Limiting End-to-End Delays in Long-Lasting Sensor Networks

Marcin Brzozowski, Hendrik Salomon and Peter Langendoerfer
IHP
Im Technologiepark 25
15236 Frankfurt (Oder), Germany
{brzozowski, salomon, langendoerfer}@ihp-microelectronics.com

ABSTRACT
Many applications require a lifetime of several years from a sensor network while expecting low and guaranteed end-to-end delays between sources and a sink. Obviously, these two parameters - lifetime and delay - contradict each other. In this work we present and evaluate a solution that limits the end-to-end delays and nevertheless achieves a long lifetime.

We introduce a model for evaluation of delay and lifetime in multi-hop sensor networks. According to our model a network of off-the-shelf sensor nodes limits an end-to-end delay to 5 seconds and works for 8 months. However, if applications can tolerate the end-to-end delay of 20 seconds, the nodes prolong the lifetime to approx. 2 years.

Our evaluation revealed that end-to-end delay affects the lifetime only to a certain limit. In our example this limit was 60 seconds, i.e. any delay change above 60 seconds does not influence the lifetime considerably.

Categories and Subject Descriptors
C.2.1 [Computer-Communication Networks]: Network Architecture and Design — Wireless communication

General Terms
Design, Measurement

Keywords
clock drift, energy efficiency, medium access protocols, quality of service, sensor network

1. INTRODUCTION
Several applications, especially in the area of automation control, homeland security or critical infrastructure protection, require guaranteed end-to-end delays. In other words, if an event occurs in the area monitored by a sensor network, the notification message must reach the sink within a desired time. Moreover, these applications require often a long lifetime of sensor nodes, even several years. On the one hand, nodes should sleep over long periods in order to maximize the lifetime. On the other hand, nodes should shorten sleeping periods to guarantee end-to-end delays. Obviously, these two parameters - lifetime and delay - contradict each other.

In this work, we present our approach for limiting end-to-end delay (LETED) and achieving a long lifetime. Firstly, nodes using LETED maintain forwarding time slots to support end-to-end delay limits. Slots are arranged similarly to the stacked schedule (DMAC [10] and [5]), i.e. upon receiving a frame in a receive slot each node forwards it almost immediately to the next node during the following transmit slot. In this way nodes shorten the forwarding delay. Secondly, to achieve a long lifetime we based LETED on an underlying low duty cycle (LDC) protocol.

Although several research efforts introduced the stacked schedule, neither of them sufficiently considered the problem of clock drift. However, in our previous work [2] we showed empirically that the drift problem cannot be neglected in long-living scenarios. In general, approaches based on active/sleep schedule may suffer from the following drift problems:

1. Forwarding slots of neighbours may overlap after time passes. As a result, nodes do not receive data.
2. To avoid the risk of waking up too soon or too late to receive a message, each node wakes up earlier and stays awake longer by a guard time. The common approach for guard times considers worst drift. However, such a solution causes unnecessary long idle listening, as run-time drift is much smaller than the worst case.

To find a reasonable solution to these problems, we examined run-time drift in a two-week drift experiment. During the experiment we measured drift of three sensor nodes in an environment with changing temperature. Typically sensor nodes have a drift parameter of 30-100 parts per million (ppm). Our experiment confirmed that run-time drift is smaller, approx. 5 ppm. Moreover, we expected that the temperature change affects clock drift. However, we found out that only the drift average and not drift jitters are affected by the temperature change. In other words, the local clock of a node oscillates around the drift average within an almost constant window (i.e. around 10 ticks a minute in our experiment) regardless of the temperature. This observation encouraged us to design new solutions to clock drift compensation.
In short, nodes using any LDC protocol which provides information about run-time drift to neighbours can use LETED - introduced in this paper - to cope with challenges stemming from clock drift. Each node shifts its time slots repeatedly to prevent the overlap risk. Moreover, the estimation of run-time drift allows the use of very short guard times, and consequently decreases idle listening. Finally, since sensor nodes estimate drift repeatedly, they adapt to changing drift and LETED works well even in outdoor scenarios.

In this paper we present also a sophisticated model for analyzing the trade-off between the end-to-end delay and the lifetime. The model considers almost 30 parameters (hardware and protocol). For example, we included self-discharge of batteries, often neglected, which is the major energy consumer (even 50% of the total energy) in LDC protocols. The model can be used to analyze various protocols. For example, IEEE 802.15.4 [7] MAC (Medium Access Control) protocol in beacon mode can be immediately evaluated. Other protocols may need some model adaptations.

We evaluated LETED using an off-the-shelf sensor node without any changes, i.e. tmote sky [12] with 2xAA batteries only. Nonetheless, LETED achieves good results for the end-to-end delay and the lifetime. For example, a multi-hop sensor network supports an end-to-end delay of 5 seconds and still achieves a lifetime of 8 months. Moreover, changing the end-to-end delay requirement to 20 seconds prolongs the lifetime to almost 2 years! Our evaluation revealed that there is a certain delay, when the lifetime does not prolong significantly, i.e. in our example it was a delay of a minute.

The main contributions of this work are:

1. An analytical model for analyzing the trade-off between the end-to-end delay and the lifetime in multi-hop sensor networks
2. Solution to limiting end-to-end delays in long-living sensor networks in the presence of clock drift
3. Empirical data of clock drift with changing temperature

The paper is organized as follows. Section 2 gives an overview about the scientific efforts related to end-to-end delays and low duty cycle protocols in sensor networks. Section 3 shows the experimental results of clock drift. In Section 4 we give an overview of the underlying LDC protocol. Section 5 introduces our new LETED solution to limiting end-to-end delays. We present in Section 6 our analytical model. In Section 7 we examine the trade-off between the end-to-end delay and the lifetime. Finally, we conclude the paper.

2. RELATED WORK

LDC protocols for wireless sensor networks have been extensively studied, e.g. SMAC [19], DMAC [10], B-MAC [15], Dozer [4], Koala [13]. Although many LDC research efforts evaluate forwarding delays, they do not address the problem of limiting end-to-end delays. A general solution to low end-to-end delays resembles the idea of stacked schedule presented in DMAC [10], Q-MAC [18] and in [5].

As nodes may maintain a stacked schedule for a very long time, it suffers from clock drift problems, as already mentioned. However, many approaches (e.g.[10], [18]) completely neglected the drift problem. Ref. [5] uses a stacked schedule as well, called streamlined wake up in the paper, and counters implicitly the drift problem. The authors assume that the neighbours have synchronized clocks, as the nodes use a time synchronization protocol (TSP). However, TSP results in additional overhead in terms of memory occupation, very limited on sensor nodes, and additional TSP control frames. On the contrary, we do not use any underlying TSP in our approach.

Although another research work [17] does not consider end-to-end delays, it presents a solution to clock drift problem by adapting TSP. Briefly, each node receives consecutive frames from a neighbour, estimates relative drift and shifts the schedule. Although the solution works for 1-hop neighbourhood, it cannot counter the drift problem of multi-hop paths. For example, if some nodes on a path have larger drift than the others, their slots overlap after some time, although they use the above-mentioned solution. On the contrary, our LETED approach solves the drift problem in multi-hop paths, as it considers drift of the whole path, and not drift of 1-hop neighbourhood only.

In [3] we present a solution to the idle listening reduction in scheduled protocols by detecting idle slots and switching transceiver almost immediately. Although the solution can extend the lifetime by even 50%, it depends on the underlying transceiver hardware. Nonetheless, we can apply it to off-the-shelf sensor nodes (tmote sky) and extend the lifetime of LETED considerably.

To preserve energy sensor nodes monitor the covered area periodically, i.e. 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. Ref. [5] 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.

Some works investigated the trade-off between delay and lifetime. The SMAC authors present in [19] energy savings vs. average sleep delay trade-off. Ref. [6] presents the mean delay and achieved lifetime of CSMA (carrier sense multiple access) and of various TDMA (time division multiple access) approaches. The trade-off relationship between the expected lifetime extension and the corresponding increase in the average detection delay achieved by different sleep scheduling algorithms is introduced in [5]. Ref. [11] explores the energy-latency trade-off for broadcast communication in sensor networks. In [8] 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 [1] we examined the trade-off 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 802.11g network.

3. DRIFT EXPERIMENT

To examine clock drift characteristic of typical sensor nodes we carried out a drift experiment and present the results in this section.

3.1 Description

As clock drift depends on the environmental conditions, mostly on the temperature, we deployed three sensor nodes (tmote sky) in a sunny area. The nodes sent beacons in-
Figure 1: Relative drift to the sink (in ticks per minute; 1 tick = 30.5 microseconds) of three examined nodes together with their temperature; we marked midday by vertical grid lines.

Figure 2: Drift distribution among all received messages; we present the experiment samples (raw drift) and the estimated values when the node measures run-time drift (DER: drift estimated repeatedly)

3.2 Results

The nodes temperature changed from approx. 25 °C to more than 50 °C, see Figure 1 (we show 1-week data only, as the second week results are very similar). However, the sink was not exposed to sunlight, and therefore its temperature was constant, about 25 °C. In this way, the relative temperature between the sink and the sensor nodes changed and affected relative drift between them. For instance, the sink received frames of node A with drift of approx. -3 ticks / minute during the nights, see Figure 1a. However, during the days the temperature of node A risen to 50 °C and relative drift of the sink and node A changed by almost 20 ticks / minute. We noticed similar results on nodes B and C, see Figures 1b and 1c.

Although the nodes temperature changed and affected drift, the drift distribution still resembles Gaussian, see Figure 2 (data set: raw drift). However, as drift was higher during the daytime, the distribution is slightly asymmetric.

We examined the drift measurements and found out that although drift changed with the temperature, it kept oscillating only by a few ticks from the average calculated among last two drift samples! This observation revealed the need of the continuous run-time drift estimation. In this way, nodes adapt to changing drift and receive most messages in a very small drift window.

Based on the drift measurements we examined analytically the benefits of the run-time drift estimation. When the nodes estimate run-time drift repeatedly, they receive approx. 99% messages within a drift window of 10 ticks from the average, see Figures 2 and 3. On the contrary, the same nodes without the run-time drift estimation need a drift window of 40 ticks to receive so many messages.

4. DLDC-MAC OVERVIEW

In this section we present DLDC-MAC [2], which is the basis for our LETED approach.

4.1 Beacons

Using the DLDC-MAC protocol each node transmits a beacon periodically and wakes up to receive beacons of its neighbours. The beacon period is the same for all nodes. Even when a node does not have data (e.g. sensor readings) to send, it still needs to send beacons.

Upon receiving a neighbour’s beacon, the node estimates the next beacon time by adding the beacon period to the reception time. Since the node performs this routine for
several neighbours, the node knows the next beacon times of its neighbours. Therefore, each node sleeps most of the time and wakes up only to receive neighbours beacons and to send its own beacons.

DLDC-MAC maintains 2-hop neighbourhood information (addresses, beacon times) exploited by LETED.

In [2] we described the protocol details (schedule setup, dealing with link failures, beacon overlap prevention etc.).

4.2 Clock drift

A receiver may miss a beacon, as it wakes up too soon or too late due to clock drift. To counter this threat, sensor nodes wake up earlier by a guard time (GT), and also stay awake longer. To reduce idle-listening introduced by guard intervals we propose that nodes estimate run-time drift to each neighbour repeatedly, i.e. on each beacon reception. Knowing run-time drift nodes use very short GT instead of long GT based on worst-case drift.

5. LETED

In this section we introduce our solution for limiting end-to-end delays ( LETED), which copes with challenges stemming from clock drift.

5.1 Overview

To limit end-to-end delays, nodes on the path to the sink establish a wake-up schedule. Each node on the path arranges a receive (rx) slot to the previous node and a transmit (tx) slot to the next node. The tx slot follows the corresponding rx slot, see Figure 4. To keep the end-to-end delay small, nodes forward messages just after the reception, i.e. the time offset between rx and tx slots is very small.

LETED maintains schedules for explicitly defined gathering paths only, i.e. nodes establish schedules on demand. Moreover, LETED can maintain schedules for paths with different end-to-end delay requirements.

5.2 Schedule setup

In LETED a sink triggers nodes to set up a new wake-up schedule for a gathering path. The sink sends to sources a request containing the maximum delay between an event detection and the notification of the sink. In other words, after a source detects an event, the sink must receive a notification within the maximum delay time.

The schedule setup involves a cross-layer cooperation, presented in Figure 5. When a source receives a schedule setup request, the application layer triggers the network layer to establish a new schedule on the path towards the sink. The application specifies the maximum end-to-end delay \( d_{E1E} \).

Then, the network layer determines the next hop and triggers LETED, coupled with DLDC-MAC in this example, to establish new time slots to the next node. However, the path selection depends highly on the routing protocol and the application requirements (e.g. a path with a short delay or with nodes having enough power resources) and is beyond the scope of this paper. Nonetheless, LETED requires only from routing protocols to deliver the next node only. However, if there is currently no route to the sink, the routing protocol establishes a new path and then provides the next node.

The source node determines how often nodes must wake up in order to support \( d_{E1E} \). In general, when an event occurs, the source node does not send a notification immediately but waits for the next tx slot, \( \text{delay to 1st time slot} \) in Figure 4. To support \( d_{E1E} \), the source node needs tx slots every \( d_{E1E} \) time. However, forwarding nodes cause additional delays in multi-hop networks, see Figure 4. Therefore, the source node estimates the total forwarding delay \( d_{\text{forwarding}} \) and calculates the slot period \( T_{\text{slot}} \), i.e. the sleep time between consecutive tx slots on each node, needed to support \( d_{E1E} \) as:

\[
T_{\text{slot}} = d_{E1E} - d_{\text{forwarding}} \quad (1)
\]

\[
d_{\text{forwarding}} = n \cdot (t_{\text{frame}} + t_{\text{txoffset}}) \quad (2)
\]

where \( n \) is the number of hops to the sink (available as the source already fixed a route to the sink), \( t_{\text{frame}} \) the ex-

Figure 3: The number of messages received within various drift windows (guard times) for raw drift samples and the approach based on DER (drift estimated repeatedly)

Figure 4: The event notification delay in multi-hop networks with duty cycling results mainly from the delay to the transmit slot on the source and from forwarding delays on each intermediate node (router)
To establish a new schedule along a path we need a cross-layer approach of the network layer (Routing) and of the data link layer (LETED coupled with DLDC-MAC).

Due to clock drift slots of different nodes move relatively to each other; in this example drift(A) < drift(source) < drift(B).

After the source estimated the slot period, it adds new tx slots to its schedule. Then, the source sends a schedule setup request (together with the times of the new tx slots) to the next node. After receiving the request the next node adds corresponding rx slots to its schedule and sends back an acknowledgment, or an negative acknowledgment when the new slots overlap with existing ones.

In next steps each node on the path establishes time slots to the next node in a similar way, see Figure 5. In general, as the source already fixed the route, each node can determine immediately the next node on the path (e.g. looks up the routing table, like in AODV [14]). Finally, nodes finish the schedule setup and limit end-to-end delays of this path.

In this example we benefit from the underlying DLDC-MAC and send LETED control frames (requests and acknowledgments) piggybacked in beacons. In that way we keep the LETED overhead very small, i.e. just a few additional bytes in beacons.

### 5.3 Schedule cancellation

If a sink does not need to gather data anymore, it sends a cancellation message to the sources. On receiving such a message each source discards the corresponding schedule. Then, sources send cancellation messages throughout the gathering paths and each node discards the corresponding schedule.

### 5.4 Time slot overlap

#### 5.4.1 Problem statement

Time slots of different nodes move relatively to each other due to clock drift, see Figure 6. If slots move towards each other, there is a risk of overlap and severe communication problems (e.g. collisions). For example, if relative drift between nodes A and B is 3 ppm and slot B follows slot A after 50 ms, the slots overlap after about 3.5 hours and stay overlapped for 13 minutes (each slot 128 bytes), causing communication problems due to collisions.

If slots drift away, the forwarding delay increases. Moreover, if slots keep moving relatively to each other, they may become completely disorganized. As a result, nodes cannot support end-to-end delay requirements.

#### 5.4.2 Solution

Each node adapts its schedule according to clock drift repeatedly, i.e. the times of slots 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 determine relative drift to the source in order to shift the time slots. In this example DLDC-MAC provides estimated run-time drift. Each node having a schedule sends to the next node its relative drift to the time of the event notification frames (time units) and $t_{tx,offset}$ the time between rx and tx slots on a forwarding node. Each node uses the same value of $t_{tx,offset}$, explained later in Section 5.6.

Each node adapts its schedule according to clock drift repeatedly, i.e. the times of slots remain stable relatively to the source tx slots. As a result, the end-to-end delay remains constant and time slots do not overlap.

1DLDC-MAC measures run-time drift to neighbours each time a beacon is received.
the source repeatedly, piggybacked in DLDC-MAC beacons in this example. Upon receiving the value of drift to the source the receiver adds to its relative drift to the sender. In that way, each node determines relative drift to the source. In general, each node with a schedule shifts its time slots in the following way:

1. After rx time slot finishes, the node calculates the time of this slot $t_{\text{rx}\text{next}}$ in the next beacon period as:

$$t_{\text{rx}\text{next}} = t_{\text{rx}\text{now}} + T_{\text{beacon}} + \delta_{\text{src}} \cdot T_{\text{beacon}} - g$$

where $T_{\text{beacon}}$ is the beacon period and $g$ is the guard time used for the next slot. Clearly, each node adapts its schedule according to relative drift to the source $\delta_{\text{src}}$. We use the same guard time as for beacon reception in DLDC-MAC (e.g. nodes prolong guard time after missing beacons, as we treat beacons as synchronization points), see Eq. 22 in our model. $t_{\text{rx}\text{now}}$ is the time of the slot just finished.

2. When a transmit time slot finishes, the node estimates the slot time in the next beacon period as:

$$t_{\text{tx}\text{next}} = t_{\text{tx}\text{now}} + T_{\text{beacon}} + \delta_{\text{src}} \cdot T_{\text{beacon}}$$

where $t_{\text{tx}\text{now}}$ is the time of the slot just finished.

### 5.5 Schedule overlap

#### 5.5.1 Time slot - time slot overlap

As already stated, nodes on gathering paths shift their schedule repeatedly to keep the slot times unchanged relatively to the source node. Thus, the of different gathering paths may move towards or apart from each other by relative drift of two corresponding source nodes, see Figure 7. As a result, time slots of different schedules may overlap.

To counter the overlap threat, each node checks if there is an overlap risk, i.e. the time between any two slots of a 2-hop neighbourhood (provided by DLDC-MAC) is shorter than a threshold. In this way, even nodes that do not maintain any schedule can detect an overlap risk.

When a node detects an overlap risk, it triggers the source node of the shorter path\(^2\) to change its schedule by sending a schedule shift request. The schedule change of a shorter path involves less schedule shift overhead. The request contains the time offset and relative drift of both affected schedules so that the source can estimate the desired shift time.

After receiving the request the source shifts the affected tx slot only, and not the whole schedule, by the shift time. After that, it sends shift requests incl. the shift time along the path. Then, each node shifts only the affected slot by the same shift time. Depending on relative drift of the colliding schedules, the nodes move the slot shortly after or before the other colliding slot. Figure 7 presents the case when the nodes shift the affected slots to the later time.

As the nodes shift a single tx slot only, and not the whole schedule, the time interval between the preceding or the following slot changes. Clearly, it affects the end-to-end delay. For example, if the nodes shifted the slot to the later time, as in Figure 7, the time difference to the previous tx slot is larger than before, compare Eq. 1 and 2. As a result, the end-to-end delay may be longer than specified at the beginning. Therefore, if an application requires a rigid end-to-end delay guarantee, the nodes move the shifted slot back to its origin after some time. As the source received relative drift between both affected schedules, it calculates the time when the nodes can shift safely the slot back to its origin without an overlap risk.

#### 5.5.2 Time slot - beacon overlap

Beacons and time slots may move relatively towards each other due to clock drift. If a node discovers that a beacon and a time slot may overlap, it triggers the beacon sender to change its beacon time. Obviously, the beacon sender can be the node itself. We decided to shift beacons and not time slots, since it results in less overhead. In our previous work [2] we presented the results of an experiment with nodes using DLDC-MAC. When a node detected a beacon overlap risk, it kept sending approx. five additional bytes (the new beacon time and the round number when the beacon is shifted) in ten following beacons. After that, the beacon was shifted without problems. We use the same solution in LETED.

### 5.6 Offset between Rx-Tx slots

As already stated, on each forwarding node tx slot should follow almost immediately the corresponding rx slot in order to keep forwarding delay small, see Figure 4 and Eq. 2. However, the offset between tx and rx slots $t_{\text{tx}\text{offset}}$ must not be too small, as it may cause slot overlap. As each node shifts its slots repeatedly to prevent the overlap, see Section 5.4, the minimal $t_{\text{tx}\text{offset}}$ must compensate drift arisen between two consecutive slot shifts, i.e. during the beacon period $T_{\text{beacon}}$ of DLDC-MAC. Thus, we estimate the minimal $t_{\text{tx}\text{offset}}$ as:

$$t_{\text{tx}\text{offset}} = T_{\text{beacon}} \cdot \delta_{\text{worst}}$$

where $\delta_{\text{worst}}$ is worst-case drift to the next node, specified for each oscillator type (e.g. +/- 20 ppm for tmote sky sensor node). We have to use worst-case drift and not current runtime drift to the next node, as the latter may change over time, e.g. due to temperature variation, and cause a slot overlap. For example, tmote sky using DLDC-MAC with 1-minute beacon period needs $t_{\text{tx}\text{offset}}$ of at least 3 ms.

### 5.7 Topology change

We are aware that a routing protocol may change a source-sink path, referred to as re-routing, e.g. due to link failures. In that case new nodes on the path do not maintain LETED schedules yet and cannot guarantee end-to-end delays. Thus, a node that discovers a new path creates a new schedule on the following nodes, similarly to schedule setup in section 5.2. Clearly, if there are more nodes on the new path than on the previous one, the end-to-end delay may increase, compare Eq. 1 and 2. Hence, the application with rigid end-to-end delay requirements sets up a new schedule for the whole new path, i.e. starting from the source.

\[^2\]the network layer may provide the path length; otherwise, the node sends a path-length query to the source node.
Table 1: Protocol parameters of the energy consumption model together with values used for evaluation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_{frame}$</td>
<td>data frame length</td>
<td>128 bytes</td>
</tr>
<tr>
<td>$t_{tx,offset}$</td>
<td>the time a tx slot follows the corresponding rx slot</td>
<td>50 ms</td>
</tr>
<tr>
<td>$T_{event}$</td>
<td>how often events occur</td>
<td>60 secs</td>
</tr>
<tr>
<td>$n$</td>
<td>hop count: source to sink</td>
<td>5</td>
</tr>
<tr>
<td>$p$</td>
<td>the number of gathering paths the node is part of</td>
<td>5</td>
</tr>
<tr>
<td>$d_{EIE}$</td>
<td>maximum end-to-end delay</td>
<td>variable</td>
</tr>
<tr>
<td>$T_{mcuactive}$</td>
<td>how long mcu is active a day when radio is powered down</td>
<td>10 minutes</td>
</tr>
</tbody>
</table>

Parameters of the underlying LDC protocol:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{beacon}$</td>
<td>beacon period</td>
<td>120 secs</td>
</tr>
<tr>
<td>$\lambda_{beacon}$</td>
<td>beacon length</td>
<td>128 bytes</td>
</tr>
<tr>
<td>$\lambda_{beacon,after}$</td>
<td>how long (bytes) the node waits in listening after sending a beacon</td>
<td>128 bytes</td>
</tr>
<tr>
<td>$nbours$</td>
<td>the number of neighbours</td>
<td>4</td>
</tr>
<tr>
<td>$MBR$</td>
<td>the average missed beacon rate</td>
<td>1%</td>
</tr>
</tbody>
</table>

6. ENERGY CONSUMPTION MODEL

To examine the trade-off between the lifetime and the end-to-end delay (next section) we present an energy consumption model in this section.

6.1 Parameters and symbols

To achieve accurate estimations we considered 27 parameters related to the protocol and hardware, listed in Tables 1 and 2. We explain all model symbols in Table 3. The parameters 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 (e.g. [12]).

6.2 Lifetime and daily energy consumption

We estimate the lifetime (expressed in days) and the daily energy consumption $E_{day}$ of a sensor node as follows (symbols explained in Table 3):

$$\text{Lifetime} = \frac{Q}{E_{day}}$$  \hspace{1cm} (6)

$$E_{day} = E_{txslots} + E_{rxslots} + E_{LDC} + E_{mcu} + E_{selfdischarge}$$  \hspace{1cm} (7)

6.3 Energy consumption of LETED

LETED consumes energy during tx and rx slots. Firstly, we calculate the slot period $T_{slot}$, i.e. the sleep time between consecutive tx slots on each node, needed to fulfill the end-to-end delay requirements $d_{EIE}$ (details in Section 5.2):

$$T_{slot} = d_{EIE} - d_{forwarding} = d_{EIE} - n \cdot (t_{frame} + t_{tx,offset})$$  \hspace{1cm} (8)

We determine the expected frame length of an event notification $t_{frame}$ in time units as:

$$t_{frame} = \frac{\lambda_{frame}}{\vartheta}$$  \hspace{1cm} (9)

$$T_{slot} = d_{EIE} - d_{forwarding} = d_{EIE} - n \cdot (t_{frame} + t_{tx,offset})$$  \hspace{1cm} (8)

Secondly, we determine the number of slots a day $N_{slots}$ of $p$ gathering paths to guarantee the requested end-to-end delay. The considered time interval is a day (24 hours). We consider the worst case, i.e. nodes maintain their schedules all the time just after powering up.

$$N_{slots} = \frac{p \cdot T_{day}}{T_{slot}}$$  \hspace{1cm} (10)

Each node sends data during tx slots only when it detects an event. Thus, the total number of tx slots a day $N_{txslots}$ depends on the event frequency $T_{event}$:

$$N_{txslots} = \frac{N_{slots}}{T_{event}/T_{slot}}$$  \hspace{1cm} (11)

To support end-to-end delay limits, nodes keep listening during each rx slot. Thus, the number of rx slots a day $N_{rxslots}$ equals:

$$N_{rxslots} = N_{slots}$$  \hspace{1cm} (12)
6.4 Energy consumption of DLDC-MAC

We consider DLDC-MAC as the underlying low duty cycle protocol, which consumes energy for sending and receiving beacons:

\[ E_{\text{LDC}} = E_{\text{txbeacon}} + E_{\text{rxbeacon}} \]  \hspace{1cm} (15)

As the energy consumption depends mainly on beacons, firstly we estimate the number of beacons a node sends during a day:

\[ B = \frac{T_{\text{day}}}{T_{\text{beacon}}} \]  \hspace{1cm} (16)

6.4.1 Beacons transmission

\[ E_{\text{txbeacon}} = B \cdot (t_{\text{txbeacon}} \cdot I_x + t_{\text{txbeaconafter}} \cdot I_x + E_{\text{txrxswitch}} + E_{\text{startup}} + E_{\text{shutdown}}) \]  \hspace{1cm} (17)

The energy consumed for sending beacons contains also the energy needed to power up the transceiver and to power it down. In DLDC-MAC each node stays in the listening mode after sending a beacon. We include the energy needed to switch the transceiver to the listening mode as well. The beacon length and the listening time after a beacon are model parameters expressed in bytes. Here, we estimate these parameter in time units:

\[ t_{\text{txbeacon}} = \frac{\lambda_{\text{beacon}}}{\vartheta} \]  \hspace{1cm} (18)

\[ t_{\text{txbeaconafter}} = \frac{\lambda_{\text{beaconafter}}}{\vartheta} \]  \hspace{1cm} (19)

6.4.2 Beacon reception

Beacon energy consumption includes energy needed to get beacons from all neighbours:

\[ E_{\text{rxbeacon}} = B \cdot \text{nbours} \cdot (t_{\text{rxbeacon}} \cdot I_x + E_{\text{startup}} + E_{\text{shutdown}}) \]  \hspace{1cm} (20)

\[ t_{\text{rxbeacon}} = t_{\text{guard}} + t_{\text{txbeacon}} \]  \hspace{1cm} (21)

DLDC-MAC uses guard times based on run-time drift estimation. We estimate the average guard time \( t_{\text{guard}} \) as:

\[ t_{\text{guard}} = \frac{(\kappa_{\text{neg}} + \kappa_{\text{pos}}) \cdot T_{\text{beacon}}}{1 - \text{MBR}} \]  \hspace{1cm} (22)

6.5 Microcontroller energy consumption

Although a microcontroller (\( \mu \text{C} \)) can use several power consumption modes, we consider only two: active (executing code, reading sensors, sending, receiving etc.) and sleep (only a low rate clock is running). In general, \( \mu \text{C} \) consumes energy also when the node sends or receives data. However, we already included this energy when considering data transmission/reception. Thus, here we consider \( \mu \text{C} \) energy (both active and sleep modes) when the transceiver is powered down:

\[ E_{\text{mcu}} = E_{\text{mcuactive}} + E_{\text{mcusleep}} \]  \hspace{1cm} (23)

\[ E_{\text{mcuactive}} = T_{\text{mcuactive}} \cdot I_{\text{mcuactive}} \]  \hspace{1cm} (24)

\[ E_{\text{mcusleep}} = T_{\text{sleep}} \cdot I_{\text{sleep}} \]  \hspace{1cm} (25)

To estimate the total sleep time a day \( T_{\text{sleep}} \), we add up all the active times during the day, that is, when the node is not sleeping. Then, we subtract this time from the “total time in a day”, as follows:

\[ T_{\text{sleep}} = T_{\text{day}} - (T_{\text{txbeacon}} + T_{\text{rxbeacon}} + \sum_{t_{\text{txrxswitch}}} + \sum_{t_{\text{txbeaconafter}}} + \sum_{t_{\text{shutdown}}}) \]  \hspace{1cm} (26)

\[ T_{\text{txbeacon}} = B \cdot (t_{\text{startup}} + t_{\text{txbeacon}} + t_{\text{txrxswitch}} + t_{\text{rxbeaconafter}} + t_{\text{shutdown}}) \]  \hspace{1cm} (27)
the lifetime is. Additionally, the event frequency affects the lifetime as well, as nodes send data mainly upon an event detection. Thus, we considered three scenarios: on average an event occurs every 10 seconds, every minute or every half an hour.

Our results confirmed that larger delays prolong the lifetime. If a sensor network must deliver event notifications within 5 seconds, the nodes work for approx. 10 months for 30-minute event frequency, see Figure 8. If we change the delay requirements to 20 seconds, the nodes prolong the lifetime to almost 2 years! However, we discovered that if the allowed delay is larger than approx. 1 minute the lifetime is no longer significantly prolonged.

As expected, frequent event detections lead to a shorter lifetime. For a desired delay of 30 seconds a node works 2 years when events occur once a minute. The same node works less than 1.5 year when events occur every 10 seconds. However, for delays shorter than approx. 5 seconds, nodes achieve almost the same lifetime regardless the event frequency.

7.2 Lifetime of routers

In general, nodes away from a sink do not participate in data forwarding. They just send their own data and do not play the role of routers. On the contrary, nodes close to the sink forward data of other node very often. Therefore, they maintain schedules of many gathering paths. As a result, they wake up more often and shorten their lifetime. We analyzed how the number of paths a node is in influences the lifetime considerably. For example, with a delay of 10 seconds the node works for 2 years when maintaining two schedules only. Adding another two schedules reduces the lifetime by more than 25% to 1.5 years. However, if the node already maintains many schedules, adding new time slots does not change the lifetime considerably. A node having 15 schedules works for 0.7 years. If the node joins to another two paths, it reduces the lifetime by 10% only.

7.3 Energy consumption

In LETED the number of receive slots depends on the supported end-to-end delay. The smaller the delay is, the more often nodes must wake up to participate in possible data forwarding. Therefore, the delay change influences mainly the energy consumption of receive slots. In Figure 10 we present energy consumption of various node activities and components in order to compare them.

As for required delays shorter than 20 seconds nodes wakes up very frequently, the rx slots are the major energy consumer. For example, with a delay of 10 seconds the node consumes about 53% (1.95 mAh/day) of the total energy consumption for rx slots. Clearly, since nodes wake up less frequently for longer delays, the rx slots consume less energy.

Figure 8: Guaranteed end-to-end delay affects the lifetime: the smaller the delay, the shorter the lifetime.

\[
T_{rbercon} = B \cdot \text{nbours} \cdot (t_{rbercon} + t_{\text{startup}} + t_{\text{shutdown}})
\]

\[
T_{tslots} = N_{tslots} \cdot (t_{\text{frame}} + t_{\text{startup}} + t_{\text{shutdown}})
\]

\[
T_{rslots} = N_{rslots} \cdot (t_{\text{frame}} + t_{\text{guard}} + t_{\text{startup}} + t_{\text{shutdown}})
\]
For instance, with a 23 seconds delay the energy consumption of rx slots equals the battery self-discharge. Moreover, with a delay of 2 minutes the rx slot energy is less than 10% of the total energy consumption.

The examined node consumes the same amount of energy for sending data (tx slots) regardless of the end-to-end delay, see Figure 10. It is due to the fact that the node sends data only when an event occurs and not at each tx slot, see Eq. 11. As the event frequency is the same for all examined delays in this example, the energy consumption of tx slots remains constant.

8. CONCLUSION

In this work we introduced a solution to limiting end-to-end delays (LETED) in multi-hop sensor networks. Nodes using LETED limit end-to-end delays considerably and still achieve a long lifetime. Moreover, LETED adapts to changing clock drift and thus can work even in outdoor scenarios, i.e. with varying temperature.

We presented a model for analyzing the end-to-end delays and lifetime in multi-hop sensor networks. Although the model considers mainly LETED coupled with our low duty cycle protocol (DLDC-MAC), it can be used with other low duty cycle solutions as well. For example, IEEE 802.15.4 [7] MAC protocol in beacon mode can be immediately evaluated with our model. For other protocols the model may need some adaptations.

In this paper we evaluated the trade-off between the end-to-end delay and the lifetime. Our key findings are:

1. A multi-hop sensor network that uses only off-the-shelf sensor nodes (e.g. with 2xAA batteries only) achieves good results in terms of delay and lifetime. For example, such a network can guarantee an end-to-end delay of no more than 5 seconds and still achieves a lifetime of 8 months.

2. Increasing the end-to-end delay above a certain limit does no longer improve the lifetime. In our example this limit was about a minute.

3. Our experiment revealed that a temperature change affects only average drift and not clock drift jitters. In other words, the clock drifts around the average (estimated from last 2 drift samples) within an almost constant window - around 10 ticks in our experiment - regardless of the changing temperature. Therefore, if a node calculates the drift average repeatedly, it can shorten guard times considerably and reduce idle listening.

Our analytical model can be used to engineer the end-to-end delay limits and lifetime of wireless sensor networks in very early stages. In our future work we will integrate means to express transmission failures in order to retrieve results even more close to the real world behaviour when estimating delay times and lifetime of nodes.

Acknowledgment

The research leading to these results has received funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 225186 and from the German Ministry of Education and Research under grant n° 03IP601.

9. REFERENCES