Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2012, Article ID 651748, 29 pages
doi:10.1155/2012/651748

Research Article

Support for a Long Lifetime and Short End-to-End Delays with TDMA Protocols in Sensor Networks

Marcin Brzozowski, Hendrik Salomon, and Peter Langendoerfer

IHP, Technologiepark 25, 15236 Frankfurt, Germany

Correspondence should be addressed to Marcin Brzozowski, brzozowski@ihp-microelectronics.com

Received 15 December 2011; Revised 20 April 2012; Accepted 3 May 2012

Academic Editor: Cristina Cano

This work addresses a tough challenge of achieving two opposing goals: ensuring long lifetimes and supporting short end-to-end delays in sensor networks. Obviously, sensor nodes must wake up often to support short delays in multi-hop networks. As event occurs seldom in common applications, most wake-up are useless: nodes waste energy due to idle listening. We introduce a set of solutions, referred to as LETED (limiting end-to-end delays), which shorten the wake-up periods, reduce idle listening, and save energy. We exploit hardware features of available transceivers that allow early detection of idle wake-up periods. This feature is introduced on top of our approach to reduce idle listening stemming from clock drift owing to the estimation of run-time drift. To evaluate LETED and other MAC protocols that support short end-to-end delays we present an analytical model, which considers almost 30 hardware and software parameters. Our evaluation revealed that LETED reduces idle listening by 15x and more against similar solutions. Also, LETED outperforms other protocols and provides significant longer lifetimes. For example, nodes with LETED work 8x longer than those with a common TDMA and 2x-3x longer than with protocols based on preamble sampling, like B-MAC.

1. Introduction

Recent development in the electronic industry, especially miniaturization, allowed the use of tiny wireless devices with sensing abilities, referred to as sensor nodes. In general, sensor nodes are the size of a matchbox, work with batteries, and send data wirelessly. As they work for several months or years and do not need wires at all that is, they get power from batteries and send data wirelessly, sensor nodes provide a new set of applications. In general, nodes form a wireless network, monitor a specific area by reading sensors, and send sensor readings to a sink.

This work addresses mainly *critical infrastructure protection* (CIP) and similar scenarios. They are most challenging, since nodes must achieve two opposing goals: ensuring long lifetimes and supporting various quality-of-service features, usually short end-to-end delays. In general, nodes check an area for specific events, and on detection they must inform the sink within a predefined time. For example, sensor networks monitor gas leakage on factory facilities. As soon as they detect it, they send notices to the sink. However, to prevent explosion danger, the sink must receive the information about leakage within a few seconds after detection.

In CIP applications sensor nodes should work reliably for a long time, month or years, without human intervention. Besides, nodes cannot usually be mains-powered, as laying new cables to each node separately is not feasible. Therefore, they are powered by batteries, which should provide energy for a long time. To ensure long lifetimes, nodes apply low-duty cycle (LDC) protocols, which keep them mostly in the sleep state. As the current consumption in the sleep state is smaller by approximately. three orders of magnitude than that in the active state, sensor nodes increase the lifetime significantly. For example, Tmote Sky [1] nodes work only a few days in the active state. If the duty cycle is reduced to 0.1%, the lifetime increases to several years.

In general, nodes with LDC protocols wake up rarely to check for potential transmissions and then continue sleeping. Thus, on event detection the source node cannot send data immediately to the next node but waits until it becomes active. Similarly, in multihop networks each node waits until

the next node wakes up before sending data towards the sink. Obviously, it results in significant end-to-end delays and the sink may receive event notices too late. To counter this threat, nodes should wake up more often, but this increases the duty cycle and shortens the lifetime.

To guarantee two opposing goals, that is, short end-to-end delays and long lifetimes, several approaches (DMAC [2] and Q-MAC [3]) maintain wake-up slots in a staggered schedule, a type of TDMA (time division multiple access) approach. The idea resembles the common practice of synchronizing traffic lights to turn green (wake up) just in time of the arrival of vehicles (packets) from previous intersections (hops). Although the staggered schedule support shorts end-to-end delays, it suffers from the following problems:

(i) to support short end-to-end delays nodes wakes up often,

(ii) as events occur seldom in CIP application, most wake-up periods are useless that is, nodes do not receive data but only waste energy due to idle listening.

To cope with these problems, this work introduces the LETED (limiting end-to-end delays) protocol that shortens the wake-up periods and saves energy by applying the following means.

(1) With the ILA (Idle Listening Avoidance) approach, nodes detect idle wake-up periods in about 100 μs and early power down the transceiver. In this way, they reduce idle listening by 15x and prolong the lifetime by 30% and more.

(2) It applies energy-efficient approaches that deal with clock drift, based on drift prediction. By doing so, nodes reduce idle listening caused by clock drift by 95% against common solutions.

To examine the tradeoff between delays and lifetime, we introduced an analytical model that evaluates various MAC (medium access control) protocols. If nodes apply LETED with ILA, they support short end-to-end delay and long lifetimes. For instance, they work about 3 years and guarantee 5-second delays, even when they send frames to the sink once a minute, that is, our approach outperforms other MAC protocols in scenarios that need short end-to-end delays.

The remainder of this work is organized as follows. Section 2 introduces research efforts in duty-cycle MAC protocols and in efficient means to deal with clock drift problems. Section 3 presents the LETED solution. Idle listening avoidance (ILA) is introduced in Section 4. We evaluate LETED in a network simulator and present the results in Section 5. The next section provides the energy model and lifetime results of LETED and other protocols that support short end-to-end delays. Finally, we address some of our future work and conclude the paper.

2. Related Work

2.1. Duty-Cycled MAC Protocols. The main concern of many sensor network applications is a limited power source. An off-the-shelf sensor node Tmote Sky [1] with standard 2x AA batteries works only few days, if it keeps the transceiver and the microcontroller permanently powered up. However, nodes must provide much longer lifetimes, several months or years. To achieve such long lifetimes, sensor nodes apply low-duty cycle (LDC) protocols. Such protocols keep the nodes sleeping most of the time and wake them up for a short time only, for instance, to get sensor readings or receive data. However, to send and to receive data nodes must be awake at the same time, referred to as *rendezvous* [4]. Obviously, each node on a multihop route needs *rendezvous* with the next node towards the destination. Reference [4] grouping solutions to the *rendezvous* problem into three categories.

Asynchronous. In this solution nodes can wake up other nodes with a dedicated hardware, for example, wake-up radios [5]. In general, when a node wants to send data to its neighbor, it wakes up the neighbor and then sends data.

Pseudo-Asynchronous. Since nodes cannot wake up other nodes like in the *asynchronous* approach, they apply a software solution that tries to work like a wake-up radio. For example, nodes may periodically listen for potential transmissions and stay awake, if they detect a transmission wish. The following protocols belong to this group: STEM (sparse topology and energy management) [6], preamble sampling [7, 8], Berkeley Media Access Control (B-MAC) [9], Wireless Sensor MAC (WiseMAC) [10], TICER (Transmitter-Initiated CyclEd Receiver) and RICER (Receiver-Initiated CyclEd Receiver) [4], and Koala [11].

Synchronous. Nodes agree on specific communication time slots: they send and receive data only during such slots. In general, senders and receivers agree on a wake-up schedule and wake up at the same time to communicate. In general, TDMA (time division multiple access) protocols belong to this group, for example, sensor MAC (SMAC) [12], time-out MAC (T-MAC) [13], Flexible power scheduling [14] (FPS), Twinkle [15], and Dozer [16]. In addition, the standard IEEE 802.15.4 [17] defines the Medium Access Control (MAC) based on beacons, which also provides a synchronous *rendezvous*.

This work applies a distributed low-duty cycle MAC (DLDC-MAC) protocol, introduced in our previous works [18, 19]. Nodes with DLDC-MAC send periodically beacons and wake up to receive beacons of neighbors (see Figure 1). Obviously, this protocol resembles other TDMA approaches, for example, S-MAC, Dozer, or IEEE 802.15.4 in beacon-enabled networks. The main differences of DLDC-MAC from other protocols are as follows.

(i) Dozer is the closest relative of DLDC-MAC, as it uses beacons in a similar way and supports short active times. However, the main drawback of Dozer is that it supports only the tree topology. That is, children

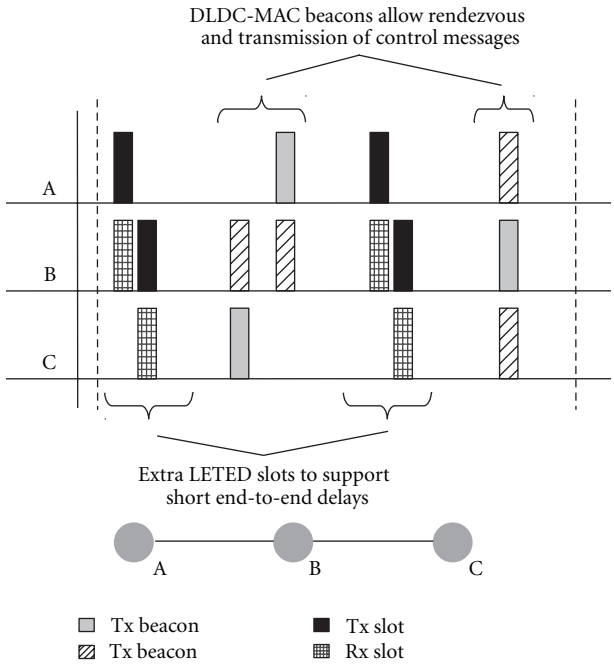


Figure 1: With DLDC-MAC nodes send periodically beacons and wake up to receive beacon from neighbors; to support short end-to-end delays nodes apply extra LETED slots.

receive only parent's beacons. Should communication problems on the link to the parent arise, the routing protocol discovers a new route to the sink. However, as nodes receive parent's beacon only they do not learn about neighbors and the routing cannot easily find alternative paths.

(ii) S-MAC reduces duty cycle, but the active periods are still long. First, it does not provide efficient way to deal with clock drift. According to [12], it may use guard times as long as 0.5 second, that is, longer by two orders of magnitude than the time needed to send a single frame. Second, S-MAC prolongs the active period by applying extra RTS and CTS frames. Third, S-MAC neglects important TDMA protocol problems, like overlap problem of two separate wake-up schedules.

In addition, DLDC-MAC supports data replication in sensor networks and handles several TDMA problems (details in [18, 19]).

Figure 1 depicts the basics of DLDC-MAC. As above said, nodes send periodically beacons and wake up to receive beacons from neighbors. In general, nodes send non-time-critical data included in beacons, for example, route discovery frames or to set up a new LETED schedule. According to the evaluation presented in Section 6, in such a configuration DLDC-MAC needs the energy amount that nodes consume in the sleep state. Besides, the DLDC-MAC energy consumption is less than the self-discharge of batteries. To support short delays, nodes set up extra LETED wake-up schedule on paths towards the sink (see details in Section 3).

2.2. Short End-to-End Delays. On the one hand, nodes reduce the duty cycle and mostly sleep to achieve long lifetimes. On the other hand, they need to wake up often in order to take part in potential data transfer and support short end-to-end delays. Clearly, there is a tradeoff between these two goals, that is, short delays and long lifetimes.

Some works investigated the tradeoff between delay and lifetime. The SMAC authors present in [12] energy savings versus average sleep delay tradeoff. Reference [8] presents the mean delay and achieved lifetime of CSMA (carrier sense multiple access) and of various TDMA (time division multiple access) approaches. The tradeoff relationship between the expected lifetime extension and the corresponding increase in the average detection delay achieved by different sleep scheduling algorithms is introduced in [20]. Reference [21] and explores the energy-latency tradeoff for broadcast communication in sensor networks. In [22] the authors examine the delay and lifetime trade-off from another point of view: the objective is to determine the best path from each node to a single gateway. Performance metrics of interest are the expected energy consumption and the probability that the latency exceeds a certain threshold. In [23] we examined the tradeoff between the end-to-end delay and the lifetime of a one-hop sensor network based on IEEE 802.15.4 connected to a IEEE 802.11 g network.

To preserve energy sensor nodes monitor the covered area periodically, that is, they keep sensors switched off for a long time. Clearly, it may result in a large event detection time (EDT), if an event occurs when all sensors are powered down. Reference [20] examines various schedule approaches of sensors that cover the same sensing area in order to minimize the average EDT. However, we do not address the problem of sensors' duty cycle in this paper.

This paragraph introduces main duty-cycled MAC approaches and their impact on delays in multihop networks.

2.2.1. Duty-Cycled TDMA. With common duty-cycled TDMA solutions, for example, SMAC [12] or Dozer [16], nodes mostly sleep and wake up on agreed times to communicate. Therefore, should source nodes detect events, they cannot send data immediately to the next node. They wait until the next node is awake and then forward frames (see Figure 2). Similarly, each node on the path to the sink waits until the next node wakes up.

To support short end-to-end delays, nodes wake up often to take part in potential transmissions. Clearly, there is a tradeoff between a long lifetime and short delays, that is, between long and short sleep periods respectively. On average end-to-end delay d_{EtE} depends on the sleep period T_{sleep}:

$$d_{\mathrm{EtE}} = n \cdot \left(\frac{T_{\mathrm{sleep}}}{2} + t_{\mathrm{frame}} \right), \tag{1}$$

where n is the number of hops to the sink and t_{frame} the frame length. Therefore, to support certain end-to-end delays, nodes adapt the sleep period in the following way:

$$T_{\mathrm{sleep}} = \frac{d_{\mathrm{EtE}}}{n} - t_{\mathrm{frame}}. \tag{2}$$

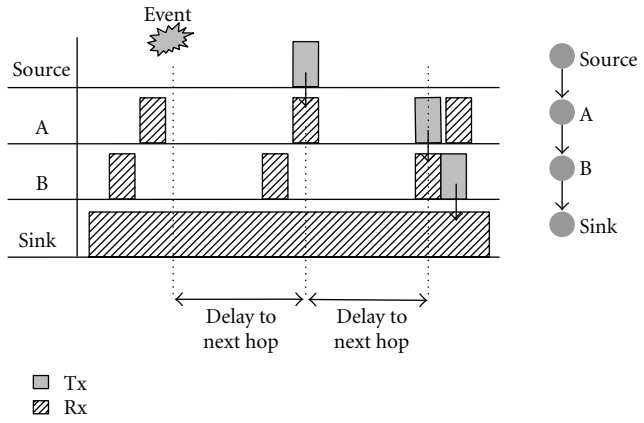


Figure 2: By applying low-duty cycle protocols, nodes mostly sleep and cannot forward data immediately but wait until the next node is awake. It causes significant end-to-end delays.

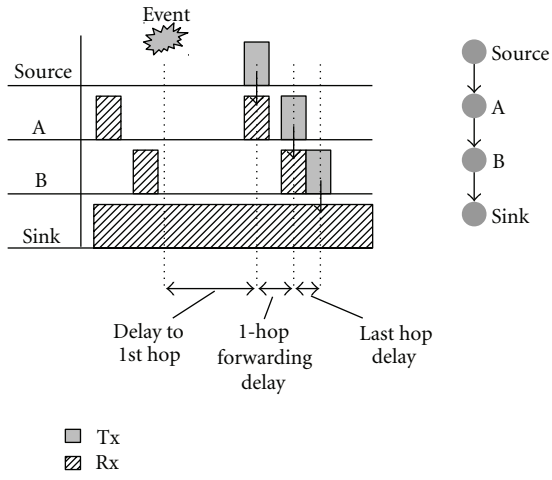


Figure 3: Nodes with staggered (aligned) schedule forwards frames just after reception and reduce end-to-end delays.

Figure 4 shows sleep periods needed to achieve certain end-to-end delays. In general, nodes wake up the period equal to the supported delay divided by the number of hops, that is, to support 5-second delays in 2-hop networks nodes wake up every 2.5 seconds. Consequently, in larger networks nodes wake up more often to support the same delay. For instance, in 10-hop networks nodes wake up every half a second to support delays of 5 seconds. Thus, should nodes apply LDC protocols but keep end-to-end delays short, they increase the duty cycle and shorten the lifetime significantly, especially in large networks.

2.2.2. Staggered Schedule. Several protocols, for example, DMAC [2], Q-MAC [3] and reference [20] introduced the staggered schedule (see Figure 3) to support short end-to-end delays and low duty cycles. It resembles the common practice of synchronizing traffic lights to turn green (wake up) just in time of the arrival of cars, that is, packets, from previous intersections (hops). Nodes on the path arrange

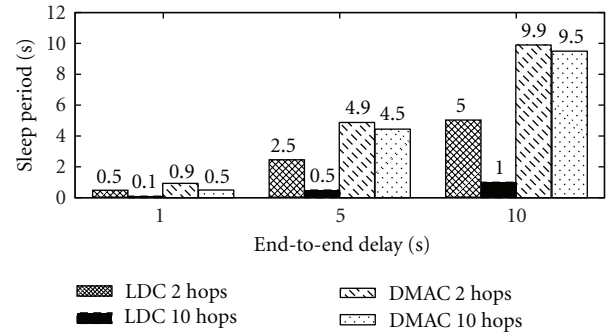


Figure 4: To support short end-to-end delays, nodes with common low duty cycle (LDC) protocols wake up often, especially when the distance between sources and the sink is long; DMAC introduces the staggered schedule and lowers the duty cycle; that is, nodes wake up more rarely than in common LDC protocols to support short delays; besides, with DMAC the distance to the sink only slightly impacts the duty cycle.

slots in a way that tx slots follows almost immediately rx slots. In that way, nodes forward messages just after the reception and keep the forwarding delay short. Therefore, the number of hops only slightly influence end-to-end delays. Obviously, the shorter the needed end-to-end delay is, the more often nodes have to wake up to take part in potential data transmission. In general, only the source node waits a long time for the next node to wake up (see Figure 3). On average end-to-end delay equals to:

$$d_{\mathrm{EtE}} = \frac{T_{\mathrm{sleep}}}{2} + t_{\mathrm{frame}} + (n-1)\cdot(t_{\mathrm{frame}} + t_{\mathrm{offset}}), \quad (3)$$

where t_{offset} in the time between the rx slot and the corresponding tx slot on each node. Should the sink does not apply a wake-up schedule, as it is in Figure 3, the number of hops n is reduced by 2 in (3). Nodes adapt the sleep period T_{sleep} to support certain end-to-end delays d_{EtE}:

$$T_{\mathrm{sleep}} = d_{\mathrm{EtE}} - t_{\mathrm{frame}} - (n-1)\cdot(t_{\mathrm{frame}} + t_{\mathrm{offset}}). \quad (4)$$

Figure 4 compares the duty cycle, that is, the sleep period, of the staggered schedule and LDC protocols. As previously mentioned, the distance between sources and the sink only slightly impacts the duty cycle of the staggered schedule. For example, to support 5-second delays nodes wake up 4.9 seconds in 2-hop networks. Should the path to the sink be 10-hop long, nodes wake up every 4.5 seconds. As expected, the staggered schedule outperforms common LDC protocols in such scenarios, especially in large networks. With 10-hop distance to the sink it reduces the duty cycle about 10x.

2.2.3. Preamble Sampling/Cycled Receiver. The protocols that support asynchronous *rendezvous,* which were mentioned previously, synchronize wake-up times by sending long preambles or wake-up beacons. In general, nodes wakes up periodically to listen for potential transmissions (see Figure 5). If a node wishes to send a frame, it sends a long preamble or many short frames, in front of the data frame. In the worst case, the preamble length equals the sleep period of

receivers. On getting the preamble, the receiver stays awake and gets the data frame.

Although these protocols were not designed to primarily support end-to-end delays, they can adapt the sleep period and limit the delays. In general, end-to-end delay d_{EtE} consists of single forwarding delays along the path:

$$d_{\text{EtE}} = \sum_{i=i}^{n} (t_n + t_{\text{frame}}), \tag{5}$$

where t_n is the forwarding delay on node i and t_{frame} the frame length. As the average forwarding delay equals the half of the sleep period T_{preamble}, which equals the worst case preamble length, the average end-to-end delay of a n-hop path is estimated as:

$$d_{\text{EtE}} = n \cdot \left(\frac{T_{\text{preamble}}}{2} + t_{\text{frame}} \right). \tag{6}$$

2.3. Clock Drift Compensation. Sensor nodes derive time from crystal oscillators, which have certain precision δ, expressed in parts per million (ppm), according to the crystal cut. That is, such oscillators provide the system time that differs from the perfect clock by δ. Therefore, in the worst case, clocks of two sensor nodes move apart by 2δ. For example, precision of oscillators applied on Tmote Sky nodes is 20 ppm and results in worst-case drift of 2.4 ms in 1-minute period among two nodes. In addition, changes of temperature and air pressure cause short-term drift variations.

Each scheduled MAC protocol suffers from the drift problem explained in Figure 6. According to the schedule receivers wake up at specific times to get data from neighbors. However, as clocks of senders and receivers may run at different speeds, referred to as clock drift, there is a risk that receivers wake up too late and miss frames. To counter this threat, they wake up earlier by guard times and compensate drift in this way. Clearly, as guard times result in extra idle listening, nodes should keep them short.

Several approaches (e.g., Dozer [16]) use guard times based on worst-case drift. However, the authors estimate worst-case drift only from the oscillator parameter and neglect other reasons for example, the time to power up radios is not constant. Besides, as run-time drift is smaller than the worst case, such solutions may waste energy due to unnecessary long idle listening. For example, it results in guard times of a few ms and more for a sleep period of 1 minute. Such long guard times are about as long as the time needed to send a frame. Clearly, it causes long idle listening and wastes energy.

To shorten guard times, reference [24] introduces sensor nodes equipped with two oscillators, reducing frequency stability to ± 1.2 ppm. Another work [25] applies the LR (linear regression) to previous drift samples and shorten guard times.

In [26] we introduced the estimation of guard times based on the moving average filter, referred to as MADC (moving average drift compensation). Based on previous drift samples, nodes predict future drift and can use short guard times. For instance, to compensate drift of 99% frames in outdoor scenarios, nodes with MADC apply guard times of about 130 μs for 1-minute sleep period. With common solutions to the drift problem, they need guard times of 2.4 ms. As MADC reduces idle listening, it saves energy and prolongs the lifetime. Our evaluation revealed that nodes with the IEEE 802.15.4 MAC work longer by 5% and more owing to the MADC approach (see details in [26]). Owing to its simplicity, MADC can be easily applied to sensor nodes, as it does not need the floating-point arithmetic. In addition, it works well with only 3 previous drift samples and thus do not occupy too much memory. Therefore, the solutions presented in this work apply MADC to efficiently deal with clock drift problems.

3. Limiting End-to-End Delays (LETED)

This chapter introduces LETED, that is, a set of solutions that limit end-to-end delays in sensor networks. LETED adapts previous approaches, that is, DMAC [2] and Q-MAC [3], to save energy and prolong the lifetime. Besides, it handles drift problems neglected in previous works.

In this work we coupled LETED with DLDC-MAC (see Figure 1). That is, nodes wake up to send and to receive beacons. Also, they arrange LETED wake-up schedule along paths to the sink. Although LETED is based on DLDC-MAC here, it works with other MAC approaches as well.

We already introduced LETED in our previous work [27] but presented only analytical results of LETED performance. Later we implemented LETED and evaluated it with a network simulator, which revealed some drawbacks of this protocol. This section presents the improved version of LETED with extra simulative evaluation.

3.1. Overview. LETED adapts the staggered schedule introduced in DMAC [2]. Nodes on the path to the sink settle a wake-up schedule in a way that it limits end-to-end delays. That is, each tx slot follows immediately the rx slot from the previous node. Therefore, nodes send messages just after reception.

Nodes with LETED start transmissions exactly at tx slots that is, they do not apply CSMA/CA (carrier sense multiple access with collision avoidance) or similar solutions, which may postpone transmissions. Due to the scheduling approach presented here, nodes usually do not need such means. Since the schedule is a TDMA approach, it inherently avoids contention. However, because of clock drift, slots may overlap and cause a collision risk, as introduced in Section 3.5. Nonetheless, extra medium access means cause excessive idle listening and shorten the lifetime. Therefore, LETED does not apply such medium access means.

To deal with unreliable wireless links, nodes apply the ARQ [28] protocol. That is, receivers send an acknowledgment (ACK) to senders on frame reception. Should senders do not receive ACKs, they assume the frame was lost and send it again. The number of tx attempts and the delay between successive retries depends on the application.

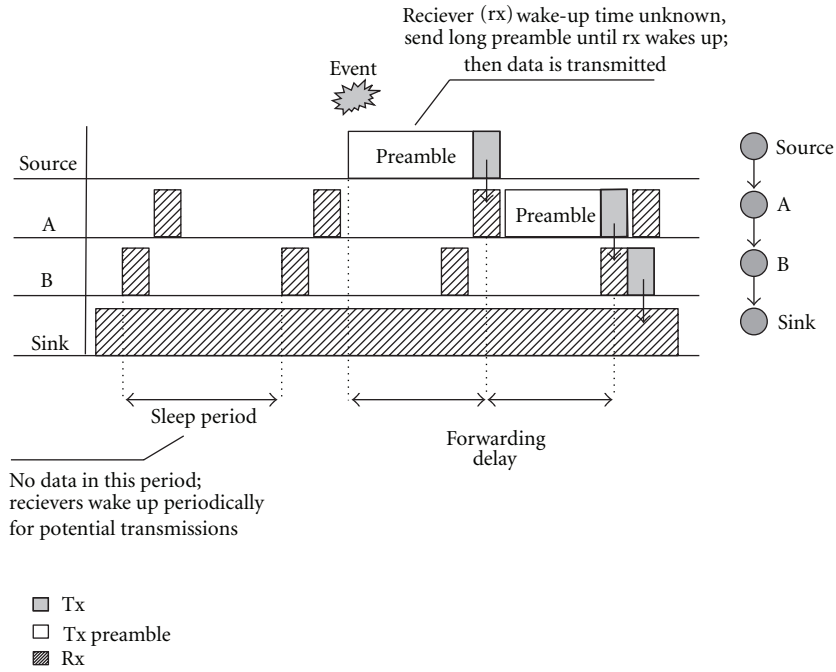


Figure 5: With preamble sampling (cycled receiver) nodes send long preambles in front of frames and receivers check periodically for transmissions. The length of preamble and of sleep periods impacts end-to-end delays resulting in significant end-to-end delays.

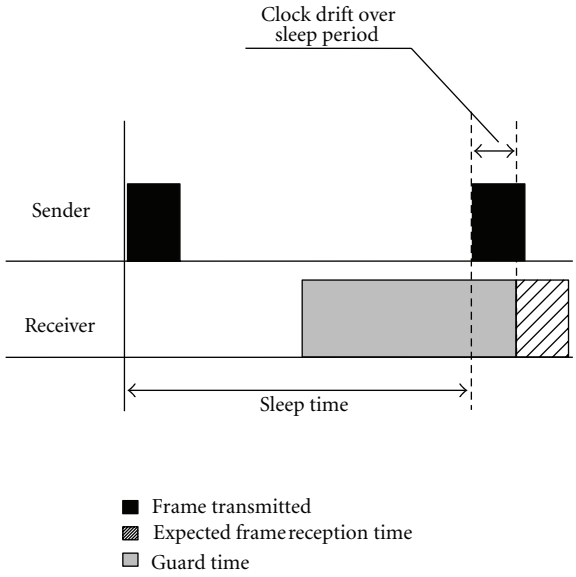


Figure 6: To compensate clock drift receivers wake up earlier by guard times.

3.2. Schedule Setup. The schedule setup involves cross-layer cooperation among the application, network, and MAC layers (see Figure 7). First, the application triggers the network layer to set up a new schedule on the path towards the sink. The application specifies the longest acceptable end-to-end delay d_{EtE}. Second, the network layer triggers the MAC layer, that is, LETED with DLDC-MAC in this case, to set up new time slots with the next node. Based on the hop distance to the sink, provided by the routing, LETED calculates how often nodes must wake up to support d_{EtE}. That is, to support d_{EtE} the source node needs tx slots every T_{slot} time. Since intermediate nodes cause extra delays (see Figure 3 in the previous section), the source estimates the total forwarding delay $d_{\text{forwarding}}$ and the slot period T_{slot} as

$$T_{\text{slot}} = d_{\text{EtE}} - d_{\text{forwarding}}$$
$$d_{\text{forwarding}} = n \cdot (t_{\text{frame}} + t_{\text{tx\_offset}}), \tag{7}$$

where n is the number of hops to the sink, t_{frame} the expected frame size, and $t_{\text{tx\_offset}}$ the time between rx and tx slots on intermediate nodes. All nodes on the path apply the same value for $t_{\text{tx\_offset}}$.

After the source estimated the slot period, it adds new tx slots to the schedule. Then, the source sends a frame with the new tx times to the next node. On receiving it, the next node adds rx slots to its schedule and sends back an acknowledgment. If the new slots overlap with existing ones, the node answers with a negative acknowledgment (NACK). The node includes preferred time slots in the NACK. In this case, the source shifts the tx slots and sends a frame with the new times again.

In next steps each node on the path sets up time slots to the next node in a similar way.

In this work LETED benefits from the underlying DLDC-MAC and sends control frames, that is, new tx times and ACKs, piggybacked in beacons. In that way nodes keep the LETED overhead small, that is, only several extra bytes in beacons.

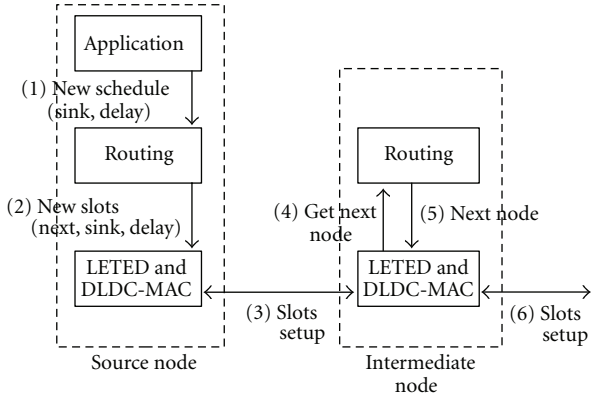


Figure 7: Setup of a new wake-up schedule; application triggers the network layer (routing) to set up a new schedule to the sink; on each node, the network layer (routing) requests MAC to establish time slots to the next node.

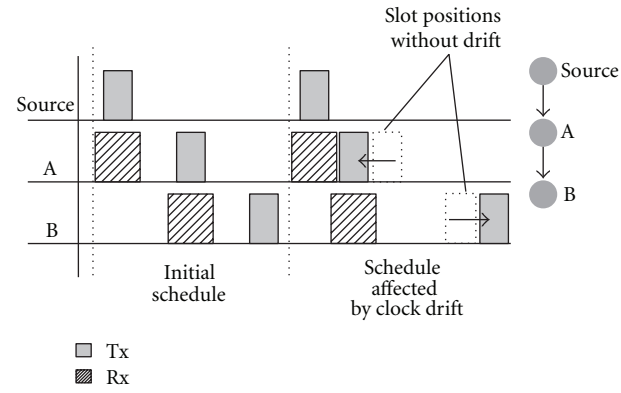


Figure 8: Due to clock drift, slots of different nodes move relatively to each other.

3.3. Guard Times. Since LETED is a TDMA protocol, it suffers from the drift problem. That is, receivers may wake up too late because of clock drift and miss frames. To counter this threat, LETED applies the solution based on drift prediction introduced in our previous work [26], referred to as MADC (moving average drift compensation). In short, nodes estimate run-time drift to neighbors by applying the moving average filter. Then, nodes calculate the time difference (drift) to the sender arisen since the last synchronization, that is, beacon reception in this work. Finally, they apply guard times long enough to compensate drift.

With MADC nodes miss about 1% frames due to not compensated drift. However, since LETED applies the ARQ protocol, the number of frames missed due to clock drift is smaller. That is, should nodes apply too short guard times and miss a frame, they can still receive it owing to ARQ retries.

3.4. Slot Synchronization. Due to clock drift timeslots of different nodes move relatively to each other, as presented in Figure 8. Should slots move towards each other, they finally overlap and pose a collision risk. For example, if relative drift between nodes A and B is 3 ppm (parts per million) and slot B follows slot A after 50 ms, the slots overlap after about 3.5 hours.

If slots drift away, the forwarding delay increases. Besides, if slots keep moving relatively to each other, they become unorganized and cannot support short end-to-end delays. To counter this threat, nodes adapt repeatedly the schedule according to relative drift to the source; that is, the timeslots remain stable relatively to the source tx slots. As a result, the end-to-end delay remains constant and time slots do not overlap.

Obviously, each node must find relative drift to the source in order to shift the time slots. In this example DLDC-MAC provides estimated run-time drift. (DLDC-MAC measures run-time drift to neighbors each time a beacon is received.) Each node with a schedule sends to the next node its relative drift to the source repeatedly, piggybacked in DLDC-MAC beacons in this example. On receiving relative drift to the source from the previous hop, nodes add drift to the last sender (neighbor). In that way, each node estimates relative drift to the source.

In general, nodes shift LETED slots in the following way.

(1) After an rx time slot finishes, nodes calculate the time of this slot $\mathrm{rx}_{\mathrm{next}}$ in the next beacon period as.

$$\mathrm{rx}_{\mathrm{next}} = \mathrm{rx}_{\mathrm{now}} + T_{\mathrm{beacon}} + \delta_{\mathrm{src}} \cdot T_{\mathrm{beacon}} - g, \tag{8}$$

where T_{beacon} is the beacon period and g is the guard time used for the next slot. Clearly, nodes adapt the schedule according to relative drift to the source δ_{src}.

(2) Nodes handle tx slot shifts in a similar way; that is, they estimate the slot time in the next beacon period as

$$\mathrm{tx}_{\mathrm{next}} = \mathrm{tx}_{\mathrm{now}} + T_{\mathrm{beacon}} + \delta_{\mathrm{src}} \cdot T_{\mathrm{beacon}}, \tag{9}$$

where $\mathrm{tx}_{\mathrm{now}}$ is the time of the slot just finished.

As nodes may estimate drift not exactly enough, they still suffer from the drift problem. That is, slots of senders and receivers still move apart (see Figure 10). Small time differences between tx and rx slots are compensated with guard times. However, slots keep moving apart, and the time difference becomes larger than guard times. In this case, slots are not synchronized and receivers miss frames (see slots on the right in Figure 10).

To deal with errors in drift estimation, sources send extra synchronization (SYNC) frames along paths and nodes synchronize all slots of the corresponding schedule. In other words, receivers calculate the time t_{diff} the slot drifted from the expected time t_{expected}:

$$t_{\mathrm{diff}} = t_{\mathrm{expected}} - t_{\mathrm{rx}}, \tag{10}$$

where t_{tx} is the frame reception time. Then, receivers shift all slots of this schedule by t_{diff}. In this way, slots are synchronized again.

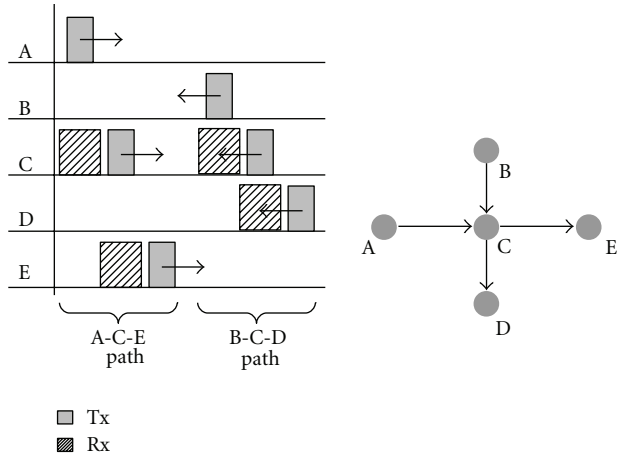


Figure 9: In LETED schedules of different sources may move relatively to each other and pose an overlap risk.

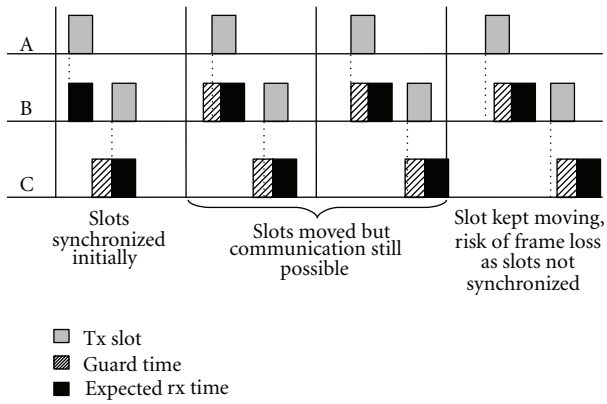


Figure 10: Nodes shift slots according to relative drift to the source. Because of errors in multihop drift estimation, they shift slots by different times. Thus, slots of senders and receivers drift away.

Clearly, the frequency of frame transmission depends on the scenario, for example, the accuracy of drift estimation or changes in external conditions that affect drift. Figure 11 depicts the time after LETED slots move by 1 ms for various precision of drift estimation. For example, with the drift estimation accuracy of 1 ppm, nodes should synchronize slots every 16 minutes to keep slots not drifted by more than 1 ms.

Another reason for errors in drift estimation are postponed SYNC frames transmissions, for example, because of variable delays in software execution and on transceivers. As delayed transmissions result in drift estimation errors, nodes should include tx timestamps in SYNC frames. By doing so, transmission delays would not result in drift estimation errors. For example, Chipcon CC2420 transceiver and MSP430 MCU on Tmote Sky nodes provide accurate hardware timestamps. On hardware events, like transmission of the start frame delimiter (SFD), MSP430 stores the current timer register. As it takes less than 100 μs to handle the timer interrupt, nodes manage to add the exact tx time to the frame that is being transmitted.

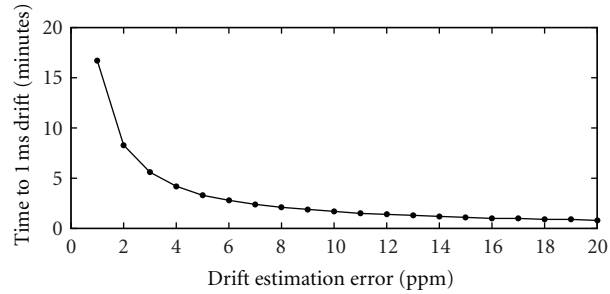


Figure 11: LETED slots move by 1 ms due to errors in drift estimation after the time depicted here.

3.5. Overlap Risk. Due to clock drift, LETED slots and beacons overlap and pose a collision risk. Although DLDC-MAC solves the beacon overlap problem, it does handle the risk of collisions with LETED slots. Therefore, nodes with LETED need to apply a solution that avoids the collision risk of LETED slots and beacons.

The previous paragraph introduced the solution to the problem of timeslot synchronization. That is, nodes shift the schedule repeatedly and keep the slot times unchanged relatively to the source. However, two independent schedules drift relatively to each other and cause an overlap risk, as presented in Figure 9. Besides, LETED slots overlap with beacons of DLDC-MAC as well. Clearly, on timeslot overlap nodes may not receive data due to collisions.

To detect an overlap, risk nodes look up the local slot table, which contains LETED slots and beacons with their start and finish times. However, nodes do not detect all overlap cases, since they do not learn about LETED schedules of neighbors that are on different routes. Figure 12 explains the problem. There are two independent routes to a sink, that is, A-B and C-D. Both paths set up separate wake-up schedules to support certain end-to-end delays. However, nodes A and B do not learn about LETED slots of nodes C and D, and vice versa. Therefore, should their slots overlap, they do not detect it and collisions occur. A similar case presents Figure 13 but both paths, that is, A-B and C-D, are not within their transmission range. Nonetheless, they still affect one another, as the transmission signal from another path increases the noise level on receivers.

When a collision occurs and nodes do not send affected frames again, the sink does not receive data. Besides, should nodes send frames again but on a later time, that is, in the next tx slot, the sink receives data too late. Clearly, frequent collisions, and indirectly a huge number of overlap cases, increase the packet error rate. Figure 14 depicts the average time to an overlap case of nodes that support 10-second end-to-end delays and receive beacons from 4 neighbors. For example, with relative drift among nodes of 8 ppm, slots overlap after less than an hour.

In our previous work [27], we introduced a solution to the overlap problem. However, simulation runs revealed some drawbacks of this approach, and therefore we apply a simpler but a robust solution based on the ARQ protocol. This approach aims to deal reasonably well with the overlap

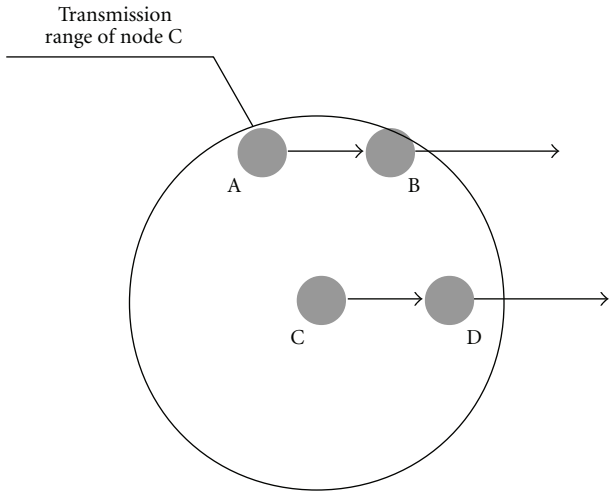


Figure 12: Two independent LETED paths are within their transmission range. As nodes do not learn about wake-up schedule of other paths, they cannot detect overlap risk between independent paths.

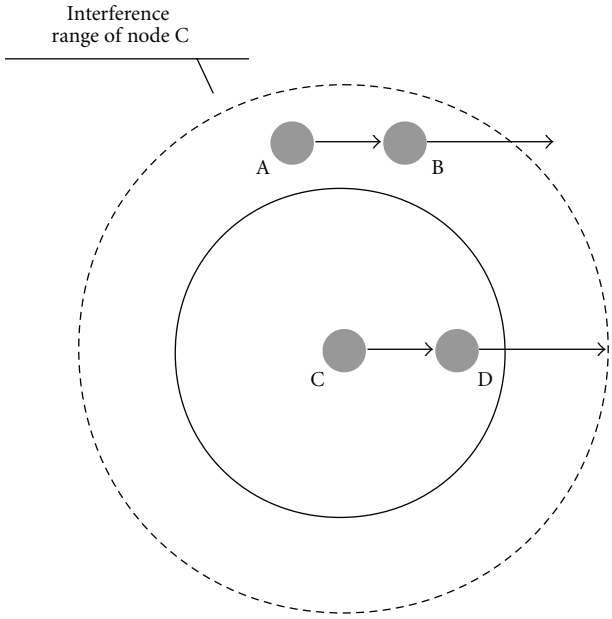


Figure 13: Although both paths are out of their transmission range, they are within the interference range and affect each other.

risk but remains simple to occupy only a fraction of sensor node memory. It exploits the nature of low-duty cycle applications: nodes rarely send data. Therefore, even when LETED slots overlap with other slots, they usually do not cause collisions, as they are mostly idle. For that reason nodes do not shift LETED slots, if they overlap with other LETED slots or with beacons. On the contrary, on the beacon overlap risk, nodes shift one of them according to the DLDC-MAC (details in our previous works [18, 19]).

In general, each node detects an overlap by comparing start and finish times of slots stored in the slot table. As

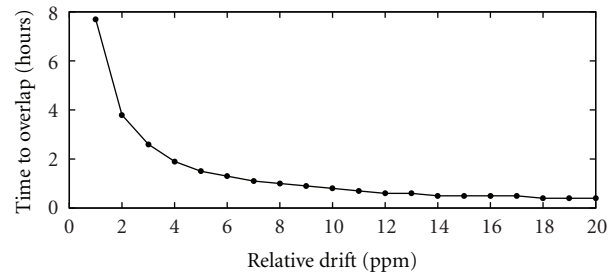


Figure 14: Average time to slot overlap, beacons or LETED slots, of nodes having four neighbors and a schedule supporting 10-second delays for different clock drift values.

stated above, nodes do not shift LETED slots, if they cause the overlap risk. Nonetheless, nodes must react to this, either skip or use the affected slots. The rule is to use the slot with a higher priority and skip other slots. However, before skipping a slot, nodes check if the slot can be partly used. For example, nodes skip the beginning of an rx slot, as it overlaps with a beacon, but try to receive retries afterwards. Besides, should all affected slots be rx slots, either LETED or beacon, the node switches the transceiver into the listening state for the time of both slots.

Table 1 depicts slot priorities used in this work. Nodes favor beacons over LETED slots, as they use beacons for the wake-up synchronization and any missed beacons result in longer guard times. Besides, nodes usually apply ARQ protocol to LETED slots in order to deal with unreliable wireless links. Thus, if a node skips a part of LETED slot, it can still send or receive ARQ retries.

Figure 15 presents handling of slot overlap. In this case node A detects an overlap of tx beacon and of an LETED rx slot. According to the slot priorities, see Table 1, nodes favor beacons over LETED slots. However, in this case the beacon covers only the beginning of the LETED slot; the remaining ARQ retries are not affected (see Figure 15(a)). Therefore, the nodes do not skip the LETED slots but only shorten it (see Figure 15(b)). As a result, node A can receive data, although there is an overlap risk with a beacon.

3.6. Topology Change. As nodes may stop working or suffer from various communication problems, source nodes cannot send data to the sink using the current routes. In these cases, routing protocols establish a new path towards the sink. However, the nodes on the new path must also set up a wake-up schedule to support short end-to-end delays. It involves the interaction between routing and MAC protocols depicted in Figure 16. In the following we explain shortly the steps presented in this figure.

(1) A node on the path to the sink discovers that it cannot send frames to the next node. In this case, DLDC-MAC does not receive several consecutive beacons from the next node and assumes the link is broken.

(2) LETED informs the routing protocol about the link failure.

Table 1: Priority of slots in the ARQ-based solution to the overlap problem; in the case of overlap nodes skip or shorten the slot with a lower priority.

Slot type	Priority
Beacon Tx	4
Beacon Rx	3
LETED Tx	2
LETED Rx	1

(3) The routing protocol sends a message to the source node, and the source discovers a new path.

(4) Routing at the source triggers LETED to establish a new schedule along the path, as introduced in Section 3.2. However, if the routing protocol supports local repairs, that is, discovering new paths without notifying the source like in AODV [29], the intermediate node discovers new routes and sets up wake-up schedule towards the sink.

Clearly, handling of topology change problems lasts a long time, as source node must discover a new path and establish a new wake-up schedule. As source nodes cannot send data to the sink during the discovery time, the network cannot guarantee required end-to-end delays. Therefore, in critical applications, the network should maintain many paths and wake-up schedules to the sink. In this case, nodes switch to another path immediately after detecting a link failure or send frames to the sink over several paths simultaneously. Although it involves extra overhead, as nodes on alternative paths wake up as well, it provides more reliable communications. However, the details of this solutions, like the number of paths that guarantee a certain degree of reliability or the impact on the lifetime, are beyond scope of this paper and need further research efforts. Besides, we intend to examine the impact of various solutions to the link failure detection on end-to-end delays and the lifetime. For example, link failures are detected after missing a certain number of frames from a neighbor. Obviously, too small numbers result in false alarms, whereas using large values increases the delay of finding new paths.

In our future work we intend to examine asymmetric links and load balancing too. We discuss it shortly later in this work, in Section 7.

4. Idle Listening Avoidance

This section introduces briefly a hardware support for LETED that reduces the idle listening time and prolongs the lifetime significantly. For more details please refer to our previous work [30].

4.1. Problem Statement. To support end-to-end delays, nodes must wake up at each rx slot. After waking up nodes listen for a time needed to receive a frame from the previous node. If no frame arrives, nodes power down the transceiver and continue sleeping. Such slots are referred to as *passive* slots. However, if nodes receive a frame from the previous node, they send it to the next node towards the sink in the following tx slot. Such slots are referred to as *active* in this work.

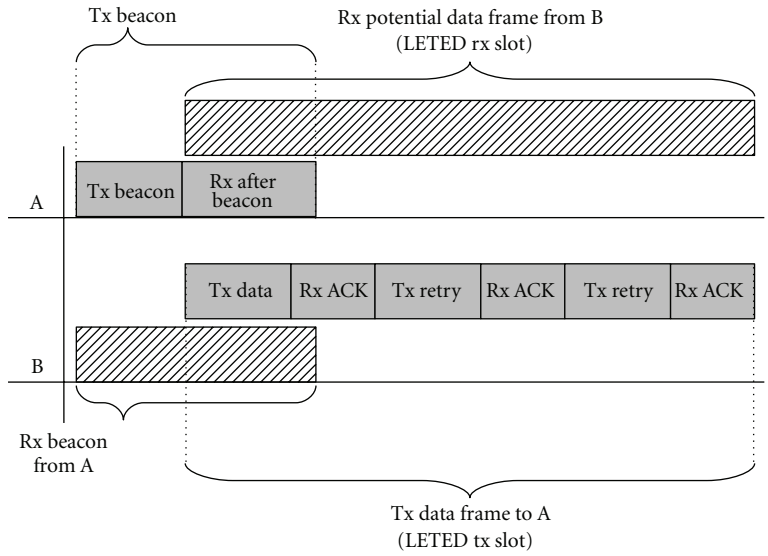
Figure 17(a) shows an active rx slot with a common software approach. After getting the preamble (Receivers use preambles to detect a new frame, the frame start/end, and to synchronize bits and symbols) and the following start frame delimiter (SFD) nodes receive the payload. Then, the payload is delivered to the application; that is, usually the transceiver drives the rx pin high and the microcontroller (μC) raises a receive interrupt (RxINT). After that, an interrupt service routine (ISR) of the operating system (OS) reads the payload from the rx buffer of the transceiver and delivers it to the application. Finally, the application calls an OS function to switch the transceiver off.

Figure 18 depicts the current consumption of various rx slot phases measured with an oscilloscope connected to Tmote Sky sensor node. In this example the node receives a 62-byte-long MAC frame of IEEE 802.15.4 standard. To compensate clock drift, the node wakes up 2 ms earlier than the expected time of incoming frame. In this case the node consumes an unnecessarily huge amount of energy; that is, it draws about 22 mA of current, for a time 3x longer than the frame itself, leading to the following problems in passive and active slots.

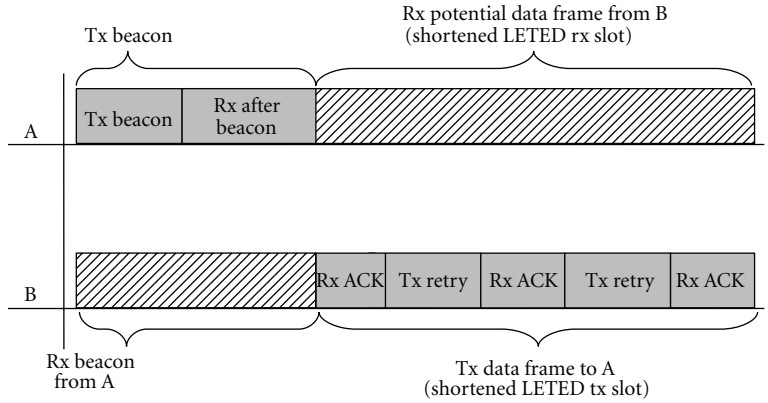
Passive Slots. Applications running on sensor nodes can only detect that a packet is received, when the operating system (OS) calls an rx routine, that is, after getting the message from the rx buffer. If no frame is received, the application will not know about it. The only indirect means to detect frame reception is to wait the normal time it takes from waking up till the OS calls the rx routine. The application powers down the transmitter as soon as this time interval has expired w/o any rx interrupt; see Figure 17(a). However, handling of RxINT and getting a frame from the rx buffer may last much longer than the frame reception; see Figure 18. Besides, if the underlying protocols use frames of various length, the application considers the longest frame when waiting for RxINT. Obviously, the indirect detection of idle rx slots causes unnecessary long idle listening.

In our previous work [30] we estimated the RxINT time. For the payload of 42 bytes, the shortest RxINT handler took 3.17 ms and the longest 3.23 ms. However, the time was significantly longer for 127-byte payload: from 8.29 to 8.35 ms. The reason for this is the long time the μC needs to read data from the rx buffer of the transceiver using the software-based SPI (serial peripheral interface bus), that is, without hardware accelerators.

Active Slots. In general, after frame reception the transceiver stays in the receive state until μC powers it down; for example, in CC2420 transceiver [31] μC writes a special command to a strobe register. Before software can power down the transceiver, it needs to read and handle the frame payload during ISR to learn whether other frames will follow. After that it can signal the μC to power down the transceiver; see Figure 17(a). Of course, if no frames follow the one just received, the transceiver should be powered down



(a) Slot overlap risk: Tx beacon of node A overlaps with LETED slot from B to A



(b) Joined slots: after beacon finishes there, nodes A and B use the rest of the LETED slot

Figure 15: Joining and shortening of slots on overlap risk.

immediately after receiving the last byte of the incoming frame to reduce idle listening. However, a node using a software-based solution handles RxINT, reads the whole message, and then powers down the transceiver. Thus, the software solution causes idle listening also in active slots.

4.2. Idle Listening Avoidance (ILA) Solution. Some commercial transceivers (e.g., CC2420 [31] used in Tmote Sky [1] nodes) offer extra features, such as capturing the exact time of SFD and raising an interrupt after SFD reception. (CC2420 just sets SFD pin to high/low. In Tmote Sky SFD pin is connected to a μC pin that is configured to raise an interrupt either on a falling or on a rising edge.) As it takes only 91 μs to raise the SFD interrupt on Tmote Sky nodes (see our previous work [30]), we exploit this feature to detect passive slots in an early stage and shorten idle listening. This solution, referred to as idle listening avoidance (ILA), is presented in the following.

Passive Slots. To reduce idle listening of passive slots, nodes need an indicator that determines as early as possible whether a frame arrives. Receiving a preamble and SFD indicates that a frame is to be received. Thus, if the node does not receive SFD in the expected time; that is, the estimated time of SFD includes guard time, preamble, SFD itself, and SFD interrupt handler, it assumes that no frame arrives in this slot; see Figure 17(b). As the SFD detect time is short on Tmote Sky, less than 100 μs, waiting for this time only before powering down the node shortens idle listening during passive slots considerably.

Active Slots. After receiving a payload, nodes should power down the transceiver quickly, if no frames follow the one just received; see Figure 17(c). When Tmote Sky receives a frame, it raises two interrupts: the first after SFD detection and the second when it receives the whole frame. The second interrupt means only that the transceiver stored the frame in rx buffer and μC must retrieve it, which takes a few ms. Clearly, the node can get the frame from rx buffer while the

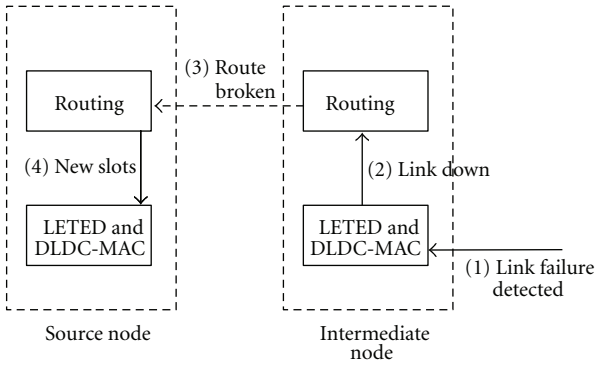


Figure 16: When a node detects a link failure (at the MAC level), it triggers the network layer (routing) to find a new path; a *route broken* message is sent to the source, which discovers another route and establishes a schedule along the new path.

main transceiver parts are powered off, resulting in energy savings. However, if another frame follows the one just received, the node needs to power up the transceiver again. As it takes a few ms to start up the transceiver, the node may miss the frame. Therefore, in this work nodes do not apply the solution to active slots.

4.3. ASIC Solution. The optimal solution for idle listening reduction involves the use of an application-specific integrated circuit (ASIC), which causes shortest delays in SFD detection and switching off the transceiver. In general, such a circuit consists of an extra logic and shorten idle listening in the following way.

(1) Passive Slots. Similar to ILA solution, ASIC should switch off the transceiver immediately if SFD is not received within a desired time; see Figure 17(b). Clearly, the SFD detection time is shorter on ASIC than the time of SFD detection on CC2420 transceiver.

(2) Active Slots. After receiving a frame, ASIC reads and evaluates the payload very quickly; that is, a few of microseconds, in order to check whether another frame follows the one just received. Therefore, ASIC must be aware of the message format to determine whether another frame follows the one received. If no frames follow, ASIC powers down the transceiver almost immediately after the frame reception; see min. overhead in Figure 17(c). In this case, a node with ASIC solution switches off the transceiver a few ms earlier than the software or ILA solution.
This work does not consider an ASIC solution in detail but only introduces it as the optimal solution—neglecting abovementioned open issues—for comparison reasons.

4.4. Evaluation. To determine idle listening caused by LETED, this evaluation uses the energy consumption model introduced in Section 6. Besides, the evaluation uses also the hardware and scenario parameters, like energy consumption from that section. For example, in this scenario an event occurs once an hour, which determines the number of active slots.

Figure 19 depicts the results of three solutions—software, ILA, and ASIC—applied to LETED for various end-to-end delays. Clearly, the shorter the guaranteed end-to-end delay is, the more receive slots are needed and the longer the total idle listening time is. For example, for an end-to-end delay of 5 seconds, the software approach causes 163 seconds of idle listening a day. In this case, the ASIC solution decreases idle listening 18x (9 sec) and ILA 15x (11 sec). Obviously, as both ASIC and ILA shorten passive slots, they reduce idle listening in this way.

Figure 20 presents the corresponding energy gain of ILA and ASIC against the software solution due to idle listening reduction. For example, nodes with ILA consume approximately. 0.9 mAh/day less than the software solution for 5-second delay. According to the lifetime evaluation presented in Section 6, nodes w/o ILA (DMAC approach) consume approximately. 2.5 mAh/day for 5-sec delay. Therefore, the energy gain of 0.9 mAh/day prolongs the lifetime by about 37%. For shorter delays, ILA achieves even better results.

As expected, ASIC solution reduces idle listening more than ILA, since it has shorter detection times. However, it results only in a minor difference in energy gain. For example, for end-to-end delay of 5-sec ASIC, energy gain is larger by only 0.008 mAh/day than ILA gain, which is less than 1% of the total energy consumption.

5. Simulative Evaluation

LETED with DLDC-MAC was implemented as a cross-platform software and tested with OMNeT++ [32] simulator. This section introduces the simulation environment and discusses the results.

5.1. Simulation Environment. All simulations introduced in this paragraph were carried out with OMNeT++, a discrete event simulator, which gained popularity in the last several years. In general, OMNeT++ consists of modules written in C++. Owing to the modular design, OMNeT++ can be easily extended with new simulation models and features. For example, mobility framework (MF) provides several models for mobile wireless communication. For instance, it simulates wireless channel at the physical level by considering signal strength, noise level, and so forth. In this way it determines whether a data packet will be processed or is treated as noise. This evaluation applies MF to simulate the wireless channel. Besides, MF supports moving hosts as well. However, this evaluation considers a static scenario; that is, sensor nodes do not move. Another extension—INET—provides protocol models for TCP, IPv4, IPv6, Ethernet, IEEE 802.11, OSPFv4, and other protocols. Thus, OMNeT++ allows also simulations of complex heterogeneous networks, for example, a sensor network connected with gateways to a local area network (LAN) or to the Internet.

This work utilizes another extension, introduced in [33], which integrates Reflex [34] operating system (OS) with OMNeT++. Owing to this extension all applications

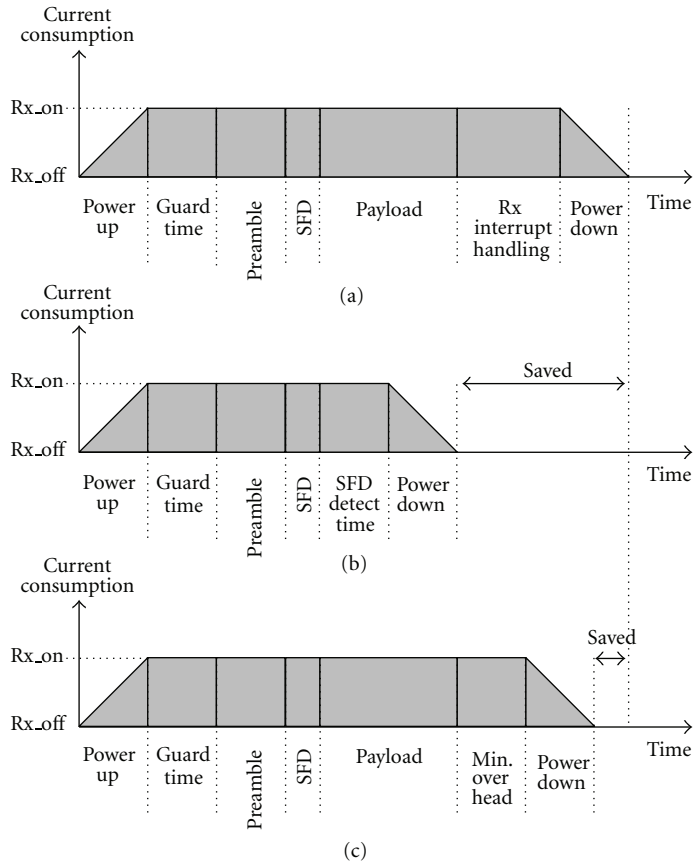


Figure 17: (a) General receive slot (software only). (b) Shortened passive slot with ILA and ASIC. (c) Active slots with reduced idle listening (ASIC only).

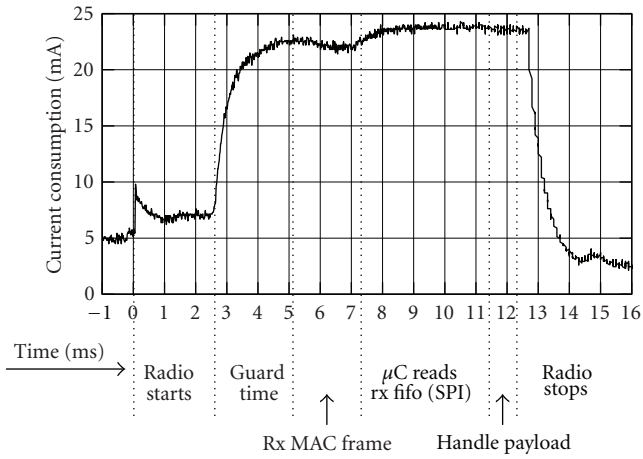


Figure 18: Rx slot of Tmote Sky sensor node (oscilloscope output: average from 2 samples); MAC frame 62 bytes data rate 250 kbps.

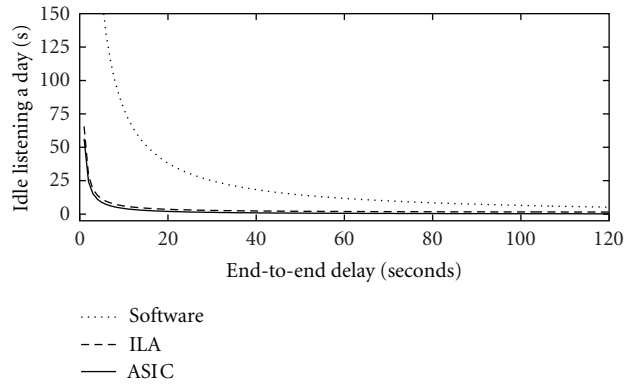


Figure 19: Idle listening causes by LETED of three various solutions: software, based on Tmote Sky (ILA) and with a dedicated hardware (ASIC).

implemented for Reflex OS run in OMNeT++ simulator and on platforms provided by Reflex, for example, Tmote Sky or Mica2. In this way developers can test their applications with OMNeT++ before deploying them on sensor nodes. However, LETED and DLDC-MAC implementation goes even one step further. It runs not only in OMNeT++ and on various hardware platforms, thanks to the Reflex OS extension, but also on any other OS, if they provide suitable adapters. Such a cross-platform design for sensor networks was introduced previously in [35].

The following paragraphs give an overview about major features of the simulation environment.

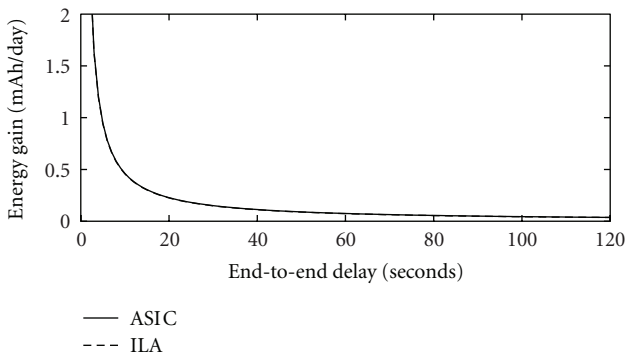


Figure 20: Energy gain of solutions to idle listening avoidance based on hardware: ILA (with CC2420 transceiver) and ASIC (based on dedicated hardware); as both solutions achieve similar results, the difference between them can hardly be spotted in this figure.

5.1.1. Integration with Reflex OS. Reflex OS is integrated with OMNeT++ with a *coroutine-based* model; that is, the module code runs in its own thread and usually consists of an infinite loop with send and receive calls. In general, each time an event associated with the Reflex module needs handling, like a message reception or a timer, the simulation kernel triggers the module to handle the event. From the Reflex OS perspective, each event is an interrupt. Therefore, each time the simulator triggers Reflex, an interrupt service routine is executed.

To simulate the system clock, OMNeT++ triggers the Reflex module every *tick* period. The tick value is set to a millisecond by default. Thus, OMNeT++ raises a system timer interrupt, every 1 ms on each node separately. Although such a practice allows exact simulations at a low operating system level, it results in a significant processing overhead. Therefore, OMNeT++ simulations take quite a long time, especially when simulating networks of many sensor nodes. To overcome this drawback the Reflex module was slightly adapted. Instead of simulating each clock tick, OMNeT++ triggers the system timer interrupt, only when the timer was set previously by applications. For example, should the MAC layer set the timer to fire in 30 seconds to send a beacon, OMNeT++ raises the timer interrupt after this time and not on every clock tick as previously. It reduces the processing overhead and allows running long-term simulations with many sensor nodes in a reasonable time.

5.1.2. Bit Error Simulation. In general, mobility framework (MF) applies the following steps to decide whether a frame was correctly received. First, it estimates the received power P_{rx} according to the Friis free-space equation [36]:

$$P_{\text{rx}} = \frac{P_{\text{tx}} \cdot \lambda^2}{16 \cdot \Pi^2 \cdot r^{\alpha}}, \tag{11}$$

where P_{tx} is the transmission power, λ the wavelength, r the distance between the transmitter and the receiver, and α the path loss coefficient with typically $\alpha \geq 2$. The coefficient α equals 2 for free-space path loss and 5 to 6 for shadowed areas or indoor scenarios [36]. Then, it estimates the signal to interference and noise ratio (SINR): it considers the constant thermal noise parameter, defined in a configuration file, and the noise caused by consecutive transmissions of other nodes. In this way MF discards frames upon collision, as consecutive frame transmissions result in noise levels higher than the frame reception power. Based on the SINR, the simulator estimates the frame bit error rate (BER) according to the modulation used. For instance, the BER of binary phase-shift keying (BPSK) equals

$$\begin{aligned} \text{BER} &= \frac{e^{S}}{2}, \\ S &= \frac{-\text{SNIR} \cdot \text{bandwidth}}{\text{bitrate}}. \end{aligned} \tag{12}$$

Next, MF estimates the probability P_{ok} that the received frame was not corrupted:

$$P_{\text{ok}} = (1 - \text{BER})^{l}, \tag{13}$$

where l is the frame length. Finally, MF gets a random value in the range from 0 to 1 and discards the frame, if the value was higher than P_{ok}.

5.1.3. Clock Drift. OMNeT++ does not consider clock drift by default. The simulator provides only the current simulation time t_{sim}. Therefore, to test the drift impact on LETED and DLDC-MAC, the OMNeT++ was extended to change node's local time according to clock drift. In short, before starting a simulation, OMNeT++ reads a configuration file and sets the drift parameter δ to each node separately. Then, each time nodes read the system time, the simulator calculates the local time t_{local} according to its drift parameter:

$$t_{\text{local}} = \frac{t_{\text{sim}}}{1 - \delta \cdot 10^{-6}}. \tag{14}$$

Clearly, if nodes have different drift parameters, their clocks run at different speeds and cause overlap risks of DLDC-MAC beacons and LETED slots.

5.2. Network Setup. The protocol stack of evaluated nodes included the application, network and data-link layer. In this scenario nodes applied AODV [29] routing protocol. In the data-link layer, nodes used LETED and DLDC-MAC. All layers were implemented as a cross-platform application in ANSI C. Thanks to the adaptation layer, the software was tested and evaluated with OMNeT++.

In this scenario the application was running only on the sources and on the sink. In the first case, the system timer triggered the application to send data to the sink once an hour. Each timer trigger corresponds to an event detected by the source. However, to examine various delays from an event detection to the first tx slot, the source nodes added a random time, within the margin of beacon period, to the next trigger time. Each frame transmitted contained the current simulation time used on the sink to estimate the total delay of multihop communication.

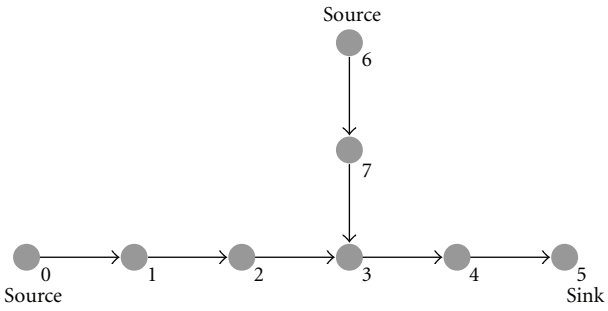


Figure 21: Two sources—nodes 0 and 6—send data periodically to the sink (node 5).

The evaluated LETED implementation applied two new approaches to the overlap problem. First, it used extra synchronization frames to keep LETED slots arranged along the path (details in Section 3). Second, it allowed LETED slots to overlap but reduced the risk of frame loss in overlap case by applying the ARQ protocol (see Section 3).

As stated above, OMNeT++ simulates frame loss according to the path loss model. Table 2 presents the parameters applied to the simulations.

This evaluation considers the physical layer (PHY) of IEEE 802.15.4 standard with 2.4 GHz carrier frequency. Thus, the characteristics of this PHY and of the corresponding transceiver, Chipcon CC2420 [31], were configured at the beginning. However, the configured binary phase-shift keying (BPSK) modulation is used in the 868 MHz and 915 MHz bands only. The 2.4 GHz band uses offset quadrature-phase-shift Keying (OQPSK) modulation. As OQPSK was not available, BPSK was selected.

Two different networks—small and large—were evaluated with OMNeT++ simulator. Both setups are introduced in the following.

Figure 21 presents the small network evaluated with OMNeT++ simulator. In general, nodes 0 and 6 served as sources; they sent data to the sink, to node 5. Since nodes 3,4, and 5 were on two gathering paths, that is, from sources 0 and 6, they set up two schedules, for each path separately. In this scenario the offset between rx and tx slots on intermediate nodes was set to 100 ms. It resulted in approximately. 100 ms forwarding delay of a single hop. Thus, with 5-hop path to the sink, the nodes had to wake up every 9.5 seconds to guarantee 10-second delays (see Section 3.2).

The thermal noise parameter was adapted to get a PER of approximately. 10%, which is higher than common PER observed in sensor networks. The reason for a higher PER is to test whether LETED works well in worse conditions than expected. However, with such a PER the LETED protocol did not work correctly without ARQ protocol. The problem stems from the way the slot synchronization works. In short, in multihop networks, synchronization frames do not reach the sink due to bit errors. As a result, nodes close to the sink are not synchronized and miss frames transmitted in LETED slots. Therefore, the evaluation considers only the protocol stack with ARQ applied. Three various simulations were performed with OMNeT++ differing in PER and ARQ parameters (see Table 3). Each scenario was simulated 3 months.

Table 2: Simulation parameters that affect the packet error rate (PER).

Parameter name	Value
Carrier frequency	2.4 GHz
Bitrate	250 kbps
Channel bandwidth	2 MHz
Transceiver Tx power	1 mW
Transceiver sensitivity	−94 dBm
Thermal noise	−84.5 dBm
Modulation	BPSK
Path loss coefficient α	3

Table 3: Packet error rate (PER) and ARQ parameters of three small-network simulations performed with OMNeT++.

Acronym	PER parameter	max. ARQ retries
S1	0%	1
S2	Approx. 10%	1
S3	Approx. 10%	2

The small network introduced previously considers a high PER, but the number of nodes is small. In such a setup, beacons and slots overlap rarely and nodes do not handle it as often as that in common scenarios. Besides, in larger networks there is a higher risk of sending frames concurrently, as DLDC-MAC avoids it only among neighbors. In such cases, nodes affect each other's transmissions and cause collisions. However, the previous scenario dose not suffer from this problem, as the number of nodes is small.

To evaluate LETED in more realistic conditions, another simulation was carried out. Figure 22 presents the network topology of 155 nodes deployed randomly in a square area of length 250 meters. The sink was placed in the middle and four sources at the corners. Sources set up routes and wake-up schedules to the sink (see Figure 22). As previously mentioned, nodes applied the AODV to find routes to the sink. They did not care about route metrics, like hop count, and selected the first available path.

In this scenario only 4 out of 155 nodes sent data to the sink, as LETED does not yet support efficiently communication from many sources. In the current version, each source sets up a separate wake-up schedule to the sink. Obviously, it causes frequent wake-up times in larger networks. An effective way to handle many sources is a part of future research.

Similar to the previous network, nodes support also 10-second end-to-end delays. To counter the frame loss problem, nodes apply the ARQ protocol with 1 retry.

5.3. PER Results. This paragraph presents the average packet error rate (PER) in all simulations. As stated above, OMNeT++ calculates the PER from several parameters, for example, the tx power or distance between nodes. Nodes with DLDC-MAC do not apply any solutions to unreliable

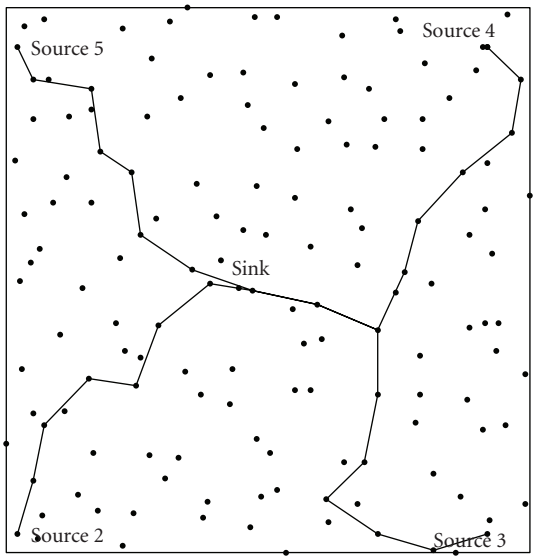


Figure 22: Evaluated network of 155 nodes. Four sources send data to the sink along the routes depicted in the figure.

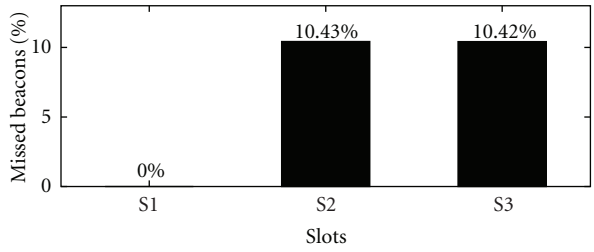


Figure 23: Small network: the total number of missed beacons among all nodes in three scenarios; as nodes did not apply ARQ for beacon transmission, it shows the average PER of links.

communication, for instance, CSMA/CA or ARQ. Therefore, the average PER is equal to the number of missed beacons. However, such a PER does not include LETED frames missed due to slot overlap.

Figure 23 presents PER values of three scenarios in the small network. As expected, the nodes did not suffer from the frame loss risk in the first scenario with a PER of 0%. In two other scenarios, the PER was close to 10%.

Figure 24 shows the PER results of the large network for each route separately. It is the average PER of all neighbors and not the PER of the previous or next node to the sink. As nodes were deployed randomly, distance among them varied and resulted in different rx power and PER, that is, from 0% to 3.35%.

5.4. Small Network Evaluation. To estimate end-to-end delays, source nodes included in frames event and transmission times. Each time the sink received a frame, it captured the reception time. Then, it calculated the time passed from the event detection.

During all simulations, sources had to deliver event notices to the sink within 10 seconds. Figure 25 shows the results of the small-network experiments: the sink received more than 99% frames within this time. Only less than 0.5% frames reached the sink too late. There are two reasons for frames reaching the sink too late. First, if there is an overlap risk, nodes skipped affected slots. Should a node skip a tx slot and have awaiting frames, it sends them in the next slot, that is, in approximately. 10 seconds in this case. The second reason for frames reaching the sink too late are bit errors. In this case, nodes apply the ARQ protocol and send frames again, if they do not receive ACK. It affects end-to-end delays only if nodes send retries in the next tx slot. However, in this case, nodes send retries in the same slot and ARQ did not influence end-to-end delays. Therefore, the main reason for receiving frames too late are skipped slots. This observation explains the results presented in Figure 25. That is, in all simulations a similar number of frames achieved the sink on time, although they differ in the PER values and ARQ parameters.

Figure 25 includes only frames received by the sink. However, here the success rate means the number of event notices received on time. Clearly, missed frames reduce the success rate and must be considered as well. Figure 26 presents the number of missed frames in all simulations. To obtain the success rate, the number of frames on time (see Figure 25) must be reduced by missed frames. In S1 and S3 scenarios it affects slightly the success rate. For example, in S3 the success rate is still higher than 99%, as the sink missed less than 0.5% frames.

Although frames were not affected by bit errors in S1, the sink missed 0.25% packets (see Figure 26). As stated above, the sink missed some frames, as nodes skipped slots to reduce the overlap risk. This shows the performance of the ARQ-based solution to the overlap problem: it resulted in 0.25% frame loss.

As expected, in the S3 scenario the sink missed more frames than that in S1, that is, 0.46% and 0.25%, respectively, for source 0. However, the number of missed frames from node 7 is equal in both scenarios. Theoretically, the sink should miss more frames in S3 due to a higher bit error rate. This phenomenon can be explained as follows. More retries in S3 resulted in longer slots, which could be used partly instead of skipped like that in S1. Figure 29 shows that nodes in S3 skipped fewer slots than in S1 and also used more slots partly. In addition, more ARQ retries in S3 recovered from some bit errors. Thus, the number of missed frames is equal in both runs.

Figure 29 shows the average number of slots skipped, partly used, and joined, in the small network scenario. These numbers depend on the total slot count and their length. The first one stems from end-to-end delays nodes have to support. In this scenario it was 10 seconds. The slot length depends mainly on the frame length and the number of ARQ retries. Since in S1 and S2 nodes applied 1 ARQ retry, it resulted in a similar amount of slots skipped, partly used, and joined. However, S3 applied more retries and resulted in longer slots. In this case, nodes used such long slots partly more often than those in previous runs, when they were skipped. Therefore, the number of partly used slots in S3 is higher than that in S1 and S2, 0.34% versus 0.20%. Clearly, if

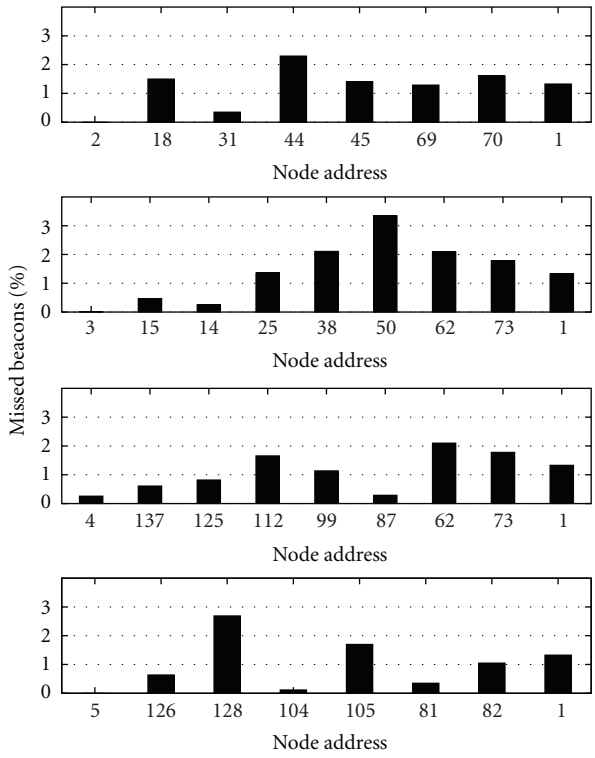


Figure 24: Large network: the average number of missed beacons (corresponds to the PER of links) for each route separately.

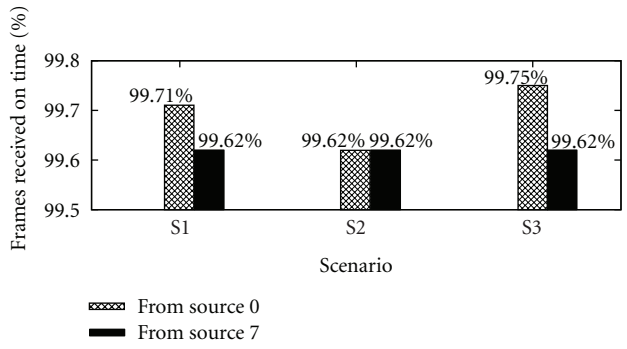


Figure 25: Small network: the number of frames (event notices) the sink received on time; it does not include missed frames.

more slots were used partly in S3 than those in the previous runs, less slots were skipped: 0.14% and 0.20% skipped slots, respectively. Besides, with longer slots there was less free space between them, causing slots to join more often: 0.88% slots in S3 versus 0.72% in S1 and S2.

5.5. Large Network Evaluation. Although nodes in the large network (LN) scenario suffered from a smaller PER than those in the small network (SN), the sink received fewer frames on time. For example, in SN more than 99% frames reached the sink on time. In LN, however, the sink got fewer than 99% frames on time from all routes, that is, within 10 seconds and less (see Figure 27). In the worst case, it

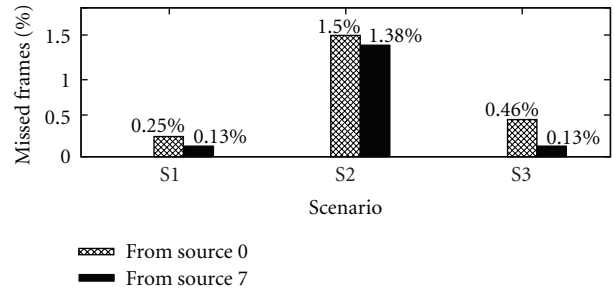


Figure 26: Small network: the total number of event notices that the sink did not receive, that is, end-to-end packets, for each source separately.

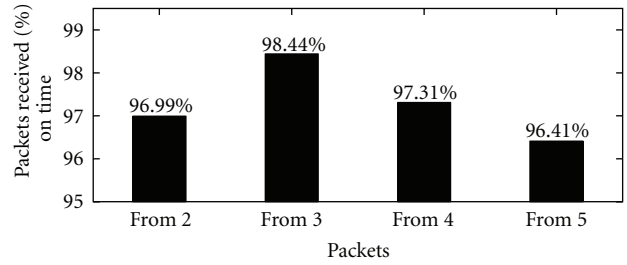


Figure 27: Large network: the amount of frame the sink received on time, that is, within 10 seconds.

received 96% frames on time. Clearly, the main reason for worse PER results of LN is the higher overlap risk due to more neighbors than that in SN. Besides, as nodes applied only 1 ARQ retry, it resulted in short LETED slots. Should beacons and LETED slots overlap, nodes skipped the latter ones and did not used them partly (details in Section 3). In this case, nodes send awaiting frames in the next slot, that is, after about 10 seconds, and the sink received it too late. In the SN scenario, nodes skipped about 0.20% slots due to overlap (see Figure 29). However, in LN nodes skipped more slots than those in SN, that is, from 0.45 to 0.65%.

Short LETED slots and a higher overlap risk impact also the total number of missed events. In the worst case, the sink missed 1.53% frames (see Figure 28), which is similar to the number of events missed in SN with 1 retry. However, in the latter case nodes suffered from a higher PER, that is, 10% instead of 1-2%. As in LN nodes missed more slots due to overlap than is SN, it caused a higher frame loss rate. For instance, nodes in LN missed even 5x more frames due to overlap than in SN, that is, 0.72% (see source 4 in Figure 30) and 0.13% (see Figure 29), respectively.

To achieve a better performance, nodes might apply the following solution. Should nodes use only a few ARQ retries, they prolong LETED slots by sending ARQ retries later and not consecutively. In the overlap case, nodes can use such slots partly and prevent sending frames too late or discarding them.

Nonetheless, LETED achieved good results in the LN scenario even without the improvement mentioned above:

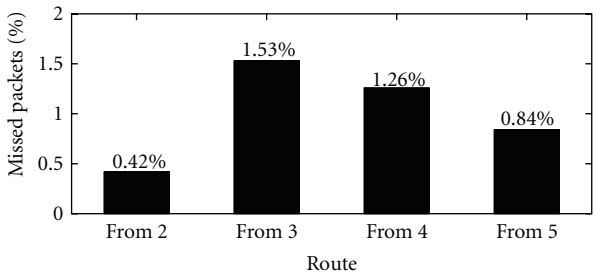


Figure 28: Large network: missed event notices.

the success rate of all routes is more than 95%; that is, the total number of frames received on time and not missed.

6. Lifetime Evaluation

This chapter evaluates the lifetime of various approaches that support short end-to-end delays. It compares the solution presented in this work, that is, LETED coupled with DLDC-MAC, with the following protocols:

(1) staggered schedule,

(2) cycled receiver/preamble sampling,

(3) schedule based (TDMA).

Although LETED applies the ARQ protocol to deal with unreliable wireless links, the model does not consider it. In this way, we wanted to provide a fair comparison of LETED with other protocols, which do not apply ARQ.

6.1. Overview. The following paragraphs introduce briefly solutions to end-to-end delays evaluated in this chapter.

6.1.1. Staggered Schedule. This chapter evaluates also the generic idea of staggered schedule separately from LETED, although the latter one applies such a schedule. By doing so, the evaluation finds energy savings of LETED against the generic staggered schedule approach.

Apart from a staggered schedule nodes, need an underlying MAC protocol, since such a schedule does not provide rendezvous with all neighbors. This evaluation considers a staggered schedule coupled with DLDC-MAC. Since LETED is based on DLDC-MAC as well, the evaluation provides an accurate estimation of the energy saved by LETED against a generic staggered schedule.

To estimate energy consumption and the lifetime of nodes with a staggered schedule, the evaluation considers the LETED energy consumption from Section 6.2 but without the idle listening avoidance improvements introduced in this work.

6.1.2. Preamble Sampling (Cycled Receiver). The following formulas provide the energy consumption model of preamble sampling approaches.

Total energy consumption of preamble sampling E_{preamble} consists of energy needed for sending data E_{tx} and the reception energy E_{rx}:

$$E_{\text{preamble}} = E_{\text{tx}} + E_{\text{rx}}. \tag{15}$$

Nodes with preamble sampling send data only when they detect an event. Thus, the total number of transmission slots N_{tx} depends on the event frequency T_{event}:

$$N_{\text{tx}} = \frac{T_{\text{day}}}{T_{\text{event}}}, \tag{16}$$

where T_{day} is "the amount of time units a day" T_{event} is expressed with, similar to LETED model. As stated above, nodes may send a series of short frames that imitate a long preamble. However, for the sake of simplicity, this model assumes that nodes send continuous preambles. Then, transmission energy is estimated as

$$E_{\text{tx}} = N_{\text{tx}} \cdot \left(T_{\text{sleep}} + t_{\text{frame}}\right) \cdot I_{\text{tx}} + E_{\text{startup}} + E_{\text{shutdown}}. \tag{17}$$

With the preamble sampling approach nodes periodically check the channel activity. The number such check operations a day N_{rx} is:

$$N_{\text{rx}} = \frac{T_{\text{day}}}{T_{\text{sleep}}}. \tag{18}$$

On average nodes receive a half of the preamble when getting data frames and the total reception energy equal

$$E_{\text{rx}} = \left[N_{\text{rx}} \cdot t_{\text{rx}} + N_{\text{tx}} \cdot \left(\frac{T_{\text{sleep}}}{2} + t_{\text{frame}}\right)\right] \cdot I_{\text{rx}}, \tag{19}$$

where t_{rx} is the sampling channel duration, that is, the time nodes need to discover whether other nodes send a long preamble before data frames. If nodes use bit streaming radios, like CC1000, they have a low-level access to individual bits while sending or receiving. In this case, nodes send long and continuous preambles and receivers use short times t_{rx} to detect channel activities. For example, low power listening introduced in B-MAC [9] needs approximately. 350 μs to detect a preamble. However, nodes with packetizing radios, for example, CC2420, cannot control the preamble length. Therefore, they imitate a long preamble by sending short wake-up frames in a sequence, like those in TICER. In this case, t_{rx} is longer than that in B-MAC. For instance, if wake-up frames are 30-byte long with the preamble included, nodes receive such frames in about 1 ms in the best case. However, t_{rx} is usually longer, as nodes need an extra time to get frames from the rx buffer. Experiments presented in Section 4 revealed that nodes need a few ms to read data from the rx buffer.

Preamble sampling causes idle listening on both senders, because of long preambles, and on receivers, due to periodic checks of the channel activity. Besides, should nodes detect a preamble on the channel, they remain in the receive state until the frame arrives and increase idle listening as well. Clearly, to prolong the lifetime nodes should reduce idle

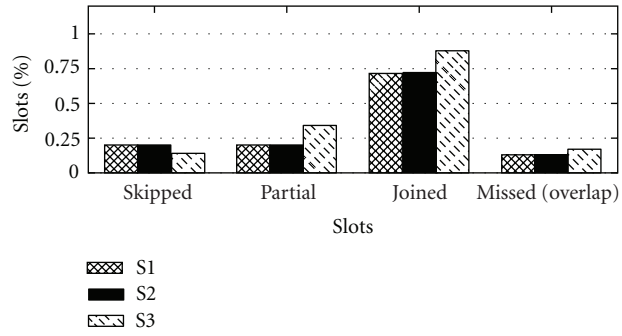


FIGURE 29: Small network: the number of slots skipped, used partially, and joined upon an slot overlap risk among all nodes, for each test run separately.

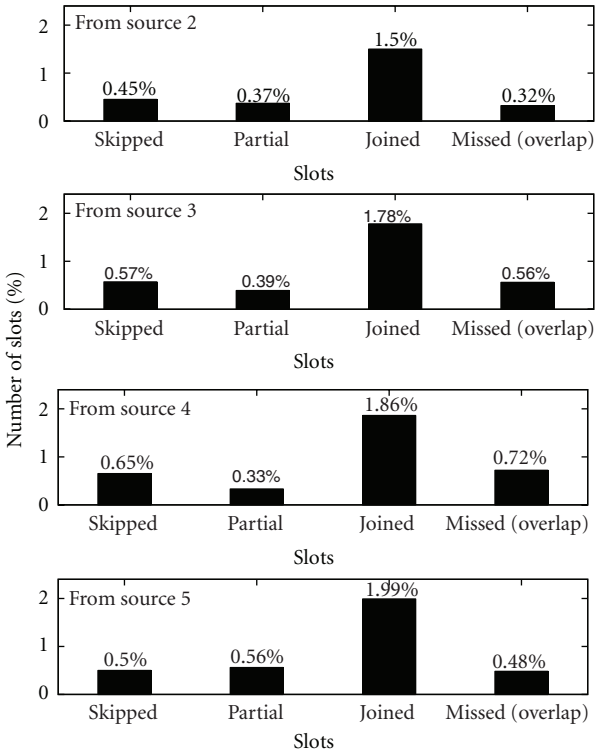
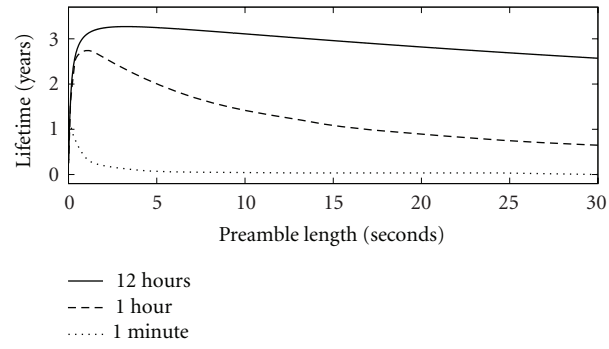
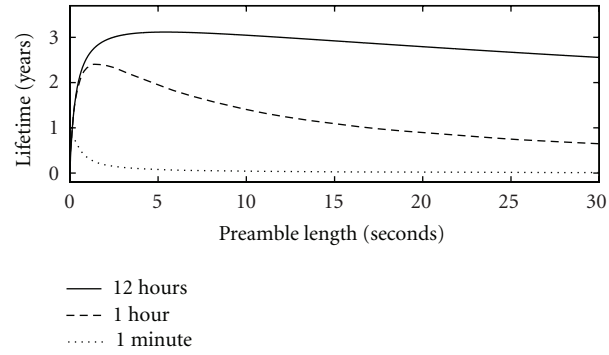


FIGURE 30: Large network: slots statistics for each route separately.



(a) B-MAC generates long preambles and receivers need approximately. 350 μs to detect it; B-MAC runs only on radios that support a low-level access to individual frame bits



(b) TICER emulates long preambles by sending consecutive beacon frames; nodes need almost a millisecond to receive a beacon and detect channel activity. Although TICER achieves worse results than B-MAC, it works with all transceivers

FIGURE 31: There is the optimal preamble length in solutions based on preamble sampling; the results show the lifetime for various duty cycle, that is, nodes send frames once a minute, once an hour and every 12 hours.

listening by adapting the preamble length. There is a tradeoff in preamble length: short preambles reduce idle listening of transmissions but increase it on receivers, as they check the channel more often. However, with frequent transmissions, short preambles may reduce idle listening on receivers, as they do not need to wait long for data frames after preamble detection.

To find the optimal preamble length, the formulas introduced previously were applied to the scenario from Section 6.3. In short, sources in a 5-hop network send data frames to the sink with a different frequency: once a minute, once an hour, and every 12 hours. Figure 31 presents the lifetime results of B-MAC and TICER protocols. By adapting the preamble length, nodes balance idle listening in send and receive states and find the optimal preamble. B-MAC achieves the longest lifetime with a preamble of 120 ms with an average tx period of a minute (see Figure 31 and Table 4). Should nodes send data more rarely, idle listening on senders becomes smaller. In this case, to balance idle listening in tx and rx states, node apply longer preambles. Thus, the optimal preamble length is longer in scenarios with lower duty cycles. For example, the optimal preamble with the average tx period of an hour is about 8x longer than in the scenario with 1-minute tx frequency.

TICER differs from B-MAC mainly in a longer time receivers need to check the channel. In this scenario, nodes with TICER expect "preamble beacons" of 30 bytes and therefore need about 960 ms to detect it. With a longer channel check time, TICER achieves worse results than B-MAC. For instance, with an average tx period of 1 hour nodes reduce the lifetime by 15%, if they apply TICER instead of B-MAC. However, the advantage of TICER is the fact it works with every transceiver, as it sends ordinary frames. On the contrary, B-MAC needs a low-level access to individual bits. Besides, the transceiver supporting B-MAC

Table 4: The optimal preamble length of B-MAC and TICER causes smallest idle listening and results in the longest lifetime, presented in brackets.

	Average tx time		
	1 minute	1 hour	12 hours
B-MAC	120 ms (1.1 years)	0.94 s (2.76 years)	3.27 s (3.27 years)
TICER	200 ms (0.76 years)	1.56 s (2.41 years)	5.42 s (3.12 years)

must allow transmission of any-length preambles, which nodes can freely adapt.

6.2. LETED Energy Model. This section introduces the energy consumption model of nodes using LETED coupled with DLDC-MAC. Table 5 lists the symbols used in the model. The parameters related to the scenario show Tables 6, 7, and 8.

6.2.1. Lifetime and Daily Energy Consumption. The lifetime of sensor nodes with LETED and DLDC-MAC is estimated as

$$\text{Lifetime} = \frac{Q}{E_{\text{day}}}, \tag{20}$$

where Q is the available battery capacity and E_{day} is the total energy consumption a day, that is, the sum of energy consumed by all activities:

$$E_{\text{day}} = E_{\text{LETED}} + E_{\text{LDC}} \\ E_{\text{mcu}} + E_{\text{selfdischarge}}, \tag{21}$$

where E_{LETED} and E_{LDC} are the energy consumed by LETED and DLDC-MAC, respectively, E_{mcu} energy consumed by the microcontroller in both active and sleep states, and $E_{\text{selfdischarge}}$ the self-discharge rate of batteries.

6.2.2. LETED Energy Consumption. LETED consumes energy when sending and receiving data in active slots and while listening for potential transmissions in passive slots. For the sake of simplicity, listening in passive slots is just keeping the transceiver in the reception state. Therefore, here LETED consumes energy either while sending frames $E_{\text{tx\_slots}}$ or when keeping the radio in the rx state $E_{\text{rx\_slots}}$:

$$E_{\text{LETED}} = E_{\text{tx\_slots}} + E_{\text{rx\_slots}} \tag{22}$$

The number of active slots N_{active} depends on the average event period T_{event}:

$$N_{\text{active}} = \frac{T_{\text{day}}}{T_{\text{event}}}. \tag{23}$$

The number of passive slots depends on end-to-end delays d_{EtE} the network supports. In other words, each node wakes up every T_{slot} period to receive and send potential data:

$$T_{\text{slot}} = d_{\text{EtE}} - n \cdot (t_{\text{frame}} + t_{\text{tx\_offset}}), \tag{24}$$

where t_{frame} is the expected frame length and $t_{\text{tx\_offset}}$ the gap between receiving a frame and sending it to the next node. Then, the total number of passive slots a day N_{passive} is

$$N_{\text{passive}} = \frac{T_{\text{day}}}{T_{\text{slot}}} - N_{\text{active}}. \tag{25}$$

As nodes send data during active slots only, the total tx energy is estimated as

$$E_{\text{tx\_slots}} = N_{\text{active}} \cdot \left(t_{\text{frame}} \cdot I_{tx} + E_{\text{startup}} + E_{\text{shutdown}}\right), \tag{26}$$

where I_{tx} is the transceiver current consumption while sending data, E_{startup} and E_{shutdown} energy consumed to power up and down the transceiver. The expected frame length t_{frame} in time units is calculated as follows:

$$t_{\text{frame}} = \frac{\lambda_{\text{frame}} + \lambda_{\text{preamble}} + \lambda_{\text{SFD}}}{\vartheta}, \tag{27}$$

where λ_{frame} is the expected data length and ϑ is the transceiver data rate. t_{frame} includes also the preamble $\lambda_{\text{preamble}}$ and the start of frame delimiter λ_{SFD}.

The single reception time during active and passive slots, $t_{\text{rx\_active}}$ and $t_{\text{rx\_passive}}$, is necessary to estimate energy consumption of frame reception:

$$t_{\text{rx\_active}} = t_{\text{guard}} + t_{\text{frame}} + t_{\text{rx\_post}}, \tag{28}$$

where t_{guard} is the guard time, t_{frame} the average frame length and $t_{\text{rx\_post}}$ the extra time needed to detect that no frames follow the current one. Nodes apply the moving average drift compensation (MADC) to deal with clock drift and estimate t_{guard} in the same way as DLDC-MAC does (see (38)).

By applying the idle listening avoidance (ILA) solution, introduced in Section 4, nodes shorten passive slots to:

$$t_{\text{rx\_passive}} = t_{\text{guard}} + t_{\text{preamble}} + t_{\text{SFD}}, \tag{29}$$

where t_{SFD} is the time needed to detect the start frame delimiter of incoming frames and t_{preamble} the preamble reception time together with the SFD field. t_{preamble} is estimated similarly to t_{frame} (see (27)).

As stated before, nodes with LETED consume energy $E_{\text{rx\_slots}}$, while receiving in both active and passive slots:

$$E_{\text{rx\_slots}} = N_{\text{active}} \cdot (t_{\text{rx\_active}} \cdot I_{\text{rx}}) \\ + N_{\text{passive}} \cdot \left(t_{\text{rx\_passive}} \cdot I_{\text{rx}}\right) \\ + \left(N_{\text{active}} + N_{\text{passive}}\right) \cdot \left(E_{\text{startup}} + E_{\text{shutdown}}\right), \tag{30}$$

$$E_{\text{rx\_slots}} = I_{\text{rx}} \cdot \left(N_{\text{active}} t_{\text{rx\_active}} + N_{\text{passive}} t_{\text{passive}}\right) \\ + \left(N_{\text{active}} + N_{\text{passive}}\right) \cdot \left(E_{\text{startup}} + E_{\text{shutdown}}\right), \tag{31}$$

where I_{rx} is the current drawn the receive state.

6.2.3. Energy Consumption of DLDC-MAC. DLDC-MAC protocol consumes energy when sending E_{txbeacon} and receiving E_{rxbeacon} beacons:

$$E_{\text{LDC}} = E_{\text{txbeacon}} + E_{\text{rxbeacon}}. \tag{32}$$

Table 5: Symbols used in the model.

Symbol	Description
E_{day}	Daily energy consumption
E_{LETED}	LETED energy consumption
E_{LDC}	Energy consumption of the underlying low-duty cycle protocol
E_{txbeacon}, E_{rxbeacon}	Energy consumed to send and to receive beacons during a day
$E_{\text{tx\_slots}}$, $E_{\text{rx\_slots}}$	Energy consumed in a day for sending and receiving data
E_{mcu}	Daily energy consumption of μC
$E_{\text{mcuactive}}$	Daily energy consumption of μC in active mode when radio is powered down
E_{sleep}	Daily energy consumption when the node sleeps
$t_{\text{rx\_active}}$, $t_{\text{rx\_passive}}$	Length of rx active/passive slot
t_{txbeacon}	Transmission time of a single beacon
t_{rxbeacon}	Average reception time of a single beacon
$t_{\text{rxbeacon after}}$	Listening time after sending a beacon
t_{guard}	Guard time for clock drift compensation
t_{frame}	Transmission time of single data frame
t_{preamble}	Time to send or receive the preamble with the start of frame delimiter field
T_{sleep}	Total sleep time in a day
T_{slot}	LETED slot period needed to support certain end-to-end delays
T_{day}	The number of time units (e.g., seconds) a day that the beacon period T_{beacon} is expressed (e.g., T_{day} equals 86 400 seconds a day, when T_{beacon} is expressed in seconds)
N_{active}, N_{passive}	Number of active/passive slots a day (LETED solution)
B	The number of beacons a node sends during a day

Table 6: Scenario parameters.

Parameter	Description	Value
λ_{frame}	Data frame length	128 bytes
T_{event}	How often events occur	Various
n	Hop count: source to sink	2, 5, and 10
d_{EtE}	Maximum end-to-end delay	Various
$T_{\text{mcuactive}}$	How long μC is active a day when radio is powered down	10 minutes

Table 7: LETED and DLDC-MAC parameters.

Parameter	Description	Value
$t_{\text{tx\_offset}}$	The time a tx slot follows the corresponding to rx slot in LETED	50 ms
$t_{\text{rx\_post}}$	The time to get a frame in the application layer after it was received by the transceiver	4.5 ms
t_{SFD}	SFD detection time	100 μs
T_{sync}	The period of sending SYNC frames to align wake-up schedule along the path	5 minutes
	Parameters of the underlying DLDC-MAC protocol	
T_{beacon}	Beacon period	120 secs
λ_{beacon}	Beacon length	128 bytes
$\lambda_{\text{beacon\_after}}$	How long (bytes) the node waits in listening after sending a beacon	128 bytes
nbours	The number of neighbors	4
MBR	The average missed beacon rate	1%

Table 8: Hardware parameters of the energy consumption model together with values used for evaluation.

Parameter	Description	Value
Q	Available energy	1800 mAh
$I_{\text{mcuactive}}$	Current consumption when μC is active	2 mA
I_{tx}	Current consumption when sending and receiving	20 mA
I_{rx}		22 mA
I_{sleep}	Current consumption when the node sleeps	0.01 mA
ϑ	Transceiver data rate	250 kbps
$E_{\text{self discharge}}$	Daily self-discharge rate of batteries	0.74 mAh
E_{startup}	Energy needed to power the transceiver up and to power it down	7.2 nAh
$E_{\text{shut down}}$		4.2 nAh
$E_{\text{tx rx switch}}$	Energy needed to change the transceiver mode from sending to receiving	4 nAh
δ	Relative clock drift between two nodes when MADC (moving average drift compensation) is applied	2.18 ppm
$\lambda_{\text{preamble}}$	Preamble length	4 bytes
λ_{SFD}	The length of SFD (start of frame delimiter) field	1 byte

Thus, to estimate the energy consumption of DLDC-MAC, the total number of beacons a day B must be calculated:

$$B = \frac{T_{\text{day}}}{T_{\text{beacon}}}, \tag{33}$$

where T_{beacon} is the beacon period, that is, the time between two successive beacons.

Beacon Transmission. Nodes with DLDC-MAC send beacons periodically every T_{beacon} time. After sending a beacon, nodes stay $t_{\text{rxbeaconafter}}$ time in the receive state to get potential network control frames, like network join request. Clearly, as nodes switch from the tx state to the reception, they consume additional energy $E_{\text{txrxswitch}}$. The total energy consumed for sending beacons equals

$$\begin{aligned} E_{\text{txbeacon}} = B \cdot (&t_{\text{txbeacon}} \cdot I_{\text{tx}} + t_{\text{rxbeaconafter}} \cdot I_{\text{rx}} \\ &+ E_{\text{txrxswitch}} + E_{\text{startup}} + E_{\text{shutdown}}\Big). \end{aligned} \tag{34}$$

Both parameters beacon length λ_{beacon} and the listening time after beacons $\lambda_{\text{beacon\_after}}$ are expressed in bytes. The following formulas express them in time unit, according to the transceiver data rate ϑ:

$$\begin{aligned} t_{\text{txbeacon}} &= \frac{\lambda_{\text{beacon}}}{\vartheta}, \\ t_{\text{rxbeaconafter}} &= \frac{\lambda_{\text{beacon\_after}}}{\vartheta}. \end{aligned} \tag{35}$$

Beacon Reception. Nodes with DLDC-MAC receive beacons from all neighbors and thus consume energy E_{rxbeacon} for beacon reception:

$$E_{\text{rxbeacon}} = B \cdot \text{nbours} \cdot \Big(t_{\text{rxbeacon}} \cdot I_{\text{rx}} + E_{\text{startup}} + E_{\text{shutdown}}\Big), \tag{36}$$

where nbours in the neighbors count and $t_{\text{rxbeacon}}$is the time needed to receiving a single beacon. As nodes compensate clock drift, they apply guard times t_{guard} for each beacon. Therefore, the time to receive a single beacon equals

$$t_{\text{rxbeacon}} = t_{\text{guard}} + t_{\text{txbeacon}}. \tag{37}$$

Obviously, guard times depend on the beacon period T_{beacon}, that is, on the last time when nodes synchronized their times by receiving beacons. Besides, if nodes miss a beacon, they double the guard time for the next reception try. The average guard time t_{guard} over a beacon period T_{beacon} is estimated as

$$t_{\text{guard}} = \frac{\delta \cdot T_{\text{beacon}}}{1 - \text{MBR}}, \tag{38}$$

where δ is relative drift among neighbors and MBR is the average missed beacon rate. In this scenario nodes apply the solution to guard times based on drift prediction introduced in our previous work [26].

6.2.4. Microcontroller Energy Consumption. Although microcontrollers (μC) use several power consumption states, this evaluation considers only two: active $E_{\text{mcuactive}}$ (executing code, reading sensors, sending, receiving, etc.) and sleep E_{sleep} (only a low rate clock is running). The total energy the microcontroller consumes a day E_{mcu} equals

$$\begin{aligned} E_{\text{mcu}} &= E_{\text{mcuactive}} + E_{\text{sleep}} \\ E_{\text{mcuactive}} &= T_{\text{mcuactive}} \cdot I_{\text{mcuactive}}, \end{aligned} \tag{39}$$

where $T_{\text{mcuactive}}$ is the time the μC is active a day and $I_{\text{mcuactive}}$ is the μC current consumption in the active state. Similarly, the energy in the sleep state is estimated as

$$E_{\text{sleep}} = T_{\text{sleep}} \cdot I_{\text{sleep}}, \tag{40}$$

where T_{sleep} is the total μC sleep time a day and I_{sleep} is the current consumption in the sleep state. In other words, T_{sleep} is the period when sensor nodes do not perform any task, that is, they sleep to save energy. Clearly, T_{sleep} is calculated as a complement of other activities, that is, sending and receiving data, listening for potential transmissions, and code execution.

6.3. Scenario. This evaluation considers sensor nodes that monitor specific events, for example, moving object or gas leakage. After source nodes detect an event, they send notice frames to the sink. Tables 6, 7, and 8 present the scenario parameters, introduced in the following.

As end-to-end delay depends on the hop distance from the source, especially for approaches not based on the staggered schedule, three different scenarios are considered: with 2, 5, and 10 hops to the sink. Besides, the lifetime depends on end-to-end delay that nodes must support, since it impacts the duty cycle. Thus, the evaluation considers various end-to-end delays, starting from less than a second.

In this scenario nodes send small frames on event detection, that is, only 128 bytes. However, it is big enough to include event notice, like gas detection at a specific place. The frequency of events impacts the number of transmissions as well. The more the events sources detect, the more frames they send. This evaluation considers various event periods, that is, from 1 minute to 12 hours.

On frames reception nodes with LETED wait the time $t_{\text{tx\_offset}}$ before sending it to the next node. The smallest $t_{\text{tx\_offset}}$ must compensate drift arisen between the sender and the receiver over the sleep period. Therefore, $t_{\text{tx\_offset}}$ should be long enough to counter the drift problem. However, $t_{\text{tx\_offset}}$ impacts end-to-end delay and duty cycle; see (3.2), and therefore should be short enough. In this scenario nodes apply $t_{\text{tx\_offset}}$ of 50 ms.

To cope with errors in multihop drift estimation, nodes with LETED synchronize wake-up schedules each time they receive data frames. To synchronize schedules in long idle periods, nodes send extra SYNC frames. The SYNC period should be short enough to recover from multihop drift changes. In this scenario, nodes send SYNC frames every 5 minutes.

Table 7 presents DLDC-MAC parameters. Although the beacon period T_{beacon} equals 2 minutes, it does not impact end-to-end delay, since nodes use LETED slots to send event notices. As nodes with DLDC-MAC receive beacons from all neighbors, the neighbor number impacts duty cycle and the lifetime. In this scenario nodes have 4 neighbors on average. On missing a beacon nodes apply a double-length guard time to receive the following beacon. It results in longer idle listening and shortens the lifetime. In this scenario the average missed beacon rate is as high as 1%; that is, the bit error rate (BER) equals approximately. 10^{-6}.

Although nodes miss beacons and data frames in this scenario, they do not apply the ARQ approach that handles the frame loss problem, as already mentioned.

Finally, Table 8 lists the hardware parameters of the sensor platform. In this evaluation an off-the-shelf sensor node—Tmote Sky—was considered. Thus, the parameters come from Tmote Sky datasheet [1] and from measurements. The values of current and energy consumption are specified in mA and mAh units, and not in W and Ws, as hardware datasheets usually provide the former ones.

Tmote Sky nodes use two AA batteries as the energy source. In this scenario they have 2x rechargeable batteries Sanyo eneloop each with the overall capacity of 2000 mAh. However, Tmote Sky uses approximately. 80% of the available energy, as these batteries provide 80% of capacity with required 1.2 V voltage. Besides, the scenario considers also the self-discharge of the batteries. In general, Sanyo eneloop batteries lose approximately 15% of the capacity in a year, resulting in a day loss of about 0.74 mAh.

To reduce idle listening stemming from guard times, nodes apply the drift prediction solution based on moving average (MADC), introduced in our work [26]. According to the experiment results, relative drift among two nodes decreases to approximately. 2.18 ppm after applying MADC.

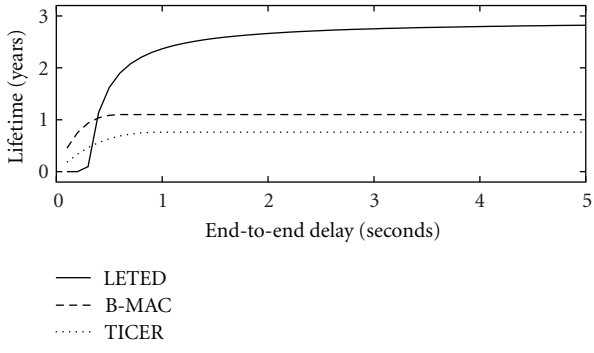
6.4. LETED and Preamble Sampling. This paragraph compares LETED against B-MAC and TICER, which are state-of-the-art MAC protocols for sensor networks. As previously mentioned, both B-MAC and TICER apply the preamble sampling approach. However, only B-MAC sends long continuous preambles and benefits from a short time needed to detect the channel activity. On the contrary, TICER emulates long preambles by sending successive wake-up beacons.

This evaluation considers three scenarios with a different event frequency: 1 minute, 1 hour, and 12 hours. As nodes sent notices to the sink on event detection, all cases differ in the average data rate. The scenarios are referred to as S-1, S-2, and S-3, respectively.

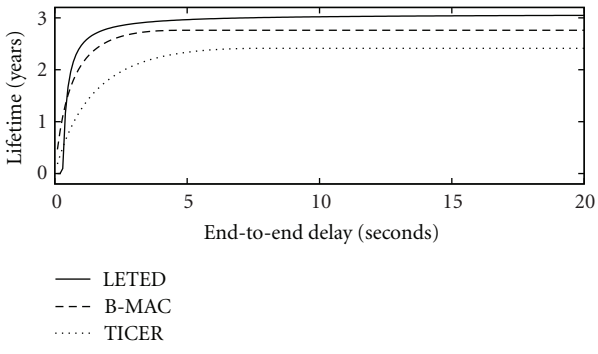
As above said, there is the optimal preamble length that achieves best results in the lifetime (see Table 4). Since this evaluation applies such preambles, it presents the best-case results of B-MAC and TICER.

6.4.1. Comparison. In S-1 LETED outperforms preamble sampling and achieves significant longer lifetimes. For example, nodes with LETED work 2x or 3x longer than those with B-MAC and TICER (see Figure 32(a)). Such a huge difference stems from the energy consumed for transmissions. LETED needs only 0.033 mAh a day for transmissions and 0.078 mAh to receive data, as presented in Figure 33. Nodes with B-MAC, however, consume an order of magnitude more energy, that is, 0.992 mAh and 0.564 mAh to send and to receive data. As LETED applies short tx and rx slots, nodes consume little energy in total when sending or receiving data. On the contrary, preamble sampling sends a long preamble in front of each frame. In this case, nodes apply a preamble of 120 ms, but the frames are only 4 ms long. Clearly, it results in a significant B-MAC and TICER overhead. Besides, on frame reception nodes listen for a half of the sleep period on average and cause extra energy waste. Therefore, preamble sampling suffers from huge idle listening especially in scenarios with high data rates.

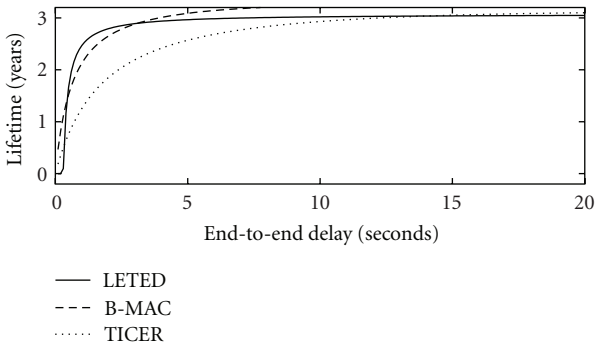
LETED and preamble sampling achieve similar results in S-2 and S-3. The lifetime of LETED is 5% to 10% better than that of B-MAC in S-2, for example, 2.96 and 2.76 years, respectively, for end-to-end delays of 5 seconds (see Figure 32(b)). In this case, B-MAC uses preambles of 940 ms and consumes slightly more energy for communication than LETED (see Figure 33). That is, B-MAC needs 0.126 mAh tx and 0.07 mAh rx energy whereas LETED consumes 0.007 mAh and 0.016 mAh, respectively. As expected, nodes with



(a) S-1 scenario: event notices sent once a minute

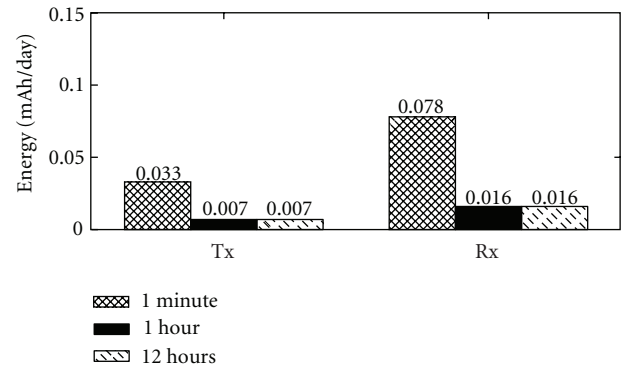


(b) S-2 scenario: event notices sent once an hour

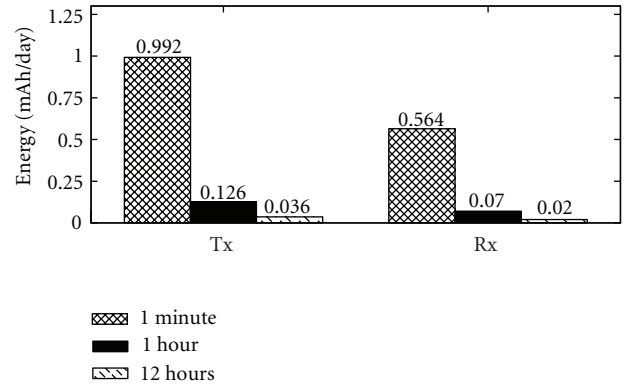


(c) S-3 scenario: event notices sent every 12 hours

Figure 32: The lifetime of nodes with LETED and with Preamble Sampling (B-MAC and TICER).



(a) LETED results. Nodes consume the same amount of energy with tx frequency 1 hour and 12 hours, since they send SYNC frame every 5 minutes in these cases



(b) B-MAC: as nodes send a long preamble in front of every data frame, they consume a huge amount of energy with high data rates (1 frame a minute)

Figure 33: Energy of data transmission and reception for various data rates (a frame sent every 1 minute, 1 hour, and 12 hours) when supporting 5-second end-to-end delays.

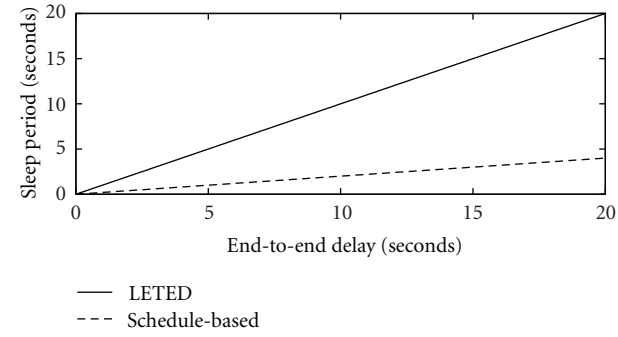


Figure 34: Sleep period of LETED and schedule-based MAC for various end-to-end delays in 5-hop networks: the longer the sleep period, the better.

TICER work shorter than B-MAC, as they need a longer time to check the channel activity (see Figure 32(b)).

In the S-3 scenario B-MAC achieves better lifetime results for delays longer than 2.4 seconds (see Figure 32(c)), since it consumes only 0.056 mAh a day in total for tx and rx (see Figure 33). Although LETED consumes even less energy for transmissions than B-MAC (see Figure 33), it needs extra energy for the underlying DLDC-MAC protocol. Therefore, nodes LETED works slightly shorter than those with B-MAC. However, there are only minor differences in lifetime between these two solutions. For example, nodes with LETED work 5% shorter than B-MAC for 5-second end-to-end delays. However, LETED still achieves better results than TICER for delays shorter than 13 seconds. As stated above, TICER results in long idle listening time due to longer periods needed to check the channel activity.

6.4.2. Preamble Sampling. The lifetime of nodes with B-MAC and TICER depends mainly on the data rate. For

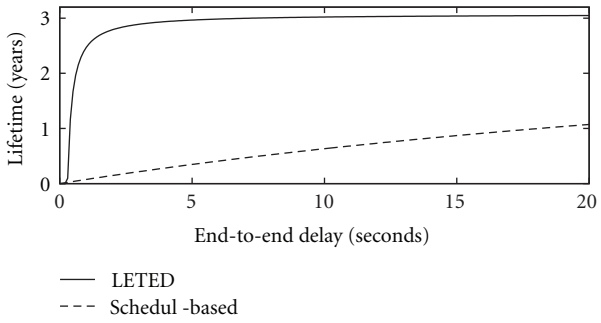


Figure 35: Lifetime of nodes with LETED and with schedule-based MAC protocols.

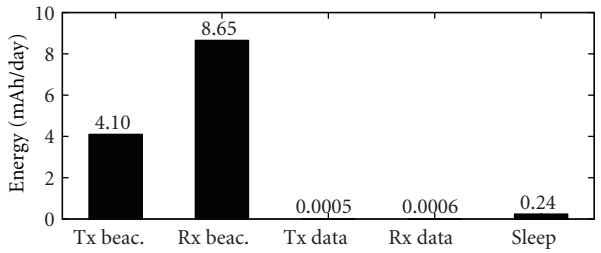


Figure 36: DLDC-MAC energy consumption; short sleep periods (1 second here) and a long time of beacons transmission and reception (about 12 ms altogether) result in an excessive energy waste.

example, nodes with B-MAC work 2.76 years when supporting 5-second delays and sending data rarely, that is, once an hour (see Figure 32(b)). Should nodes send frames once a minute, they reduce the lifetime to 1.1 years (see Figure 32(a)). TICER works in a similar way; that is, nodes work significantly shorter when sending data often. As stated above, nodes send long preambles in front of every data frame. Therefore, they consume a huge amount of energy for preamble transmissions with high data rates. For example, with a tx frequency of 1 hour nodes with B-MAC consume the tx energy of about 0.126 mAh a day (see Figure 33). Should they send frames once a minute, they increase the energy consumption 8 times to 0.992 mAh. Besides, with each frame nodes receive also a half of the preamble on average. Thus, high data rates increase also rx energy consumption (see Figure 33).

6.4.3. LETED. There are only minor differences in the lifetime of nodes based on LETED working with various data rates. For example, with an event frequency of 1 minute; that is, data rate is one frame a minute, nodes work 2.82 years and support 5-second delays (see Figure 32(a)). Should event occur once an hour, the lifetime is longer by 4% only. Such minor differences in various data rates stem from a tiny amount of energy consumed for communication in general. For instance, transmissions with 1-minute period need only 0.033 mAh a day (see Figure 33). Thus, lower data rates cannot reduce the energy consumption significantly.

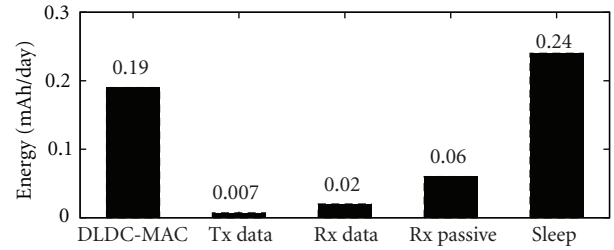


Figure 37: Energy consumed by LETED; owing to Idle Listening Avoidance nodes consume little energy while waiting for potential transmissions (Rx passive).

Two LETED scenarios with an event period of 1 hour and 12 hours achieve the same results. That is, nodes work equally long and consume the same amount of energy. In both cases nodes apply the same 5-minute period for SYNC frames to cope with multihop drift. Therefore, although events occur rarely, every 1 and 12 hours, respectively, nodes send SYNC frames every 5 minutes. As they transmit the same amount of frames in both cases, the results do not differ at all.

6.4.4. Summary. The above observations show that LETED fits better than preamble sampling (PS) to scenarios with moderate data rates, that is, about one frame a minute. In these cases, nodes with LETED work significantly longer. With lower-duty cycles PS achieves similar results to LETED.

The main advantage of PS over LETED is the small code size. For example, B-MAC needs 4 kB of ROM [9] whereas LETED with DLDC-MAC occupy about 10x more memory. Besides, PS does not rely on time synchronization and works with imprecise clock oscillators as well. Nonetheless, some PS features lead to various problems in sensor networks.

(1) As usually several sources detect the same event, they try to send notices to the sink. In this case, PS poses a high collision risk, since many sources send long preambles at the same time. Clearly, by applying the CSMA/CA approach, PS postpones transmissions on busy channels and reduce the collision risk. However, it causes extra end-to-end delays.

(2) Receivers wake up periodically to check shortly for the channel activity. The above evaluation considered the best scenario with nodes reliably detecting activities. However, receivers can wrongly sense the channel, that is, either miss an activity or detect an idle channel as active (false positive).

(a) In B-MAC nodes sample the channel to detect the activity. However, there is a risk of false positives, that is, the receivers detect an activity on idle channels. In this case, they wait the time needed to get the long preamble, do not receive a frame, and power down the radio. Obviously, it increases idle listening and causes energy waste.

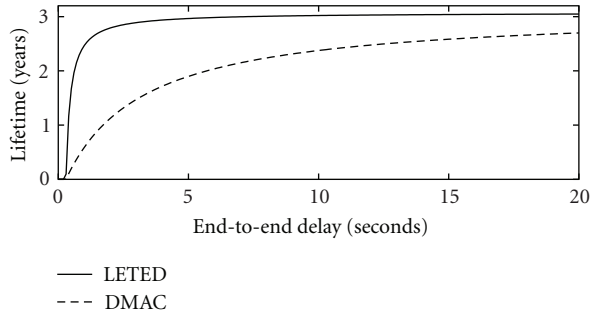


Figure 38: Lifetime of LETED and DMAC; owing to idle listening avoidance (ILA), LETED shortens significantly passive slots, saves the energy, and prolongs the lifetime.

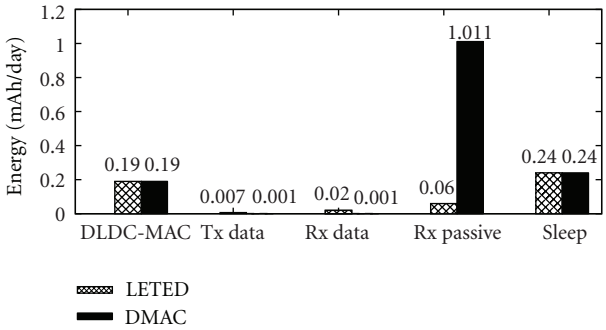


Figure 39: Energy consumed by LETED and DMAC, which represents MAC protocols with staggered schedule.

(b) Receivers with TICER expect wake-up beacons before data frames. Should receivers miss beacons, the sender must send them for another sleep period to wake up the receivers. It results in extra energy consumption and shortens the lifetime.

In general, LETED should work better in dense networks and/or with higher data rates. Owing to the TDMA, approach of LETED solves the collision problem. Besides, even with low data rates, LETED achieves as good results as B-MAC. Although the LETED size is 10x bigger than B-MAC, it fits into the limited memory of sensor nodes. In addition, as with recent developments sensor nodes get more memory, they do not suffer from large LETED size. LETED poses the risk that nodes lose the synchronization and do not wake up at the same time. For example, should the clock oscillators start running imprecisely, nodes with LETED cannot handle it and the protocol does not work. Although nodes did not encounter such oscillator problems during drift experiments introduced in our previous work [26], we cannot exclude such risks.

6.5. LETED and Schedule-Based MAC. This paragraph compares LETED with schedule-based MAC (S-B) protocols based on a wake-up schedule represented by DLDC-MAC in this case. Since schedules of S-B approaches are not aligned along the path, nodes wake up often to support short

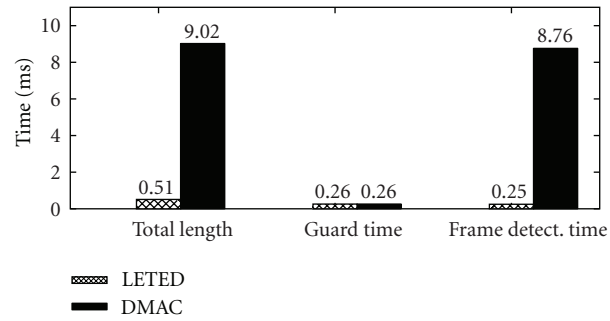


Figure 40: Passive slots consist of guard times that compensate clock drift and the time needed to detect that no frame arrives in the current slot.

delays. In general, the sleep period equals the delay time divided by the number of hops to the sink. Figure 34 presents sleep periods of LETED and S-B. Owing to the staggered schedule, nodes with LETED sleep long, almost the time equal to supported delays. For example, to support 5-second delays nodes wake up every 4.7 seconds in this case. On the contrary, S-B shortens the sleep period to about a second. Thus, nodes with S-B wake up often, consume more energy, and shorten the lifetime.

As expected, LETED outperforms S-B protocols in scenarios with short end-to-end delays. Figure 35 shows that LETED achieves an 8x longer lifetime than S-B for delays of 5 seconds and shorter. For example, nodes with LETED work almost 3 years and support 5-second delays. In this case, the S-B lifetime is only 0.35 years. For longer delays, that is, from 5 to 10 seconds, LETED achieves on average lifetimes 6x longer than S-B.

Figure 36 presents energy consumption of DLDC-MAC. As above stated, nodes wake up often to be ready for potential transmissions. With DLDC-MAC, nodes send or receive beacons during wake-up times. By doing so, they consume a huge amount of energy only to be ready for transmissions. That is, nodes need about 13 mAh for beacons altogether (see Figure 36), which is larger by four orders of magnitude from the energy consumed on data transmission. Other S-B protocols achieve similar or worse results. For example, Dozer [16] applies beacons as well but uses longer guard times than DLDC-MAC. Therefore, it spends even more energy for beacons. Besides, S-MAC [12] needs longer times in the active state than DLDC-MAC, as it uses extra RTS and CTS frames apart from guard times. Thus, DLDC-MAC achieves results close to the best case of S-B protocols.

LETED keeps nodes ready to transmissions in passive slots; that is, nodes listen periodically for incoming data. The energy of keeping nodes ready is depicted in Figure 37 as *Rx passive*. In this case, nodes spent 0.06 mAh a day to be ready for transmissions. It is about 200x less than DLDC-MAC needs for the same. As already mentioned, it stems from longer sleep periods of LETED. Besides, owing to the idle listening avoidance (see Section 4) and moving average drift compensation nodes wake up for 0.55 ms only in passive

slots. Therefore, LETED saves consumes less energy than S-B protocols.

Nodes with LETED apply also DLDC-MAC as the underlying protocol. However, in this case DLDC-MAC consumes less energy (0.19 mAh a day) than DLDC-MAC working as a stand-alone protocol (13 mAh/day) that supports short delays. In the first case, nodes do not have to send beacons often, since LETED takes care of fast transmissions along the path. They use beacons to send control frames only, for example, to set up a new schedule. Therefore, they apply a beacon period of 2 minutes. In the latter case, nodes with DLDC-MAC send beacons every second to support 5-second delays. Therefore, DLDC-MAC consumes a different amount of energy in both cases.

6.6. Staggered Schedule. In general, LETED is based on the staggered schedule introduced in DMAC [2] but applies new solutions that reduce idle listening. First, it minimizes passive slots and therefore powers down the transceiver early, referred to as ILA and introduced in Section 4. Second, LETED shortens guard times owing to the drift prediction based on the moving average. Since DMAC does not introduce a way to detect and to shorten passive slots, this work assumes it uses a generic software solution (see Section 4).

Figure 38 presents the lifetime of nodes working with LETED and with DMAC. Owing to the energy-saving solutions, LETED prolongs the lifetime by more than 50% for delays of 5 seconds and shorter. For example, with 5-second delays nodes with LETED work almost 3 years and with DMAC 2 years. LETED achieves such good results, as it reduces significantly the energy consumption of passive slots (see Figure 39). That is, nodes with DMAC need 16x more energy in passive slots than LETED.

In this scenario, LETED shortens passive slots by more than 10x, that is, from 9 ms to 500 μs (see Figure 40). Each passive slot consists of two parts: a guard time and the time t_{detect} needed to detect that no frame arrives in the current slot. In this case, t_{detect} of LETED equals 250 μs, as nodes need approximately. 150 μs to receive a preamble with the start frame delimiter (SFD) and about 100 μs to handle the SFD interrupt. Should the transceiver not raise the SFD interrupt within this time, nodes power it down. DMAC, however, uses a generic software approach. That is, it usually needs long times to get frames from the tx buffer, almost 9 ms in this case. Thus, after 9 ms listening nodes do not receive frames, assume that nothing arrives in the current slot, and switch off the radio. Clearly, such a huge difference in t_{detect}, that is, 0.25 ms versus 8.76 ms, is the main reason for such good results of LETED.

In this work, LETED and DMAC applied the same solution to drift compensation and thus do not differ in the length of guard times (see Figure 40). However, DMAC should uses an external time synchronization protocol and not the energy-efficient solution to guard times based on the moving average. Therefore, DMAC should achieve even worse results.

7. Future Work

The idea of load balancing was not considered, that is, alternating routes from sources to the sink. Thus, the network uses the same routes for a long time, exhausting energy of some nodes in an early stage. In this case, the network may partition, and some sources cannot reach the sink. Therefore, routing protocols should alternate paths to distribute the communication burden, preventing the partition risk. Clearly, it involves a cooperation with LETED to set up wake-up schedules along many paths. These challenges will be investigated in future work.

Sensor nodes can also provide load balancing by using multiple paths to the sink in parallel. In this case, the wake-up times alternate between paths. For example, if nodes with LETED have to support 5-second delays, they wake up every 5 seconds. However, if there are two paths to the sink available, intermediate nodes wake up every 10 seconds, provided sources can select any of two paths for transmissions. In this case, sources can send data using the path that has earlier wake-ups. Owing to this solution, the network spreads the wake-up load to multiple nodes and prevents early partitioning. Another aspect of load balancing addresses the problem of nodes that are on many gathering paths, for example, as they are close to the sink, and gathering paths converge towards them. Obviously, if such nodes frequently wake up, they quickly exhaust energy and stop working. Thus, future work will consider solutions that reduce the duty cycle of such nodes without affecting end-to-end delays. For example, these nodes can wake up more rarely and save energy, as previous nodes send some frames with full TX power and directly reach the sink.

Currently, the support of wake-up schedules from different nodes is not efficient, as LETED sets up a separate schedule for each source. As a result, intermediate nodes maintain schedules of several sources and wake up frequently. To save energy, LETED should limit the number of wake-up schedules. For example, there should be only a few global schedules in the network, and nodes should use common wake-up slots instead of separate ones. We intend to address this issue in future research efforts.

Although some wireless links are asymmetric, DLDC-MAC ignores such links by default and LETED does not use them for transmissions. First, the amount of asymmetric links is rather small, and discarding them should not affect connectivity considerably. According to [37], 5–15% of all links are asymmetric. Another experiment [38] revealed that more than 10% of link pairs have a packet loss difference greater than 50%, meaning they are asymmetric. Second, the use of asymmetric links results in extra transmission overhead. For example, when applying the ARQ protocol, nodes cannot send acknowledgments directly to senders but over multihop paths. In our future work, we intend to investigate the impact of asymmetric links on LETED, especially on energy consumption and on the communication performance.

8. Conclusion

This work introduced the main challenges of particular sensor network applications. That is, nodes must work for a long time and support short end-to-end delays in multihop networks. It presented previous research efforts that address such challenges. For instance, many medium-access control (MAC) protocols keep nodes mostly sleeping to save energy and synchronize wake-up times to allow communication. Some of them, for example, DMAC, align the wake-up schedule in a way that it minimizes multihop delay in data transfer. Although such protocols support short end-to-end delays and a lifetime of several years, they still suffer from idle listening. The main reason is the long time needed to detect an idle channel and to power down the radio. This work introduces a novel solution to this problem and exploits features of some off-the-shelf transceivers. By doing so, it reduces idle listening about 15x and prolongs the lifetime significantly, that is, by approximately. 30% in some scenarios.

A lifetime evaluation model for various MAC protocols was presented in this work. The model estimates the lifetime and energy consumption according to various parameters of hardware and software. It turned out that off-the-shelf sensor nodes achieve good results in lifetime and end-to-end delays. For example, should nodes apply solutions presented here and support 5-second delays in multihop networks, they work as long as 3 years. Besides, the evaluation considered standard rechargeable batteries with a significant self-discharge rate. Therefore, with dedicated long-life batteries, nodes work even longer.

In this work we evaluated also our solution for limiting end-to-end delays (LETED) with other duty-cycled MAC protocols. Owing to the energy-efficient solutions, LETED achieves better results than protocols based on preamble sampling. For example, with a data rate of 1 frame per minute, nodes with LETED work 2x or 3x longer. In addition, LETED outperforms common TDMA protocols in scenarios that need short end-to-end delays. That is, LETED prolongs the lifetime by 8x when nodes support 5-second delays in multihop networks.

Although this work addressed particular applications only, the solutions presented here can be used in other scenarios and with different protocols. For instance, any TDMA protocol can benefit from the solution to the idle listening reduction based on the early detection of idle channel. Besides, owing to the analytical model, researches and developers can quickly estimate energy consumption and the lifetime of low-duty cycle protocols running on various hardware platforms.

References

[1] Moteiv Corporation, "Tmote Sky Ultra low power IEEE 802.15.4 compliant wireless sensor module," 2006, http://www.sentilla.com.

[2] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "An adaptive energy-efficient and low-latency MAC for tree-based data gathering in sensor networks," *Wireless Communications and Mobile Computing*, vol. 7, no. 7, pp. 863–875, 2007.

[3] N. A. Vasanthi and S. Annadurai, "Energy efficient sleep schedule for achieving minimum latency in query based sensor networks," in *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, pp. 214–219, June 2006.

[4] E. Y. A. Lin, J. M. Rabaey, and A. Wolisz, "Power-efficient Rendez-vous schemes for dense wireless sensor networks," in *Proceedings of the IEEE International Conference on Communications*, pp. 3769–3776, June 2004.

[5] L. C. Zhong, R. Shah, C. Guo, and J. Rabaey, "An ultra-low power and distributed access protocol for broadband wireless sensor networks," in *Proceedings of the IEEE Broadband Wireless Summit*, vol. 3, 2001.

[6] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava, "Optimizing sensor networks in the energy-latency-density design space," *IEEE Transactions on Mobile Computing*, vol. 1, no. 1, pp. 70–80, 2002.

[7] A. El-Hoiydi, "Aloha with preamble sampling for sporadic traffic in Ad Hoc wireless sensor networks," in *Proceedings of the International Conference on Communications (ICC '02)*, pp. 3418–3423, May 2002.

[8] A. El-Hoiydi, "Spatial TDMA and CSMA with preamble sampling for low power ad hoc wireless sensor networks," in *Proceedings of the ISCC*, 2002.

[9] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of the Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pp. 95–107, November 2004.

[10] A. El-Hoiydi and J. D. Decotignie, "WiseMAC: an ultra low power MAC protocol for multi-hop wireless sensor networks," in *Proceedings of the ALGOSENSORS*, 2004.

[11] M. E. Răzvan, C. J. M. Liang, and A. Terzis, "Koala: ultra-low power data retrieval in wireless sensor networks," in *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN '08)*, pp. 421–432, April 2008.

[12] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *Proceedings of the IEEE Infocom*, pp. 1567–1576, June 2002.

[13] T. Van Dam and K. Langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks," in *Proceedings of the First International Conference on Embedded Networked Sensor Systems (SenSys '03)*, pp. 171–180, November 2003.

[14] B. Hohlt, L. Doherty, and E. Brewer, "Flexible power scheduling for sensor networks," in *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN '04)*, pp. 205–214, April 2004.

[15] B. Hohlt and E. Brewer, "Network power scheduling for TinyOS applications," in *Proceedings of the DCOSS*, 2006.

[16] N. Burri, P. Von Rickenbach, and R. Wattenhofer, "Dozer: ultra-low power data gathering in sensor networks," in *Proceedings of the 6th International Symposium on Information Processing in Sensor Networks (IPSN '07)*, pp. 450–459, April 2007.

[17] IEEE Standard for Information Technology, Specific requirements Part 15.4: Wireless MAC and PHY, 2006.

[18] M. Brzozowski, K. Piotrowski, and P. Langendoerfer, "A cross-layer approach for data replication and gathering in decentralized long-living wireless sensor networks," in *Proceedings of the International Symposium on Autonomous Decentralized Systems (ISADS '09)*, pp. 49–54, March 2009.

[19] M. Brzozowski, H. Salomon, and P. Langendoerfer, "Completely distributed low duty cycle communication for long-living sensor networks," in *Proceedings of the 7th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC '09)*, pp. 109–116, August 2009.

[20] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic, "Towards optimal sleep scheduling in sensor networks for rare-event detection," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN '05)*, pp. 20–27, April 2005.

[21] M. J. Miller, C. Sengul, and I. Gupta, "Exploring the energy-latency trade-off for broadcasts in energy-saving sensor networks," in *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, pp. 17–26, June 2005.

[22] W. Lai and I. C. Paschalidis, "Sensor network minimal energy routing with latency guarantees," in *Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '07)*, pp. 199–208, September 2007.

[23] M. Brzozowski and P. Langendoerfer, "On prolonging sensornode gateway lifetime by adapting its duty cycle," in *Proceedings of the WWIC*, 2009.

[24] T. Schmid, J. Friedman, Z. Charbiwala, Y. H. Cho, and M. B. Srivastava, "Low-power high-accuracy timing systems for efficient duty cycling," in *Proceedings of the 13th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '08)*, pp. 75–80, August 2008.

[25] S. Ganeriwal, I. Tsigkogiannis, H. Shim, V. Tsiatsis, M. B. Srivastava, and D. Ganesan, "Estimating clock uncertainty for efficient duty-cycling in sensor networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 3, pp. 843–856, 2009.

[26] M. Brzozowski, H. Salomon, and P. Langendoerfer, "On efficient clock drift prediction means and their applicability to IEEE 802.15.4," in *Proceedings of the IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC '10)*, pp. 216–223, December 2010.

[27] M. Brzozowski, H. Salomon, and P. Langendoerfer, "Limiting end-to-end delays in long-lasting sensor networks," in *Proceedings of the 8th ACM International Symposium on Mobility Management and Wireless Access (MobiWac '10)*, pp. 11–20, October 2010.

[28] S. Lin and D. J. Costello, *Error Control Coding*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1983.

[29] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99)*, pp. 90–100, February 1999.

[30] M. Brzozowski, H. Salomon, and P. Langendoerfer, "ILA: idle listening avoidance in scheduled wireless sensor networks," in *Proceedings of the WWIC*, 2010.

[31] Texas Instruments, "2.4 GHz IEEE 802.15.4/ZigBee-ready RF Transceiver," 2007, http://focus.ti.com/docs/prod/folders/print/cc2420.html.

[32] A. Varga, "Using the OMNeT++ discrete event simulation system in education," in *Proceedings of the ESM*, 2001.

[33] S. Hoeckner, A. Lagemann, and J. Nolte, "Integration of event-driven embedded operating systems into OMNet++—a case study with reflex," in *Proceedings of the SIMUTools*, 2009.

[34] K. Walther and J. Nolte, "A flexible scheduling framework for deeply embedded systems," in *Proceedings of the 21st International Conference on Advanced Information Networking and ApplicationsWorkshops/Symposia (AINAW '07)*, pp. 784–791, May 2007.

[35] M. Brzozowski, H. Salomon, K. Piotrowski, and P. Langendoerfer, "Cross-platform protocol development for sensor networks: lessons learned," in *Proceedings of the 2nd International Workshop on Software Engineering for Sensor Network Applications (SESENA '11)*, pp. 7–12, May 2011.

[36] T. S. Rappaport et al., *Wireless Communications: Principles and Practice*, Prentice Hall PTR, NJ, USA, 2002.

[37] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, "Complex behavior at scale: an experimental study of lowpower wireless sensor network," Tech. Rep. UCLA/CSD-TR 02, 2002.

[38] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor," in *Proceedings of the Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03)*, pp. 1–13, November 2003.