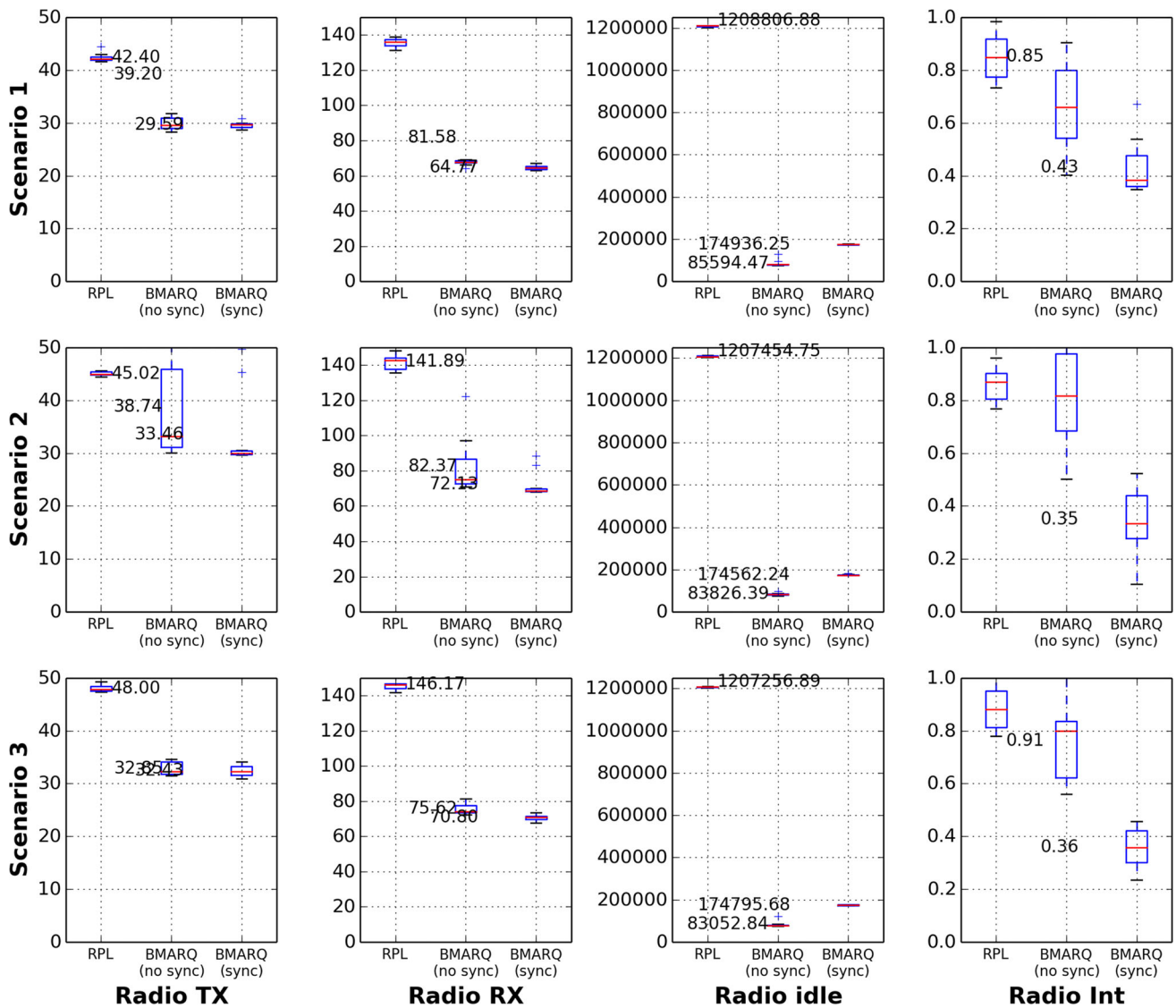
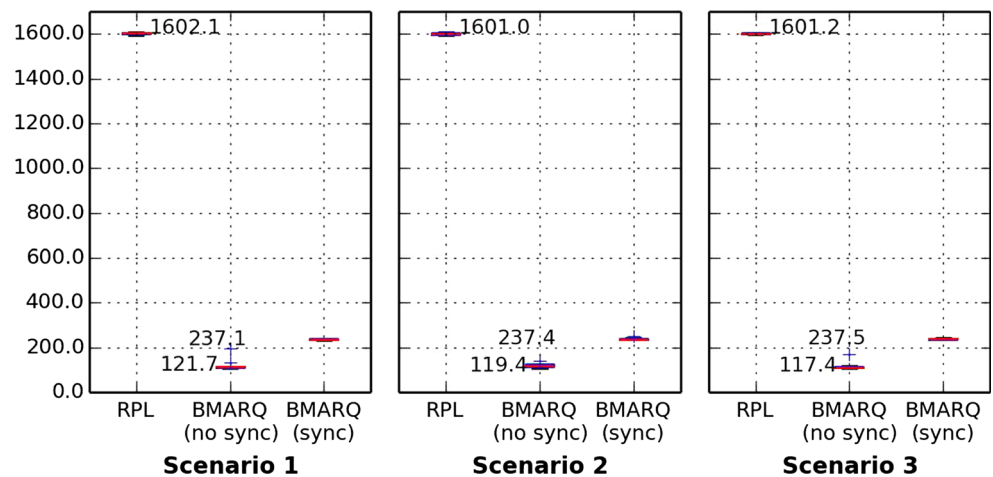


Fig. 22 Mean total radio activity time (in seconds)

Fig. 23 Energy consumed by each solution in each scenario (in J)



as it turns the node's radio *off* during more time than *RPL*. Using Eq. 6 we compute the energy gain for *RPL-BMARQ* and results show that, using the *RPL-BMARQ* solution not implementing the synchronization mechanism, the nodes spend about 92 % less energy than the same nodes running *RPL*. In the case of the second implementation of *RPL-BMARQ* in which the synchronization mechanism is used, the nodes spend about 85 % less of energy than *RPL*. The difference in gain between the two implementations of *RPL-BMARQ* is due to the excess time the nodes in the synchronized implementation need to be awaked, thus spending more energy. Even though, this enables us to conclude that *RPL-BMARQ* extends the network lifetime.

Synchronization: each time a node receives a query it computes the time it must be wake before the next application cycle in order to be able to receive and forward packets, and to reply back to the sink successfully. For that purpose, it uses the synchronization mechanism described in 3.3, where β value from Eq. 3 was empirically obtained, and set to 10. Figure 24 shows the mean value of the synchronization adjusting time used by the nodes in the scenarios evaluated. As it can be seen, the nodes would sleep not the correspondent T_{off} time, but in average $T_{off} - 15.7$ s, corresponding to the second implementation of *RPL-BMARQ*.

Query Success Ratio: we define *Query Success Ratio* (QSR) 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 (see Eq. 9).

$$QSR = \frac{\text{number_of_received_replies}}{\text{number_of_expected_replies}}, \quad (9)$$

$$0 \leq QSR \leq 100 \%$$

Adapting Eq. 9 to our case, the number of received replies is 7. Figure 25 shows simulation results with 95 % confidence interval. One can conclude that *RPL* presents better result than *RPL-BMARQ*, although the values from both

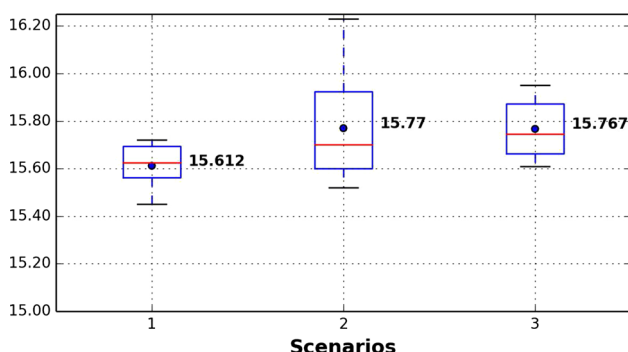


Fig. 24 Mean synchronization adjusting time ($\beta \cdot |\delta_k|$, in s) for each scenario

solutions are very close. Please note that in Fig. 25 we are representing *QSR* values between 96 and 100 %. We can conclude that with *RPL-BMARQ*, *QSR* does not suffer much, presenting a average value of 98.5 %, when compared to *RPL*, which presents a similar value of 99.5 %. At the end, the difference value of 1 % has no great significance.

QSR fairness: fairness metrics are used in network engineering to determine whether users or applications are receiving a fair share of system resources. There are several definitions of fairness. *Jain's fairness index* is an example, and it is defined in Eq. 10 [20], and it describes the fairness of a set of values where there are n users and x_i is the throughput for the i^{th} user. A straightforward computation shows that the fairness measure γ ranges from $\frac{1}{n}$ (maximum unfairness) to 1 (all x_i are equal) [6]. In the evaluation of our proposed solution also we want to know if the sensor nodes have the same opportunity to receive and to reply to query packets. We also use the *Jain's fairness index* as it is independent of scale, it applies to any number of sensor nodes, and it is bounded between 0 and 1, where $\gamma = 1$ indicates a totally fair network.

$$\gamma(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \cdot \sum_{i=1}^n x_i^2} \quad (10)$$

Applying Eq. 10 to our case, x_i corresponds to the *QSR* per node. In average, values obtained from simulations for the 3 scenarios show that all solutions present fairness indexes above 99 %.

Delay: we define *delay* (Δ) as the time interval between the time instant a query was sent by a sink node and the time the sink receives the correspondent reply, as shown in Fig. 26. We compute this delay as

$$\Delta_{k,n} = t_{R_{k,n}} - t_{Q_k} \quad (11)$$

where $\Delta_{k,n}$ is the delay for the reply from the node n to a query k , t_{Q_k} is the time the query k was sent, and $t_{R_{k,n}}$ is the time the reply k from the node n was received. Figure 27 shows delay values with 95 % confidence for the simulated scenarios using both solutions and implementations. These values were computing as defined in Fig. 26. We note that with *RPL-BMARQ* we can achieve higher mean delays. For instance, in *Scenario 3*, the mean delay value is 1.524 s for *RPL-BMARQ* with the synchronization mechanism implemented, 1.539 s for *RPL-BMARQ* not using the synchronization mechanism, and 1.375 s for *RPL*. This is expected since nodes using *RPL-BMARQ* have bigger processing times. Also, analyzing the DAGs generated (Fig. 21) we note that, in average, *RPL-BMARQ* creates longer DAGs so the packets would take more time to reach their destinations. In average, the delay value for *RPL* is 1.24 s, the delay value for *RPL-BMARQ* (*no sync*) is 1.39 s, and the

Fig. 25 Mean Query Success Ratio—QSR (in %)

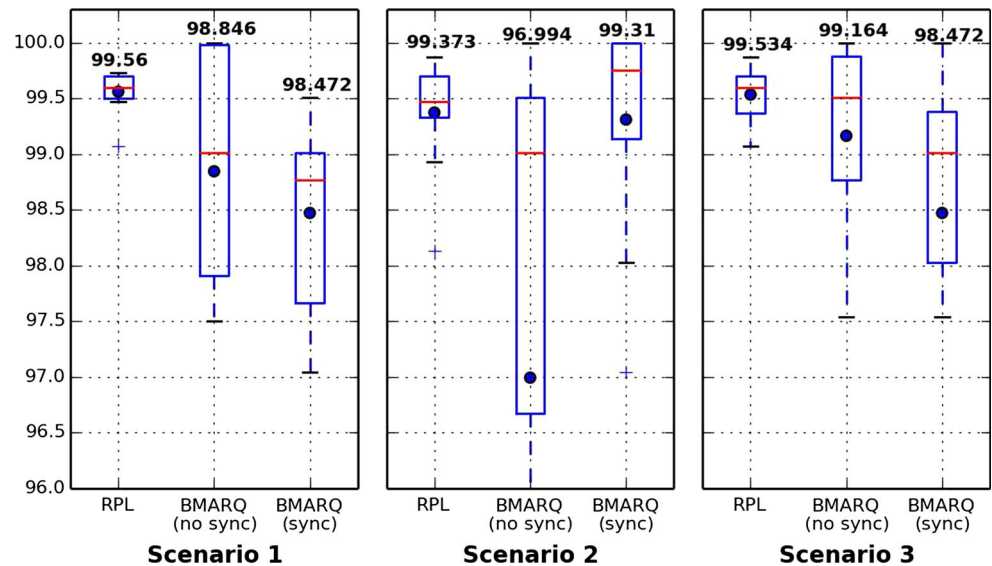
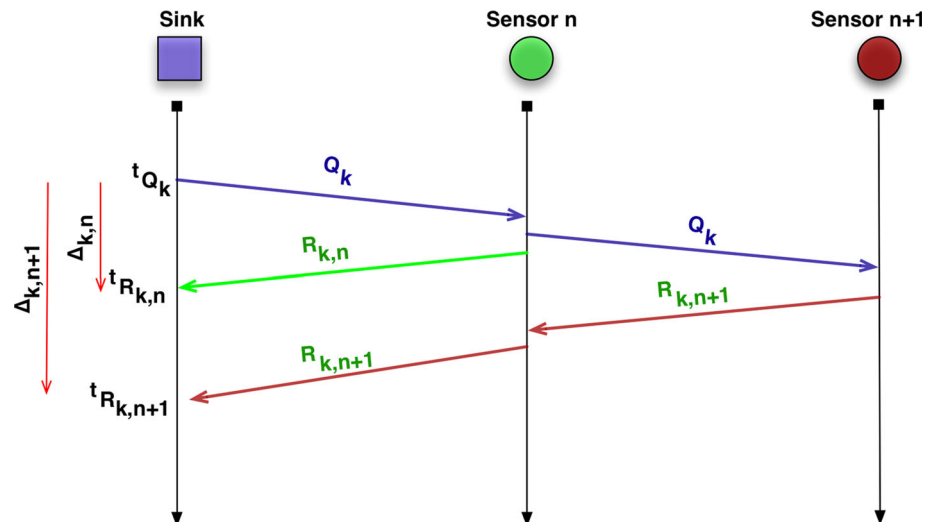


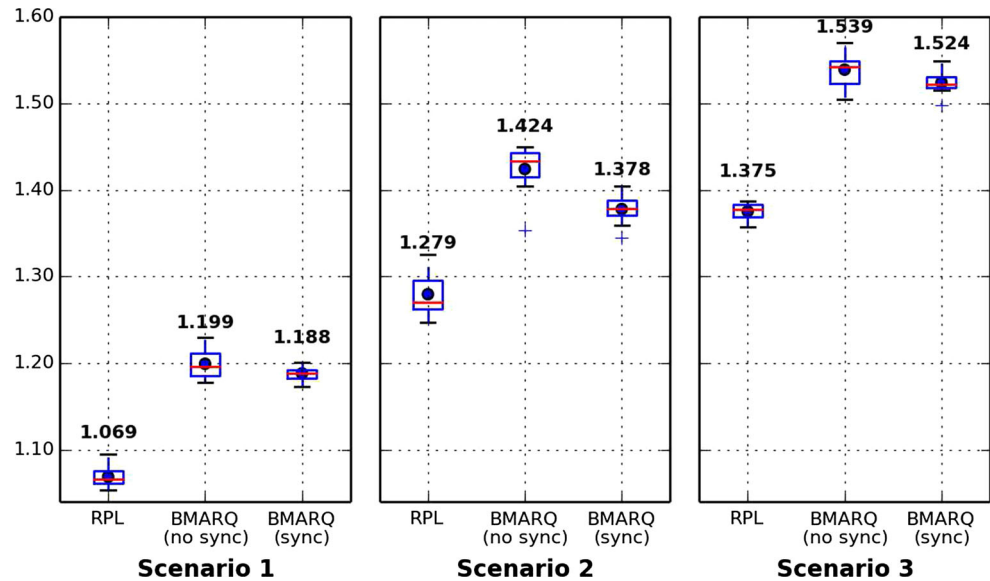
Fig. 26 Delay definition



delay value for *RPL-BMARQ (sync)* is 1.36 s. This corresponds to more 10.8 and 8.8 % of delay time, respectively for *RPL-BMARQ (no sync)* and *RPL-BMARQ (sync)* implementations. Finally, analyzing delays for both *RPL-BMARQ* implementations, when nodes use the synchronization mechanism, they would not be active all at same time, but as necessary. This has the effect of reducing communication activities, occupying the transmission media for less time.

One problem that our solution may raise is related to load balancing between nodes. There are applications which require the transmission of more packets per time unit than others and, as a consequence, nodes belonging to the subnetwork defined by the high packet rate applications are required to process more packets. This may create load balancing related issues such as higher network delays

through some of the network paths. These delays may be higher in our solution than in regular *RPL*. However, if the offered loads are low and stable, as it is the case of the traffic scenarios envisaged, these differences should not be relevant for our claims that are related to energy savings, as may be observed for instance in Fig. 27 and in Fig. 33, where one application demands the transmission of more packets than the other; therefore, the nodes in the subnetwork defined by the high packet rate application process more packets and delays become higher. *RPL* does not subdivide the network and routing paths do not depend on applications and, for this reason, *RPL* presents lower delays. Figures 27 and 33 enable us to estimate the magnitude of increasing delays (in %) introduced by our solution. Analyzing Fig. 27 and comparing to regular *RPL* we can verify that: for scenario 1, and not using the

Fig. 27 Mean delay (in s)

synchronization mechanism, *RPL-BMARQ* presents 10.8 % more delay, whereas using the synchronization mechanism the same is 10 %; for scenario 2, and not using the synchronization mechanism, *RPL-BMARQ* presents 10.2 % more delay, whereas using the synchronization mechanism the increase is 7.2 %; and for scenario 3, and not using the synchronization mechanism, *RPL-BMARQ* presents 10.7 % more delay, whereas using the synchronization mechanism the increase is 9.8 %. For the case of scenario 4, from Fig. 33 we can observe that, when not using the synchronization mechanism, *RPL-BMARQ* presents an increase of delay of 17.7 %, whereas using the synchronization mechanism the same increase is 16.8 %. In this scenario delays are higher because there is a node running one application which is required also to process packets from the other application. From the above analysis we may conclude that our solution makes delays to increase about 10 %, in average, but our claims on energy still hold.

Packets per query: in the simulations performed, the sink node running *App. A* generates a total of 24 queries, whilst sink node running *App. B* generates 96 queries. Analyzing the data extracted from simulations we could investigate on a *per-query* basis, how many *Layer 3* multicast and unicast packets were sent and received by all the nodes. We consider also routing packets. Table 7 summarizes simulation results showing the total number of multicast and unicast packets transmitted and received by the nodes in each network solution. For all the scenarios, both *RPL-BMARQ* solutions implementation present lower mean number of total *Layer 3* multicast packets, sent and received. For example in *Scenario 1*, using *RPL-BMARQ (no sync)*, and *per-query*, the mean number of total packets sent is 10 and the mean number of total packets received is

Table 7 Mean total number of packets per query (with 95 % confidence interval)

Scenario	Solution	Multicast		Unicast	
		TX	RX	TX	RX
1	<i>RPL</i>	15	45	45	135
	<i>BMARQ (no sync)</i>	10	25	33	77
	<i>BMARQ (sync)</i>	8	17	29	49
2	<i>RPL</i>	15	45	50	141
	<i>BMARQ (no sync)</i>	9	24	36	76
	<i>BMARQ (sync)</i>	7	17	32	69
3	<i>RPL</i>	15	45	53	144
	<i>BMARQ (no sync)</i>	10	26	37	59
	<i>BMARQ (sync)</i>	7	6	33	53

25; using *RPL-BMARQ (sync)*, the mean number of total packets sent is 8 and the mean number of total packets received is 17; using *RPL*, the mean number is higher (15 packets sent and 45 packets received). From the results we conclude that the major gain of the *RPL-BMARQ* solution relies on the total number of both multicast and unicast packets sent and received, which is lower than the equivalent *RPL*.

Reaction to topology changes: *RPL-BMARQ* solution behaves just like *RPL* where the time to converge, and the reaction to network topology changes are similar to those observed for *RPL*. *RPL-BMARQ* uses the same *Trickle timer mechanism*. When a node does not agree with its neighbors, that node communicates quickly to resolve the inconsistency. On the other and, when nodes agree they slow their communication rate exponentially, and end by exchanging packets very infrequently. Instead of flooding a

network with packets, the algorithm controls the sending rate so that each node hears a small trickle of packets, just enough to stay consistent [25].

4.5 Special case: scenario 4

We have simulated this scenario as the others, and while running and analyzing preliminary results, this scenario presented an issue: node 9 (sink from application B) does not receive replies from nodes 10, 13 and 16, although queries are received and replied by them. This behavior have the root cause in node 8 which, although running application A, is also parent of node 16 and therefore responsible to forward packets from nodes 16, 13 and 10. Since node 8 selects node 7 as parent (see Fig. 21d), reply packets are never forwarded by node 7 because it does not run the application of nodes 10, 13 and 16. To solve this issue, node 8 upon the reception of nodes 10, 13 and 16 reply packets, should forward them directly not to node 7, but to node 15, which runs their application, to allow for node 9 the reception of all expected reply packets. In this kind of scenarios, every node being parent of nodes not running the same application, and having selected as parent a node running the same application, should send directly all received reply packets to a neighbor node running the same application from which reply packets where originated. A close snapshot of this is showed in Fig. 28 where node 8 has selected node 7 (fe80::212:7407:7:707) as parent, but detecting that a reply packet is, as example, from node 10 and node 16 (aaaa::212:740a:a:a0a and aaaa::212: 7410:10:1010) from the application it doesn't run, changes the link to node 15 (fe80::212:740f:f:f0f).

Energy consumption: we also consider energy consumption related only to communications aspects and from the results obtained we extracted data from communication time which is shown in Fig. 29. This figure shows the total time, for 95 % confidence intervals, in which nodes radios were in their different considered states. As one can observe, the permanence time in each state is lower for *RPL-BMARQ* (both *no sync* and *sync* implementations) than for *RPL*. We have computed the total energy

consumed by the nodes using Eq. 8, and presented results in Fig. 30. They show that the total energy consumed by the nodes running *RPL-BMARQ* is very low, as it turns the nodes radio *off* during more time than *RPL*. We have also computed the energy gain for *RPL-BMARQ* as described in Sect. 4.3.2. Results show that, when the synchronization mechanism is not used the nodes spend about 93 % less energy than the same nodes running standard *RPL*. On the other hand, if the synchronization mechanism is used, the nodes still spend less energy (about 85 %) (Fig. 31).

Synchronization: Figure 24 shows the mean value of the synchronization adjusting time used by the nodes in this scenario. As it can be seen, the nodes would sleep less about 15.7 seconds per application cycle, making them staying wake more time.

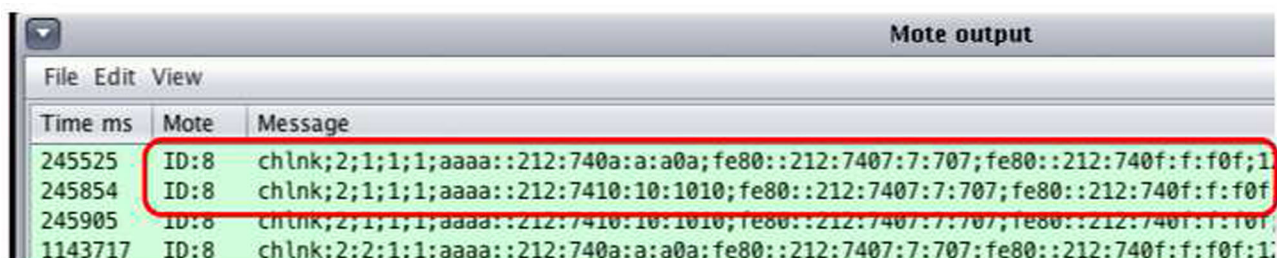
Query Success Ratio: Figure 32 shows simulation results with 95 % confidence interval. Although the values from both solutions are very close, *RPL* presents better result than *RPL-BMARQ*. In this figure we are representing *QSR* values between 96 and 100 %, and considering that the difference value is about 1 %, so we can conclude that with *RPL-BMARQ* *QSR* does not suffer much.

QSR fairness: For scenario 4, mean *QSR fairness* values obtained from simulations present similar fairness indexes (about 99.9 %) for both solutions.

Delay: Figure 33 shows 95 % confidence intervals for delay values using both solutions. We note that with *RPL-BMARQ* we achieve higher delays. The mean delay value for *RPL-BMARQ* is 1.55 s, and 1.28 s for *RPL*. This is also expected since in this scenario *RPL-BMARQ* constructs a longer DAG (see Fig. 21), so nodes use more hops to communicate. Also, in each node is used more processing time.

Packets per query: in the simulations performed, the sink node running *App. A* generates a total of 24 queries, whilst sink node running *App. B* generates 96 queries. Analyzing the data extracted from simulations we could investigate on a *per-query* basis, how many *Layer 3* multicast and unicast packets were sent and received by all the nodes. We consider also routing packets.

Figure 34 shows for 95 % confidence intervals, an *per-query* basis, the number of packets transmitted and



Time ms	Mote	Message
245525	ID:8	chlnk;2;1;1;aaaa::212:740a:a:a0a;fe80::212:7407:7:707;fe80::212:740f:f:f0f;1
245854	ID:8	chlnk;2;1;1;aaaa::212:7410:10:1010;fe80::212:7407:7:707;fe80::212:740f:f:f0f
245905	ID:8	chlnk;2;1;1;aaaa::212:7410:10:1010;fe80::212:7407:7:707;fe80::212:740f:f:f0f;1
1143717	ID:8	chlnk;2;2;1;1;aaaa::212:740a:a:a0a;fe80::212:7407:7:707;fe80::212:740f:f:f0f;1

Fig. 28 Changing link-local address to correctly forward reply packets

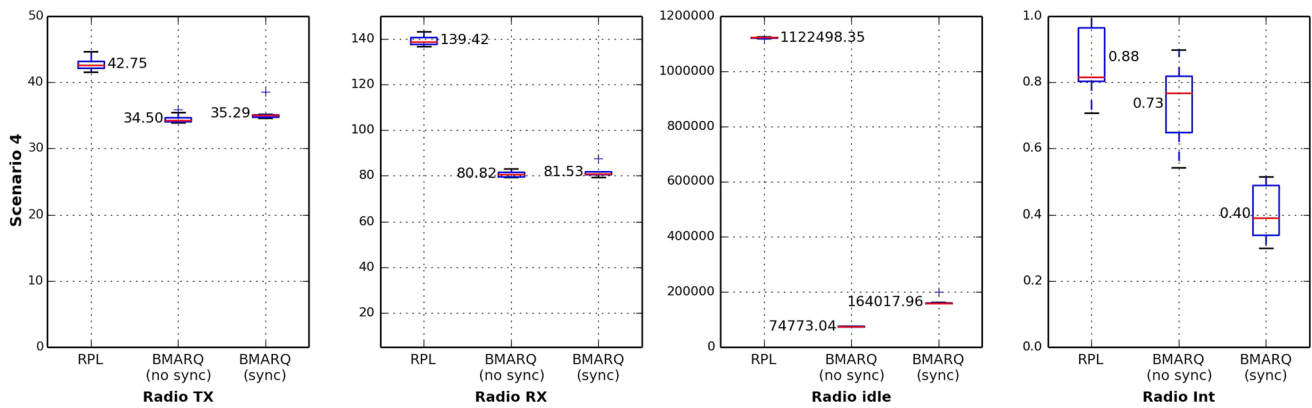


Fig. 29 Mean total radio activity time in scenario 4 (in s)

Fig. 30 Energy consumed by each solution in scenario 4 (in J)

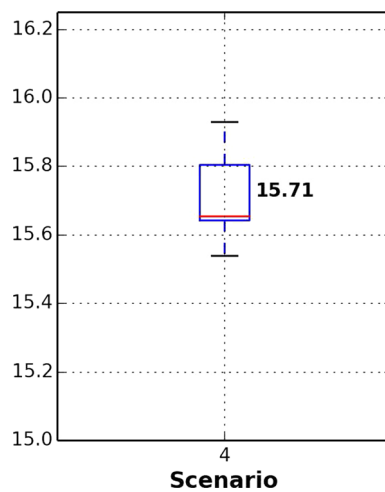
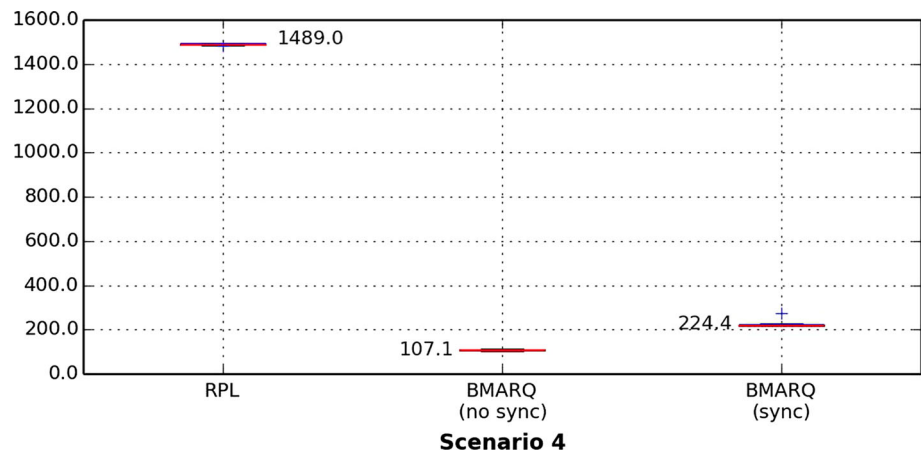


Fig. 31 Mean synchronization adjusting time ($\beta \cdot |\delta_k|$, in s) for scenario 4

received. Both *RPL-BMARQ* solutions present lower mean number of total *Layer 3* multicast packets, sent and received. Using *RPL-BMARQ (no sync)*, the mean number

of total packets sent is 11 and the mean number of total packets received is 29; using *RPL-BMARQ (sync)*, the mean number of total packets sent is 9 and the mean number of total packets received is 22; using *RPL*, the mean number is higher (15 packets sent and 45 packets received). For the mean number of total *Layer 3* unicast packets sent and received, the results show that also both *RPL-BMARQ* implementations present lower numbers. The mean number of total unicast packets sent using *RPL-BMARQ* (in both implementations) is 44, whilst using *RPL* this value is 50; the mean number of total unicast packets received using both implementations of *RPL-BMARQ* is 82 and using *RPL* the same is 135.

From the above analysis we conclude that the *RPL-BMARQ* presents the same behavior as in the other scenarios, and that the major gain introduced by *RPL-BMARQ* solution is the capability to wake and to sleep the sensor nodes in a synchronized manner, reducing radio activity time, while maintaining *QSR* and *fairness* ratios high.

Fig. 32 Mean Query Success Ratio (QSR) for scenario 4 (in %)

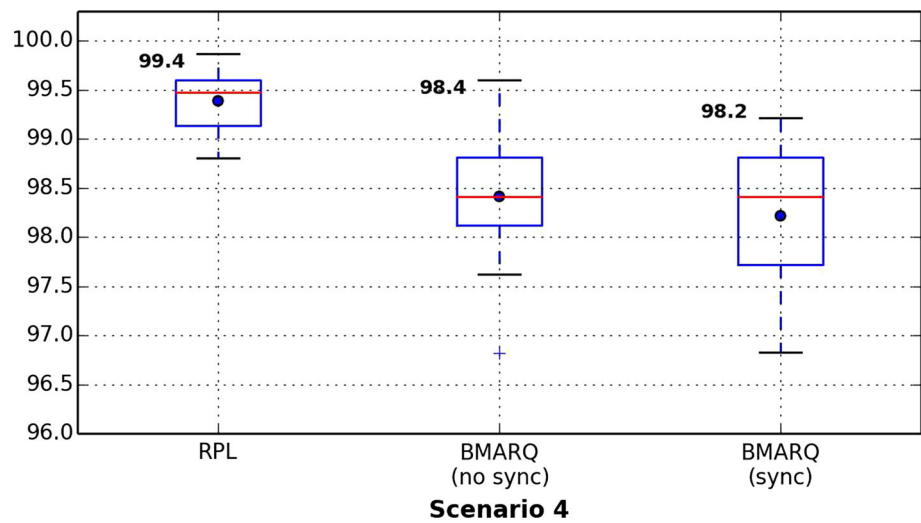


Fig. 33 Mean delay for scenario 4 (in s)

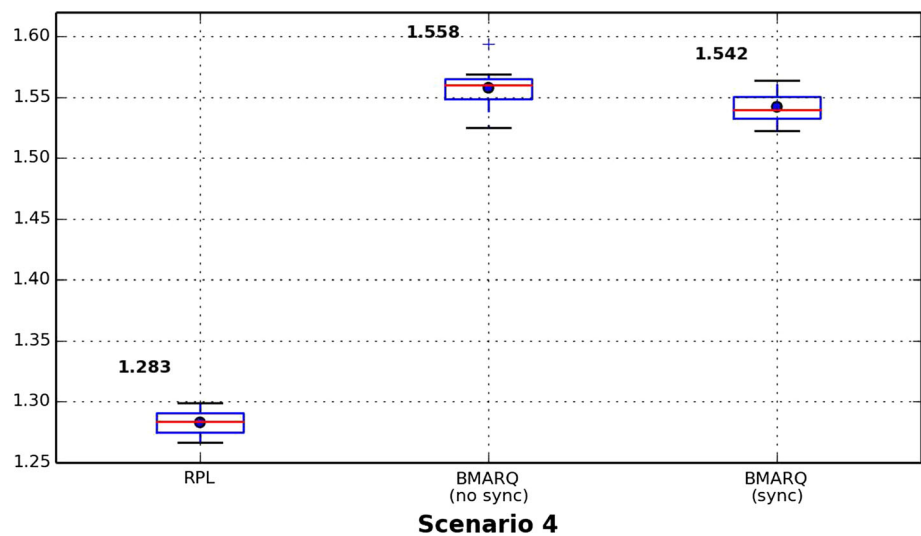
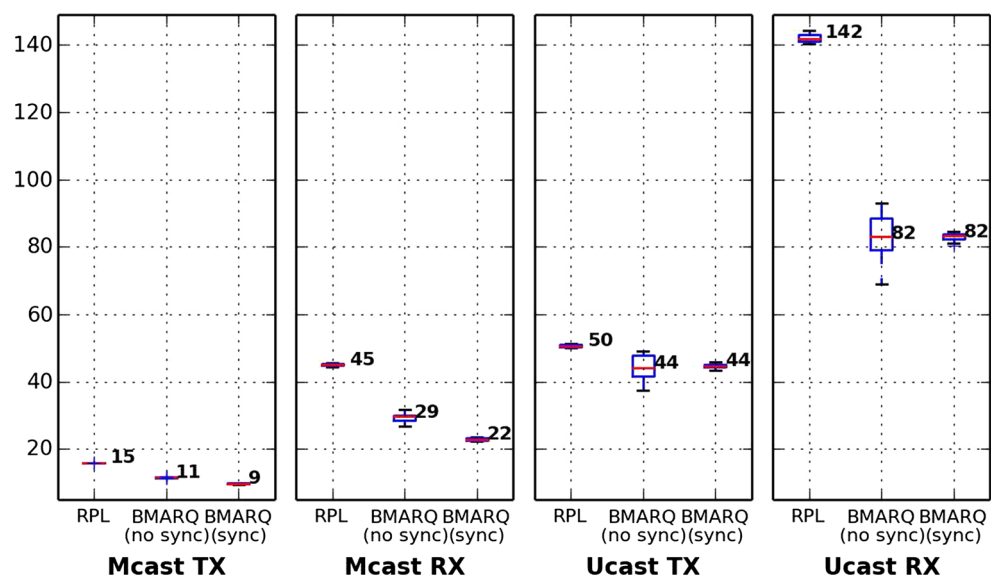


Fig. 34 Mean total number of packets per query for scenario 4



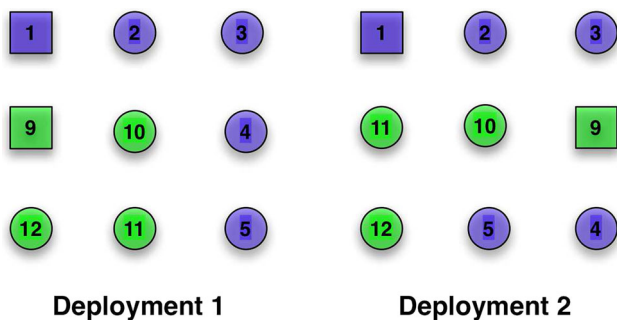


Fig. 35 Scenarios deployed

5 Testbed experiments

5.1 Scenarios deployed

In order to confirm the results obtained from simulations, it was necessary to test *RPL-BMARQ* in a real environment. For that purpose, two of the scenarios studied were select (scenarios 1 and 3) and deployed. Since it was not possible to reproduce them at same scale, the scenarios deployed correspond to a 3x3 square lattice topology, while keeping all the assumptions of the evaluated scenarios. Figure 35 shows both deployments, which were realized in an *Auditorium* inside *Escola Superior de Tecnologia e Gestão de Viseu* facilities (see Fig. 36). The nodes were placed at a distance of 5 m, and the radio transmission power was reduced to -7 dBm in order to reduce nodes radio influence space. Application A runs in five nodes (1, 2, 3, 4 and 5), while application B in four nodes (9, 10, 11 and 12). Node 9, is at the same time the root of the DAGs and sink. Node 1 is the other sink.

5.2 Energy metering

It was necessary to measure real power consumption to compare with the energy consumption resulted from simulations. Again, a node (node 5) was randomly selected to measure real power consumption. For this, it was necessary to implement a small *Energy meter*, which could measure in real time the energy consumed by this node. Figure 37 shows this implementation which uses a *BeagleBone Black* platform [5] responsible to acquire analog values from the measured node. These values are acquired using an instrumentation amplifier which samples the voltage drop of a serial shunt resistor, supplied to the measured node. This platform runs a python program which acquires every second 25 samples, returning their mean value, and uses it to compute power consumption. For accuracy it was necessary to measure real values, compare and correct them from the nodes datasheet [9], and use them in the

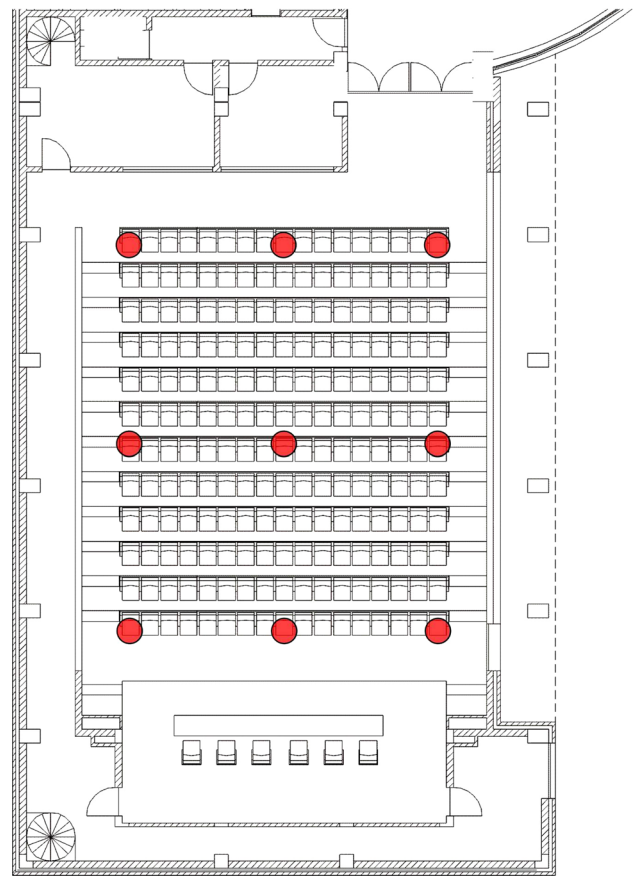


Fig. 36 Local of testbed deployment

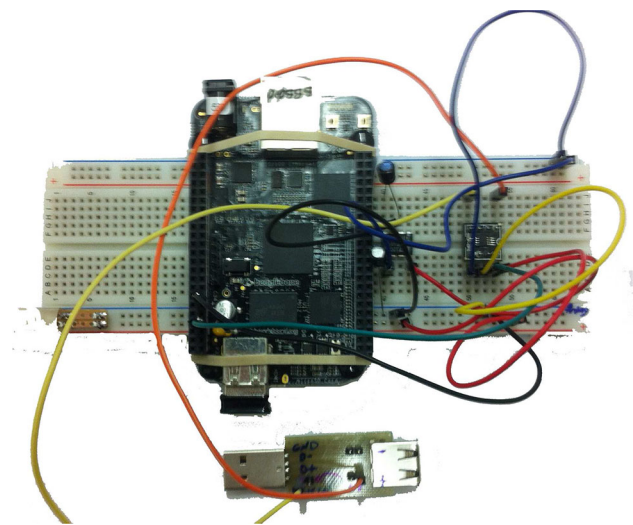


Fig. 37 Energy meter implemented

simulations to reflect real values. As such, for power supply the value was corrected to 3 V; and the power consumption when MCU is on and the Radio Off to 12 mA.

5.3 Results and discussion

The experiments were carried for 4 h, and the obtained results are presented and discussed as following. To log some real time data, two *Raspberry Pi* platforms [37] were used, connected to both sink nodes via a serial connection. Inside each *Raspberry Pi* platform was a python program running, responsible to get timestamp data from each sink with respect to queries sent, replies received; and signaling and routing packets sent and received, which were used to obtain the following results.

Energy consumption: simulation and real implementation results for the scenarios deployed demonstrated that, in the real testbed implementations, the node consumes a little more energy than in simulations. Additionally the energy recorded corresponds not only to other hardware components consumption that the nodes have (e.g. LEDs, ADCs) which were not considered in the simulation platform, but also to the total number of transmitted and received packets by the node which was not possible to record. In the first deployment, simulation showed that the node consumes 485.83 J, and in the second 534.26 J. Node real measurements have shown a consumption of 567.73 and 552.41 J. In the first deployment the node consumes more 16.8 % of energy whereas in the second the same node consumes more 3.4 % of energy, when compared to simulations results. From these results, and considering that some hardware platform components were not considered in the simulations, we can conclude that the real node energy consumption in each deployment can be considered as valid.

Query Success Ratio: Figure 38 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, maintaining their synchronization.

QSR fairness: in average, *QSR fairness* values obtained from simulations and testbeds implemented present same

values (100 %). This also means that in real testbeds, the nodes have the same opportunity to reply to all the queries sent by the sinks, as all the queries are equally received by all the nodes.

Delay: Figure 39 shows simulation and real implementation delay results. As one can observe, both present almost the same values (about 900 ms), which means that nodes in the testbeds have the same behavior as in the simulation environment. Note that in this figure we are representing *delay* between 600 and 950 ms.

From the above results we can conclude that there are no major differences between what was observed in the simulation environment and that what was expected in the testbed environment, and consider valid the testbeds, confirming the usability of *RPL-BMARQ*.

6 Conclusions and future work

In this paper we proposed a solution named (*RPL-BMARQ*) to deploy WSN defined by the applications the nodes run. The paper presents and discusses the performance of *RPL* and *RPL-BMARQ* by simulating sensor devices using *IEEE 802.15.4* radios. The work presented reflects a theoretical analysis of a solution which assumes that the nodes may run different applications inside the same WSN with multi-hop connectivity. The solution tackles the problem by restricting routing and forwarding functions mainly to the nodes running the same applications, letting other nodes sleeping and waking in a synchronized manner, and avoiding their utilization. *RPL-BMARQ* was evaluated against *RPL* by means of simulations using *ContikiOS* and *Cooja*, and confirmed that the proposed solution reduces overall network energy consumption, thus increasing the network lifetime. Our results show that when designing WSN applications with the *RPL-BMARQ* solution, the nodes will not retransmit data packets from applications which they do not run; nodes not involved in

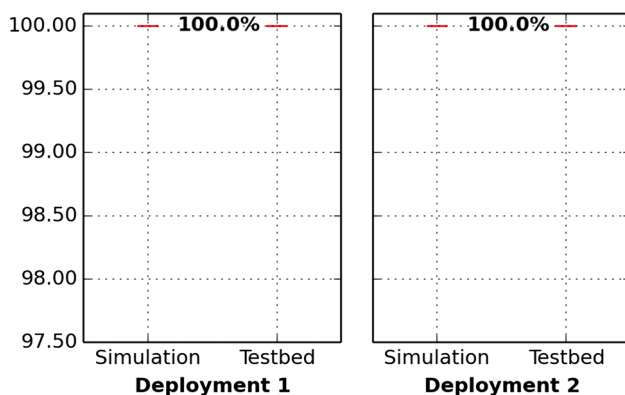


Fig. 38 Query Success Ratio (QSR) for the scenarios selected (in %)

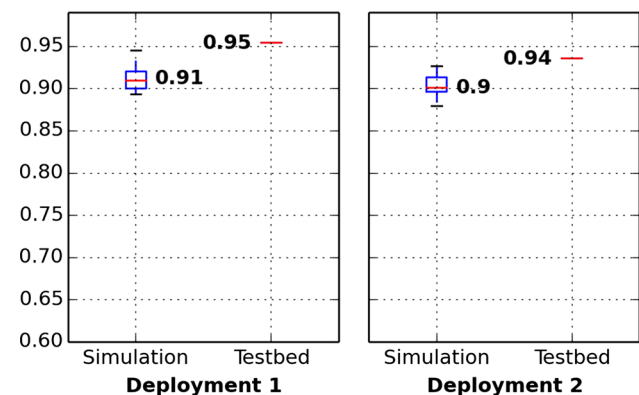


Fig. 39 Delay for the scenarios selected (in %)

communications are kept sleeping as much as possible. Finally, real testbed experiments confirmed some simulations results, proving that *RPL-BMARQ* can be used in real applications to extend a WSN lifetime. Further experiments with *RPL-BMARQ* may need to be performed using scenarios where hundreds of nodes are randomly deployed.

Acknowledgments 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”. Authors would like to thank also the support from 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.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Human and animal rights The work does not contain any studies with human participants or animals performed by any of the authors.

References

- Accettura, N., & Piro, G. (2014). Optimal and secure protocols in the IETF 6TiSCH communication stack. In *Proceedings of IEEE international symposium on industrial electronics (ISIE)*.
- Akyildiz, I. F., Vuran, M. C., Akan, O. B., & Su, W. (2006). Wireless sensor networks: A survey revisited. *Computer Network Journal* (Elsevier Science).
- Anastasi, G., Conti, M., Francesco, M. D., & Passarella, A. (2009). Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3), 537–568. doi:10.1016/j.adhoc.2008.06.003.
- Baccelli, E., Philipp, M., & Goyal, M. (2011). The P2P-RPL routing protocol for IPv6 sensor networks: Testbed experiments. In: *2011 19th international conference on software, telecommunications and computer networks (SoftCOM)* (pp. 1–6) (2011).
- BeagleBone Black. (2014). <http://beagleboard.org/BLACK>.
- Boudec, J. (2010). Performance evaluation of computer and communication systems. Computer and communication sciences. EFPL Press. <http://books.google.pt/books?id=nibpCdEjUEYC>.
- Catarinucci, L., Colella, R., Del Fiore, G., Mainetti, L., Mighali, V., Patrono, L., et al. (2014). A cross-layer approach to minimize the energy consumption in wireless sensor networks. *International Journal of Distributed Sensor Networks*, 2014, 11. doi:10.1155/2014/268284.
- Chipcon Products and Texas Instruments. (2006). 2.4 GHz IEEE 802.15.4/ZigBee-ready RF transceiver. Document SWRS041.
- Crossbow TelosB. (2004). http://www.memsic.com/userfiles/files/Datasheets/WSN/6020-0094-02_B_TELOSB.pdf.
- Culler, D. E. (2008). Wireless mesh networks promise low power ip-based connectivity. *Industrial Ethernet Book*, 49, 16–22.
- Dunkels, A. (2013). Contiki OS, open source, highly portable, multi-tasking operating system for memory-efficient networked embedded systems and wireless sensor networks. <http://www.contiki-os.org>.
- Dunkels, A., Osterlind, F., Tsiftes, N., & He, Z. (2007). Software-based on-line energy estimation for sensor nodes. In *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors* (pp. 28–32). New York, NY: ACM. doi:10.1145/1278972.1278979.
- Fasolo, E., Rossi, M., Widmer, J., & Zorzi, M. (2007). In-network aggregation techniques for wireless sensor networks: A survey. *Wireless Communications, IEEE*, 14(2), 70–87. doi:10.1109/MWC.2007.358967.
- Montenegro, G., Kushalnagar, N., & Culler, D. (2007). Transmission of IPv6 packets over IEEE 802.15.4 networks. In *IETF*.
- Hester, L., Huang, Y., Andric, O., Allen, A., & Chen, P. (2002). NeuRon netform: A self-organizing wireless sensor network. In *Proceedings of the eleventh international conference on computer communications and networks* (pp. 364–369).
- Hui, J., & Culler, D. (2008). Extending ip to low-power, wireless personal area networks. *Internet Computing, IEEE*, 12(4), 37–45. doi:10.1109/MIC.2008.79.
- IEEE Computer Society. (2006). IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs).
- IEEE-Computer-Society. (2006). IEEE Std 802.15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). Revision of IEEE Std 802.15.4-2003.
- IETF Network Working Group. (2003). Ad-hoc on-demand distance vector (AODV) routing algorithm. <http://tools.ietf.org/html/rfc3561>.
- Jain, R., Chiu, D. M., & Hawe, W. (1998). A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. CoRR cs.NI/9809099 (1998). <http://dblp.uni-trier.de/db/journals/corr/corr9809.html#cs-NI-9809099>.
- JENNIC. (2006). Calculating 802.15.4 data rates. Application Note: JN-AN-1035.
- Khan, M. R. H., Hossain, M. A., & Mukta, M. S. H. (2008). Zigbee cross layer optimization and protocol stack analysis on wireless sensor network for video surveillance. In *International conference on electronics, computer and communication (ICECC 2008)* (pp. 795–799). Bangladesh: University of Rajshahi.
- Koushanfar, F., Taft, N., & Potkonjak, M. (2006). Sleeping coordination for comprehensive sensing using isotonic regression and domatic partitions. In *INFOCOM 2006. Proceedings of 25th IEEE international conference on computer communications* (pp. 1–13). doi:10.1109/INFOCOM.2006.276.
- Latré, B., Mil, P. D., Moerman, I., Dhoedt, B., Demeester, P., & Dierdonck, N. V. (2006). Throughput and delay analysis of unslotted IEEE 802.15.4. *JNW*, 1(1), 20–28.
- Levis, P., Clausen, T., Hui, J., Gnawali, O., & Ko, J. (2011). The trickle algorithm. RFC 6206 (Proposed Standard). <http://www.ietf.org/rfc/rfc6206.txt>.
- Ma, T., Xu, Z., Hempel, M., Peng, D., & Sharif, H. (2014). Performance analysis of a novel low-complexity high-precision timing synchronization method for wireless sensor networks. *Wireless Communications, IEEE Transactions on*, 13(9), 4758–4765. doi:10.1109/TWC.2014.2331286.
- Marques, B. F., & Ricardo, M. P. (2011). Application-driven design to extend WSN lifetime. In *Proceedings of 1st Portuguese national conference on sensor networks (CNRS2011)*, Coimbra, Portugal.
- Marques, B. F., & Ricardo, M. P. (2014). Improving the energy efficiency of WSN by using application-layer topologies to constrain RPL-defined routing trees. In *Ad Hoc networking*

- workshop (MED-HOC-NET), 2014 13th annual Mediterranean (pp. 126–133). doi:[10.1109/MedHocNet.6849114](https://doi.org/10.1109/MedHocNet.6849114).
29. Martin, J. (1965). Distribution of the time through a directed acyclic network. *European Journal of Operations Research*, 13, 44–66.
 30. Mendes, L. D., & Rodrigues, J. J. (2011). A survey on cross-layer solutions for wireless sensor networks. *Journal of Network and Computer Applications* 34(2), 523–534. doi:[10.1016/j.jnca.2010.11.009](https://doi.org/10.1016/j.jnca.2010.11.009). <http://www.sciencedirect.com/science/article/pii/S1084804510002079>. Efficient and Robust Security and Services of Wireless Mesh Networks.
 31. Montenegro, G., Kushalnagar, N., Hui, J., & Culler, D. (2007). RFC 4944—Transmission of IPv6 packets over IEEE 802.15.4 networks. In *IETF RFC*.
 32. Nam, Y., Kwon, T., Lee, H., Jung, H., & Choi, Y. (2007). Guaranteeing the network lifetime in wireless sensor networks: A MAC layer approach. *Computer Communications*, 30(13), 2532–2545. doi:[10.1016/j.comcom.2007.05.031](https://doi.org/10.1016/j.comcom.2007.05.031).
 33. Organization, Z. S. (2008). Zigbee specification. <http://www.zigbee.com>. Document 053474r17.
 34. Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., & Voigt, T. (2006). Cross-level sensor network simulation with COOJA. In *Proceedings 2006 31st IEEE conference on local computer networks* (pp. 641–648). doi:[10.1109/LCN.2006.322172](https://doi.org/10.1109/LCN.2006.322172).
 35. Pantazis, N., Nikolidakis, S., & Vergados, D. (2013). Energy-efficient routing protocols in wireless sensor networks: A survey. *Communications Surveys Tutorials, IEEE*, 15(2), 551–591. doi:[10.1109/SURV.2012.062612.00084](https://doi.org/10.1109/SURV.2012.062612.00084).
 36. Paxson, V., Allman, M., Chu, H. J., & Sargent, M. (2011). Computing TCP's retransmission timer. <http://tools.ietf.org/html/rfc6298>.
 37. RaspBerry Pi. (2014). <http://www.raspberrypi.org>.
 38. Santi, P. (2005). Topology control in wireless ad hoc and sensor networks. *ACM Computing Surveys*, 37(2), 164–194. doi:[10.1145/1089733.1089736](https://doi.org/10.1145/1089733.1089736).
 39. Sturek, D. (2009). ZigBee IP stack overview. ZigBee Alliance.
 40. Tang, C. M., Zhang, Y., & Wu, Y. P. (2012). The P2P-RPL routing protocol research and implementation in contiki operating system. In *2012 Second international conference on instrumentation, measurement, computer, communication and control (IMCCC)* (pp. 1472–1475). doi:[10.1109/IMCCC.2012.345](https://doi.org/10.1109/IMCCC.2012.345).
 41. van Dam, T., & Langendoen, K. (2003). An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the 1st international conference on embedded networked sensor systems, SenSys '03* (pp. 171–180). New York, NY: ACM. doi:[10.1145/958491.958512](https://doi.org/10.1145/958491.958512).
 42. Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., et al. (2012) RPL: IPv6 routing protocol for low-power and lossy networks. In RFC 6550 (Proposed Standard). <http://www.ietf.org/rfc/rfc6550.txt>.
 43. Xie, W., Goyal, M., Hosseini, H., Martocci, J., Bashir, Y., Baccelli, E., et al. (2010) A performance analysis of point-to-point routing along a directed acyclic graph in low power and lossy networks. In *2010 13th International conference on Network-based information systems (NBIS)* (pp. 111–116). doi:[10.1109/NBIS.2010.65](https://doi.org/10.1109/NBIS.2010.65).
 44. Ye, W., Heidemann, J., & Estrin, D. (2002). An energy-efficient MAC protocol for wireless sensor networks. In *INFOCOM 2002. Proceedings of twenty-first annual joint conference of the IEEE computer and communications societies* (Vol. 3, pp. 1567–1576), IEEE. doi:[10.1109/INFCOM.2002.1019408](https://doi.org/10.1109/INFCOM.2002.1019408).



Bruno Marques received in 2001 a M.Sc. degree in Electrical and Computers Engineering from University of Porto. He is presently studying for a Ph.D. degree. He is an assistant professor at 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 Ph.D. degree in Electrical and Computers Engineering from Porto University. He is an associate professor at the Faculty of Engineering of 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.