Energy-Efficient Means to Support Short End-to-End Delays in Wireless Sensor Networks

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

genehmigte Dissertation

vorgelegt von

M.Sc.

Marcin Brzozowski

geboren am 28. Februar 1979 in Ostrów Wielkopolski (Polen)

Gutachter: Prof. Dr. rer. nat. Peter Langendörfer
Gutachter: Prof. Dr.-Ing. Jörg Nolte
Gutachter: Prof. Dr. Evgeny Osipov

Tag der mündlichen Prüfung: 4. Juli 2012

Abstract

This work addresses tough challenges of sensor network applications with Quality of Service requirements. That is, nodes must work with batteries for a long time, support short end-to-end delays and robust communication in multi-hop networks. It starts with presenting 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 for communication. Although such protocols offer short end-to-end delays, they still suffer from long idle listening and shortened lifetimes. The main reasons are the long time needed to detect an idle channel and inefficient ways of dealing with clock drift. This work introduces novel solutions to these problems, mainly at Layer 2 of the OSI model, that significantly reduce idle listening. First, nodes predict future drift and reduce the time needed to compensate clock uncertainty among neighbors. Second, they quickly detect an idle channel and power down the transceiver. In some scenarios, nodes work 30% longer owing to these solutions.

To tackle problems with unreliable wireless links, sensor nodes may apply various solutions at Layer 2. For example, with Automatic Repeat reQuest (ARQ) protocol they send retries on frame losses, resulting in extra energy consumption. This work examines the impact of ARQ on the lifetime and on the reception rate. Several indoor and outdoor experiments showed that with only 1-2 retries nodes can handle many communication problems. Besides, owing to the idle-listening reduction, mentioned previously, ARQ shortens the lifetime by 10% only.

Although this work addresses particular applications, the solutions presented here can be used in other scenarios and with different protocols. For instance, the energy-efficient drift compensation approach can be directly used in any schedule-based MAC protocols, like the one based on the IEEE 802.15.4 standard. Besides, any protocol can benefit from the solution to the idle-listening reduction based on the early detection of idle channel. Finally, owing to the analytical model that estimates the lifetime of nodes, researches and developers can early evaluate MAC protocols running on various hardware platforms.

Zusammenfassung

Diese Arbeit beschäftigt sich mit den Herausforderungen von Sensornetzanwendungen mit Quality-of-Service-Anforderungen. Die Sensorknoten in einer solchen Anwendung müssen über einen langen Zeitraum mit Batterien auskommen und gleichzeitig kurze Ende-zu-Ende-Verzögerungen und zuverlässigen Datenversand in einem Multi-Hop-Netzwerk unterstützen.

Zunächst werden bisherige Forschungsarbeiten zu diesem Thema vorgestellt. Viele Medienzugriffsprotokolle (MAC) lassen die Knoten die meiste Zeit "schlafen", um Energie zu sparen, und synchronisieren die Wachzeiten, um Kommunikation zwischen den Knoten zu ermöglichen. Solche Protokolle unterstützen zwar kurze Ende-zu-Ende-Verzögerungen, jedoch wird aufgrund von sogenanntem Idle Listening (Abhören des Funkkanals und Warten auf Nachrichten) nur eine kurze Lebensdauer erreicht. Dies liegt zum einen daran, dass zuviel Zeit benötigt wird um festzustellen, dass das Medium inaktiv ist und zum anderen an ineffizienten Verfahren für die Kompensation der Uhrendrift. Diese Arbeit stellt neue Lösungen für diese Probleme vor, die das Idle Listening erheblich reduzieren und hauptsächlich auf der Schicht 2 des OSI-Modells implementiert werden. Erstens berechnen die Knoten die zukünftige Uhrendrift ihrer Nachbarn, wodurch Unsicherheiten bzgl. der Drift beseitigt werden. Zweitens wird die nötige Zeit für die Erkennung eines inaktiven Mediums und dem Abschalten des Transceivers verringert. Die Lebensdauer der Knoten kann damit um bis zu 30% gesteigert werden.

Es gibt unterschiedliche Ansätze - implementiert in der OSI-Schicht 2 - um mit der Unzuverlässigkeit der drahtlosen Kommunikation umgehen. Bei Automatic Repeat reQuest (ARQ) z.B. werden Pakete bei Verlust noch einmal gesendet. Dies erhöht jedoch den Energieverbrauch. Die Auswirkungen von ARQ auf die Lebensdauer und die Empfangsrate wird daher in dieser Arbeit untersucht. Experimente haben gezeigt, dass schon ein bis zwei Wiederholungen ausreichen, um die meisten Kommunikationsprobleme zu beseitigen. Aufgrund der Verkürzung des Idle Listenings durch die oben genannten Lösungen verkürzt ARQ die Lebensdauer nur um 10%.

Obwohl diese Arbeit nur bestimmte Anwendungen betrachtet, können die hier vorgestellten Lösungen auch in anderen Szenarieren und auf andere Protokolle angewandt werden. Zum Beispiel kann das energieeffiziente Verfahren zur Kompen-

sation der Uhrendrift direkt in vielen MAC-Protokollen verwendet werden, z.B. im IEEE 802.15.4 MAC. Zudem kann jedes Protokoll von der Lösung für die schnelle Erkennung eines inaktiven Mediums und der daraus resultierenden Reduktion des Idle Listenings profitieren. Schließlich können Forscher und Entwickler das vorgestellte analytische Modell nutzen, um die Lebensdauer eines Sensornetzes beim Einsatz verschiedener MAC-Protokolle zu berechnen.

Contents

1 Introduction

1.1 Challenges of Long-Living Sensor Networks

Recent development in the electronic industry, especially miniaturization, allowed the use of tiny wireless devices with sensing abilities, referred to as sensor nodes. Sensor nodes are usually the size of a matchbox, work with batteries and send data wirelessly. Since they work for several months or years, and do not need wires at all, sensor nodes provide a new set of applications. Nodes usually form a wireless network, monitor a specific area by reading sensors and send sensor readings to a sink. Ref. [22] lists several sensor network applications, for example:

- *Disaster relief applications*, like wildfire detection. Sensor nodes equipped with thermometers produce a "temperature map" of a forest. Then, areas with a high temperature are accessed from outside in advance to prevent fire.

- *Intelligent buildings.* Sensor nodes can provide real-time, high-resolution monitoring of temperature, airflow or humidity. In this way, they can increase the comfort level and reduce the energy consumption. According to [39] such a technology could reduce energy consumption by two quadrillion British Thermal Units in the US alone, that is, $55 billion a year and 35 million metric tons of reduced carbon emissions. Besides, sensor nodes can check mechanical stress levels of buildings in seismically active zones. By doing so, they provide information about building condition after an earthquake.

- *Precision agriculture.* Sensor nodes placed in soil allow precise irrigation and fertilizing. Also, attaching a sensor node to an animal, for example, to a pig or to a cow, allows controlling its health status and raises alarms early enough.

- *Logistics.* Individual parcels with sensors allow an easy tracking during transport or in stores

- *Critical Infrastructure Protection* (CIP). 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. When 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.

This work addresses mainly 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, mainly short end-to-end delays and reliable data transfer. These challenges are introduced briefly in the following.

Long Lifetime of Sensor Nodes

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 is not feasible. Therefore, they are powered by batteries, which should provide energy for a long time. To ensure long lifetimes, nodes apply protocols that support a low duty cycle (LDC). Such protocols keep nodes mostly in the sleep state. As the current consumption in the sleep state is smaller by three orders of magnitude than in the active state, sensor nodes increase the lifetime significantly. For example, Tmote Sky [33] 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.

Short End-to-End Delays

Nodes with LDC protocols wake-up rarely to check for potential transmissions. Thus, on event detection the source node cannot send data immediately to the next node, but waits until it becomes active. In multi-hop networks, each node waits until the next node wakes up before sending data towards the sink. Apparently, it can result in significant end-to-end delays, and the sink may receive event notices too late. Therefore, nodes should wake up more often, but this increases the duty cycle and shortens the lifetime. Clearly, there is a tradeoff between these two goals, that is, a long lifetime and short end-to-end delays.

Packet Losses

Since wireless communication is prone to errors [40], sensor nodes often suffer from packet losses. For example, experiments introduced in ref [54] show that some wireless links suffered from a packet error rate of 50%. Thus, in CIP applications the sink may miss some event notices, e.g., gas leakage detection, and can fail to prevent the danger.

Obviously, many solutions tackle the problems of packet losses in wireless networks. However, they result in extra energy consumption, for example, because of transmissions of additional frames. On one hand, nodes should apply such solutions to recover from packet losses. On the other hand, the use of these remedies must be limited to achieve long lifetimes.

1.2 Solutions

Reduction of Idle Listening

To guarantee two opposing goals, that is, short end-to-end delays and long lifetimes, several approaches (DMAC [29] and Q-MAC [50]) 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 supports short end-to-end delays, it suffers from the idle-listening problem. That is, nodes keep transceivers in the receive state, consume energy, but do not get any frames. The main reasons for idle listening of the staggered schedule are the following:

- Clock drift
  - Because of clock drift, a sensor node may wake up too soon to receive a message.
  - To solve this problem, sensor nodes wake up earlier by a guard time.
  - Common solutions to guard times consider worst drift. That is, based on the crystal oscillator parameter, neighbors estimate the worst-case time their clocks may drift away over a sleep period.
  - Run-time drift is a few times smaller than the worst case and such solutions cause unnecessary long idle listening.
- Useless wake-ups
  - To support short end-to-end delays nodes wake up often.
  - Since events seldom occur in CIP applications, most wake-up periods are useless, that is, nodes do not receive data but only waste energy.

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. 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.

2. 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, prolonging the lifetime by 30% and even more.

Link-Layer Means to Packet Loss Problem

The major groups of means that deal with unreliable communication are *hop-by-hop* and *end-to-end*. The former recovers from packet losses at each node on the multi-hop paths. For example, each intermediate node sends retries on a frame loss. The latter is applied only to the source and the destination and not to intermediate nodes. That is, only the source node repeats transmissions and expects a response from the destination.

Because of unreliable wireless communication, sensor nodes should primarily recover from packet losses on the hop-by-hop basis. Even small-scale experiments show that hop-by-hop solutions outperform end-to-end means [2]. According to ref. [22], link layer retransmissions (i.e., hop-by-hop) keep energy costs within reasonable bounds whereas costs for end-to-end approach explode from a certain threshold of the bit error rate.

The main hop-by-hop means are ARQ (Automatic Repeat reQuest) and CSMA/CA (Carrier Sense Multiple Access With Collision Avoidance). The former can recover from all types of transmission errors, as senders repeat transmissions on frame losses. The latter deals only with the collision problem by postponing transmissions on a busy channel. Since both solutions need extra energy, mainly because of longer idle-listening times and more transmissions, they affect the lifetime of sensor nodes. This work examines the tradeoff between a long lifetime of nodes and reliable data transfer, provided by ARQ and CSMA/CA, which are applied separately or together. Real-world experiments presented in this work show that nodes ensure a reliable communication with 1-2 ARQ retries. By doing so, they improve the reception rate by 20% and shorten the lifetime by 10% only. However, such a minor impact on the lifetime stems from the reduction of idle listening mentioned previously. As expected, CSMA/CA does not improve the connection quality in applications with a low duty cycle, as the collision risk is low. Besides, it significantly reduces the lifetime of nodes and therefore should not be applied to CIP scenarios.

1.3 Structure of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 presents the architecture of sensor networks and main Medium Access Control (MAC) protocols that support the long lifetime of nodes, owing to low duty cycles.

In Chapter 3 the main challenges addressed in this work are explained, i.e., ensuring long lifetime, short end-to-end delays and reliable communication. This chapter also introduces potential solutions and argues the selection of approaches considered in this work.

Chapter 4 presents Distributed Low Duty Cycle MAC (DLDC-MAC) protocol for low power. The protocol provides long lifetimes of nodes and serves as a basis for LETED. This chapter also evaluates DLDC-MAC in a real-world experiment.

Chapter 5 introduces the LETED approach, which supports short end-to-end delays in multi-hop networks. Besides, this chapter explains and evaluates ILA (Idle Listening Avoidance), i.e., the main solution that reduces idle listening of the staggered schedule. Finally, this chapter presents results of small- and large-scale simulations with nodes based on LETED and DLDC-MAC protocols.

Chapter 6 presents a drift experiment with sensor nodes places indoors and outdoors. Based on the empirical results, it provides energy-efficient solutions to the drift problem of schedule-based MAC protocols. Furthermore, it estimates potential gains in lifetime provided by these solutions.

Chapter 7 provides a model that estimates the lifetime and energy consumption of sensor nodes. The model is used to evaluate the solutions of this work against other state-of-the-art protocols. The evaluation results are discussed in this chapter as well.

Chapter 8 presents the results of indoor and outdoor experiments with ARQ and CSMA/CA approaches. Besides, it examines the impact of both solutions on the lifetime, based on the model introduced in the previous chapter.

Chapter 9 summarizes the achievements of this thesis and presents intended future research efforts.

2 Sensor Networks

This chapter introduces the architecture of sensor nodes and the main challenges of sensor networks. Main problems of such networks stem from a limited power source, as nodes usually work only with batteries that cannot be easily recharged. Thus, many protocols, presented later in this chapter, support low duty cycles and save energy in this way. Since such protocols keep nodes mostly in the sleep state, they may not support certain features of the Quality of Service (QoS), like short end-to-end delays in multi-hop networks. The main QoS aspects are briefly introduced at the end of this chapter.

2.1 Overview

2.1.1 Single Node

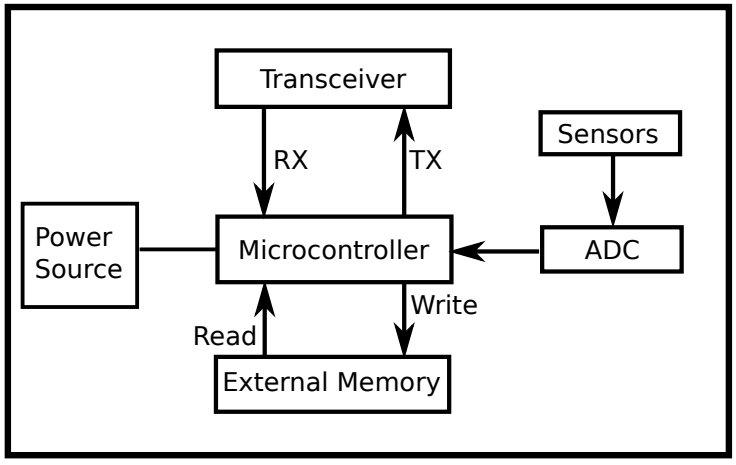


Figure 2.1.1: Sensor node architecture; External memory is optional; ADC means analog-to-digital converter

Figure 2.1.1 depicts the main parts of sensor nodes. A central processing unit (CPU) is the heart of sensor nodes. To save energy, however, sensor nodes use CPUs not as powerful as Personal Computers (PCs). Usually, nodes have a 16-bit CPU with a frequency of a few MHz, referred to as a microcontroller (µC or MCU). A

typical µC draws current of a few mA while running and 1000x less in the sleep state. Table 2.1 compares the energy consumed by PC processors and a microcontroller used in sensor nodes. For instance, in the active state, the microcontroller needs about 5 orders of magnitude less energy than the Intel Core i7 processor.

Table 2.1: Energy consumption and achieved lifetime of desktop processors (Intel Core i7, Intel 386) and a microcontroller used in sensor networks (MSP430) with standard rechargeable 2x AA batteries (1.2V and 2700 mAh each)

Processor	Energy consumption	Lifetime
Intel Core i7	130 W	4 min
Intel 386	2 W	4 hours
MSP430 (active)	0.015 W	22.5 days
MSP430 (sleep)	0.000005 W	184 years

Table 2.2: Current consumption of the ChipCon CC2420 transceiver used in sensor nodes

Mode	Current consumption
TX with 0 dBm	17.4 mA
TX with -25 dBm	8.5 mA
RX	19.7 mA
Sleep (Power Down)	1 µA

Table 2.3: Current consumption of WLAN transceivers; Orinoco 11b is a PCMCIA card used in laptops; OWLAN211g is a transceiver designed for low power

	Orinoco 11b	OWLAN211g
TX	285 mA	170 mA
RX	185 mA	170 mA
Sleep	9 mA	0.8 mA

Sensor nodes send and receive data wirelessly. Common transceivers used in sensor nodes have a low data rate and consume little energy. For example, ChipCon CC2420 [47] transceiver, based on the IEEE 802.15.4 [20] standard, supports data rates up 250 kbit/s and draws 20 mA current when sending or receiving (see Table 2.2). Besides, to preserve energy, sensor nodes keep mainly the radio in the sleep state. Therefore, transceivers must provide an energy-efficient sleep state as well. On the contrary, a typical WLAN transceiver supports data rate of a dozen of

Mbit/s, but draws much more current than radios of sensor nodes. Table 2.3 presents current consumption of two WLAN transceivers. The first module, Orinoco 11b PC Card[38], is a standard wireless PCMCIA card. The second, OWLAN211g[11], is a WLAN transceiver designed for low power. For example, OWLAN211g draws almost 8x more current than CC2420 when sending data and 800x more in the sleep (idle) state.

Sensor nodes usually use batteries as a power supply. In many applications, batteries cannot be easily replaced or recharged, but nodes must work several months or longer. In some applications, nodes can get part of its energy from the environment, for example, with solar cells. Finally, in other applications, especially indoors, operators can easily replace or recharge batteries. In addition, some nodes can be mains-powered. This work considers the worst case, that is, nodes use non-rechargeable batteries only.

Nodes monitor the environment with various sensors, for example, temperature, humidity, noise level, vibrations. Such sensors are usually analog devices and provide various voltage levels for different readings. An analog-to-digital converter (ADC) translates an analog voltage level to a digital value, expected by the µC.

A basic sensor node does not need any external memory, as presented in Figure 2.1.1, since the microcontroller has on-chip memory available. For example, MSP430 can have up to 256 kB of Flash memory and 16 kB of volatile memory. However, in some applications, an external memory is necessary, for example, to store important data in nonvolatile memory.

2.1.2 Sensor Network

Sensor nodes usually monitor a specific area and send sensor readings to a sink. As the area is typically larger than the radio range, nodes transmit data over multi-hop paths. Sensor nodes forward data in the store-and-forward fashion, that is, they receive whole frames, store them in the RX buffer and send to the next node.

Even when the radio range covers the area, nodes may benefit from multi-hop networks by using a smaller TX power. In this case, they decrease the collision risk because of a smaller radio interference range. Besides, the loss of the signal strength is proportional to at least the square of the sender-to-receiver distance. Therefore, nodes on a multi-hop path may consume in total less energy than a direct transmission with full power. However, the authors in [32] contradict this statement and show that a direct transmission of a Bluetooth transceiver (2.4 GHz frequency) is mostly more energy efficient than sending data in multi-hop networks.

Since sensor nodes form a multi-hop network, they must find a route towards the

sink. Although the topology of sensor networks resembles a centralized approach, with a powerful sink available, the nodes usually work in a decentralized, ad-hoc fashion. Therefore, even when some nodes break down, the network adapts quickly to the new topology.

Sensor networks resemble ad-hoc networks, as nodes send data wirelessly and organize themselves in a distributed way. The main differences between these two network types are the following:

- *Scalability*
  The number of nodes is usually significantly larger in sensor networks. For example, some works introduce applications with sensor networks of hundreds or thousands nodes [15]. On the contrary, ad-hoc networks commonly consider dozens of wireless nodes.

- *Limited energy*
  Sensor nodes have a limited power source available, for example only two AA batteries on Tmote Sky node, but they must work for a long time. On the contrary, ad-hoc nodes do not usually suffer from such a limited power supply. Besides, users can easily recharge ad-hoc devices or connect them to the electricity.

- *Limited memory*
  Despite of a limited memory available on sensor nodes, sometimes less than 64 kB, the software with all protocols must fit into it. Therefore, the software includes only necessary parts. Ad-hoc nodes, on the contrary, have much more memory available, e.g., a few GB in case of laptops.

- *Failures*
  Sensor nodes are prone to failures, especially when working in harsh environments, for example, in hot or wet conditions. In addition, if sensor nodes stop working, it is often impossible to reset them manually. Ad hoc devices, like laptops or PDAs, work more reliably than nodes placed outdoors. Even when there is a problem with ad-hoc nodes, users can fix it quickly, e.g., start nodes again, etc.

Although sensor nodes differ from ad-hoc networks, they can apply some ad-hoc protocols. Sometimes the protocols need just small adaptations to consider the constraints of sensor nodes. For instance, many ad-hoc routing protocols, like Dynamic Source Routing (DSR) [21] or Ad hoc On-Demand Distance Vector Routing (AODV) [35], work well in sensor network applications.

2.1.3 Wireless Communication

Sensor nodes send and receive data wirelessly and therefore suffer from common wireless problems introduced in the following.

Collisions

Should two or more nodes send data at the same time, see Figure 2.1.2, the electromagnetic waves interfere, and the receiver does not get any frame correctly. Clearly, the collision risk depends on the data rate in the network. That is, the higher the data rate is, the higher is the collision risk. Since sensor nodes rarely send data, the collision risk should be low. However, even in sensor networks with a low data rate, collisions may frequently occur on certain periods. For example, in several applications nodes monitor the same area. If they detect an event, like a gas leakage, all nodes try to notify the sink about it and send out immediately frames. Obviously, in this case, the collision risk is high, although the data rate is low on average.

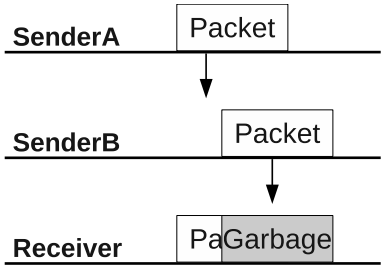


Figure 2.1.2: Collision problem: two nodes send frames at the same time, the frames "collide" at the receiver, and it cannot receive anything

Idle listening

Figure 2.1.3 depicts the idle-listening problem of wireless communication. The problem arises when a node keeps the radio in the RX state but no frame arrives. For example, as nodes cannot predict when an event, like a gas leakage, occurs, they expect to receive data from neighbors at any time. In order not to miss data, nodes keep the transceiver often in the RX state and cause excessive idle listening. To limit idle listening sensor nodes apply LDC (low duty cycle) protocols, introduced in the next paragraph, which keep the transceiver mainly powered down.

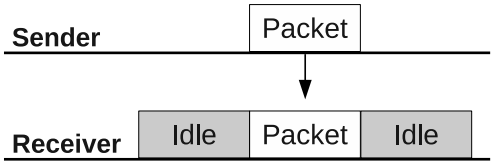


Figure 2.1.3: Idle Listening: a receiver keeps its radio in the receive state for a time longer than it is needed to receive a frame

Overhearing

When a node receives a frame addressed to another node (see Figure 2.1.4), it wastes the energy, as it discards the packet anyway. To save the energy the node should not receive such packets at all. Because of a broadcast feature of wireless communication, however, all nodes in the transmission range of a source receive frames and suffer from the overhearing problem.

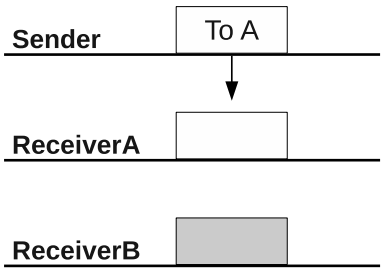


Figure 2.1.4: Overhearing: ReceiverB gets a frame addressed to ReceiverA because of a broadcast feature of wireless communication

Protocol overhead

Each communication layer, like Network Layer or Data Link Layer, adds its header to the frames before transmissions. As a result, each node sends application data with protocol headers (see Figure 2.1.5). For example, the MAC header of IEEE 802.15.4 can occupy about 10% of the frame or more. Obviously, sending protocol headers results in extra TX energy consumption. Besides, large protocol headers need frame fragmentation, as application data does not fit into the frame. First, it leads to even more protocol overhead, since every fragmented frame needs extra protocol headers. Second, it makes the protocols complicate and needs extra memory for the implementation.

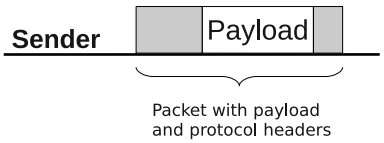


Figure 2.1.5: Protocol overhead: nodes send not only the payload, i.e. application data, but also control data of underlying protocols

MAC protocols

To solve the problems with wireless communication, nodes apply Medium Access Control (MAC) protocols, a sublayer of the Data Link Layer according to the 7-layer OSI (Open Systems Interconnection) model. MAC protocols grant or deny an access to the wireless channel. That is, before a transmission each node "asks" its MAC protocol, whether it is allowed to send a frame. For instance, before sending data, nodes check whether the channel is idle by performing so called channel sense. Should they detect any transmissions, they wait a certain time in order not to interfere and check the channel again.

As there are many different sensor network applications, there is no MAC protocol "one-size-fits-all". For instance, in indoor scenarios, nodes get the energy from power outlets and do not suffer from energy problems, like idle listening. Because of interference with other wireless networks inside a building, however, they must deal with a high collision risk. On the contrary, the main concern of nodes that work outdoors is the limited energy. Thus, they use different MAC approaches, designed mainly for low power, than nodes placed indoors.

2.2 Low Duty Cycle (LDC) and Rendezvous

The main concern of many sensor network applications is a limited power source. An off-the-shelf sensor node Tmote Sky [33] with standard two AA batteries works only few days, if it keeps the transceiver and the microcontroller permanently powered up. However, nodes must provide longer lifetimes, several months or years. To achieve such long lifetimes, sensor nodes apply 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 to receive data. However, to send and to receive data, nodes must be awake at the same time, referred to as *rendezvous* [26]. Obviously, each node on a multi-hop route needs *rendezvous* with the next node towards the destination.

Figure 2.2.1 introduces the idea of *rendezvous* in multi-hop networks. In this case, nodes mostly sleep to save energy. When the source detects an event, e.g., a gas

leakage, it must inform the sink about it. However, as the next node towards the sink (node A) still sleeps, it cannot receive data from the source just after event detection. Both nodes must synchronize their wake-up times: *rendezvous*. Then the source sends a frame to node A. Immediately after the frame reception, node A tries to send it to node B, but it needs to wait for *rendezvous*. The process continues until the last node delivers data to the sink.

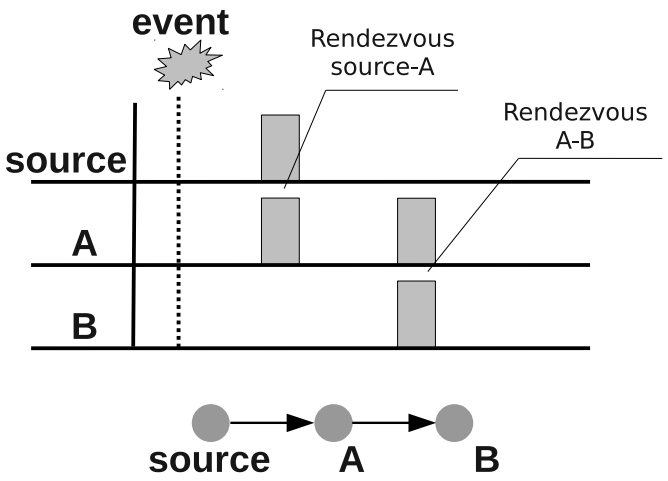


Figure 2.2.1: Rendezvous in multi-hop networks: each node pair (sender and receiver) synchronize wake-up times

The ideal solution does not cause any *rendezvous* overhead. In this case, nodes on the path to the sink wake up just in time to receive data from the previous node. However, as nodes cannot predict when events occur, they cannot just wake up at the right time to get frames. Thus, nodes have to apply underlying *rendezvous* solutions, which synchronize wake-up time between senders and receivers. Ref. [26] groups such solutions into three categories:

- *asynchronous*
  Nodes can wake up other nodes with a dedicated hardware, for example, wakeup radios [55]. When a node wants to send data to its neighbor, it wakes up the neighbor and 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 wakeup radio. For example, nodes may periodically listen for potential transmissions and stay awake, if they detect a transmission wish.

Table 2.4: Low Duty Cycle protocols for sensor networks and types of rendezvous

Name	Rendezvous	
	Synchronous	Pseudo-Asynchronous
S-MAC	x	
T-MAC	x	
IEEE 802.15.4 with beacons	x	
DMAC	x	
FPS	x	
Twinkle	x	
Dozer	x	
STEM		x
Preamble Sampling		x
B-MAC		x
WiseMAC		x
TICER / RICER		x
Koala		x

- *synchronous*
  Nodes agree on specific communication time slots: they send and receive data only during such slots. Senders and receivers usually arrange a common schedule and wake up at the same time to communicate.

Table 2.4 lists several MAC protocols with LDC support, which are presented shortly in the following.

S-MAC

Sensor-MAC (S-MAC) [53] alternates between two states: listen (active) and sleep (see Figure 2.2.2), like usual low duty cycle protocols. S-MAC provides a wake-up schedule, that is, nodes are in the active state at the same time and therefore can communicate. Figure 2.2.3 (top) presents the active period of a receiver; it consists of two phases: SYNC and RTS. In the SYNC phase, nodes send and receive synchronization frames to deal with longtime clock drift. In this case, *Sender1* and *Sender3* send such frames to the receiver, as they intend to transmit data afterwards. On

receiving SYNC frames, nodes adapt their timers and compensate clock drift. According to [53], S-MAC does not need a tight synchronization, since the SYNC phase is longer than typical clock drift. For example, the listening period of 0.5 second, mentioned in ref. [53], is more than 10^6 times longer than typical clock drift rates.

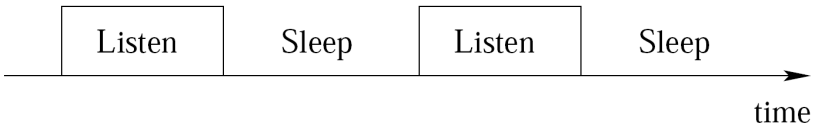


Figure 2.2.2: S-MAC alternates between two states, Listen (nodes send data) and Sleep (nodes save energy), like other protocols that support low duty cycles

During the RTS phase (see Figure 2.2.3) nodes send data with the approach (Request to Send / Clear to Send) based on MACAW [1]. In short, a node sends a RTS frame to the destination, and it replies with a CTS frame. In this way, the node gains permission to send frames. Other nodes receive the CTS frame as well and postpone transmissions.

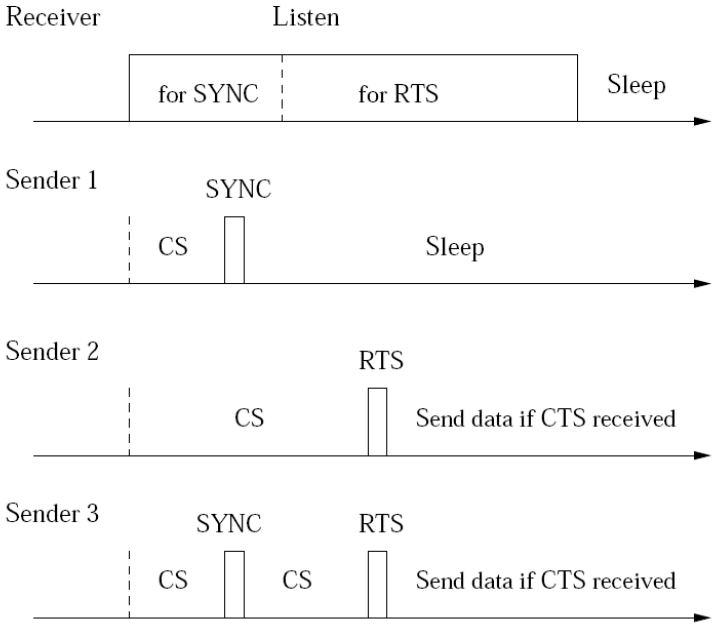


Figure 2.2.3: Listen state of S-MAC protocol consist of two phases SYNC and RTS. In SYNC nodes synchronize their clocks and compensate drift. In RTS they send data with the RTS/CTS approach

T-MAC

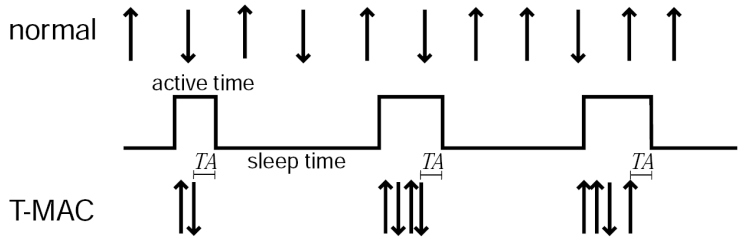


Figure 2.2.4: T-MAC adapts the active time according to the traffic load; it shortens active periods when the traffic is low and prolongs them on a heavy traffic load

Timeout-MAC (T-MAC) [48] attacks the idle-listening problem of S-MAC [53] and adapts the active period length according to the traffic load. Figure 2.2.4 shows the basic scheme of the T-MAC. Each node alternates between active and sleep states, like in S-MAC, and applies the RTS/CTS approach. However, in T-MAC nodes finish the active period when no event occurs for a time TA (see Figure 2.2.4). Thus, T-MAC adapts its duty cycle according to the network load, i.e., nodes shorten active periods on idle channel and prolong them under a heavy traffic load.

IEEE 802.15.4 MAC

The standard IEEE 802.15.4[20] defines the physical and the Medium Access Control (MAC) layers, but this work considers MAC only.

There are two device types in IEEE 802.15.4: full-function device (FFD) and reduced-function device (RFD). The latter ones do not provide several features, e.g., RFDs cannot serve as network coordinators. Besides, RFDs cannot send data direct to another RFD. They communicate only with the coordinator, which is always a FFD. Thus, a network of RFDs forms a star topology with a FFD in the center (see Figure 2.2.5). On the contrary, FFDs can send data to any other device and therefore can form other topologies, like peer-to-peer (see Figure 2.2.5).

IEEE 802.15.4 networks work in two modes: with and without superframes. In the first case, the network coordinator periodically sends beacons. Nodes use beacons mainly to synchronize wake-up times and to define the following timeslots. Figure 2.2.6 depicts the superframe. To save energy, nodes can switch off their transceivers between the successive beacons.

In networks without superframes, coordinators are always powered on, and nodes use CSMA/CA (Carrier Sense Multiple Access With Collision Avoidance) to avoid

collisions. An RFD receives messages from the coordinator by occasionally polling it.

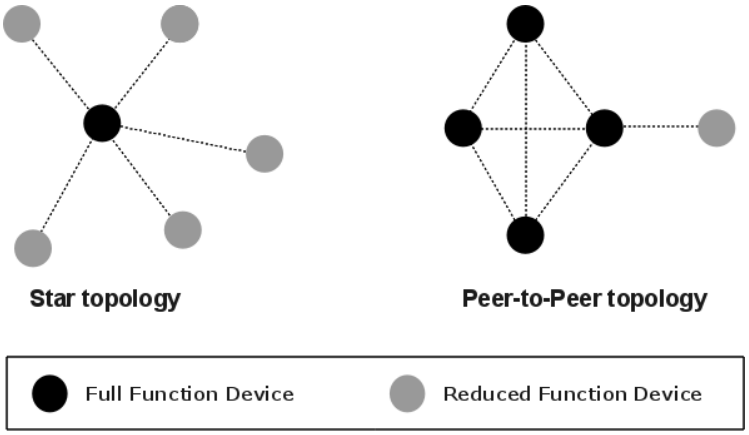


Figure 2.2.5: IEEE 802.15.4 topology: start and peer-to-peer
Reduced Function Devices send data to a network coordinator (Full Function Device) only

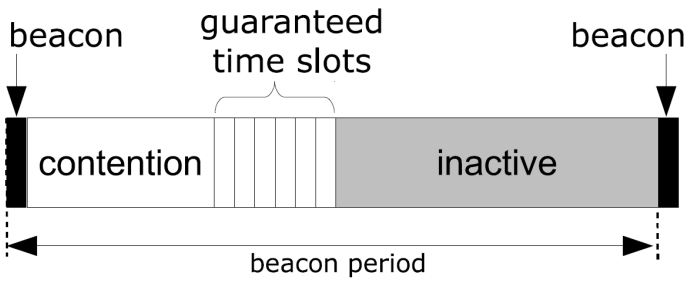


Figure 2.2.6: Superframe of IEEE 802.15.4 MAC; on beacon reception, nodes access the channel in the contention mode (CSMA/CA) followed by the contention-free period with timeslots; after that, nodes power down the radio and sleep till the next beacon

DMAC

DMAC [29] addresses the problem of low-latency data gathering in duty-cycled sensor networks. It considers a network with a tree topology, that is, the nodes form a tree rooted at the sink (see Figure 2.2.7). To preserve energy, nodes mostly sleep and wake up for a short time according to the schedule.

Each node alternates among three states: sleep, send (TX), and receive (RX). In the RX state, nodes expect packets and send corresponding ACKs to the sender. In

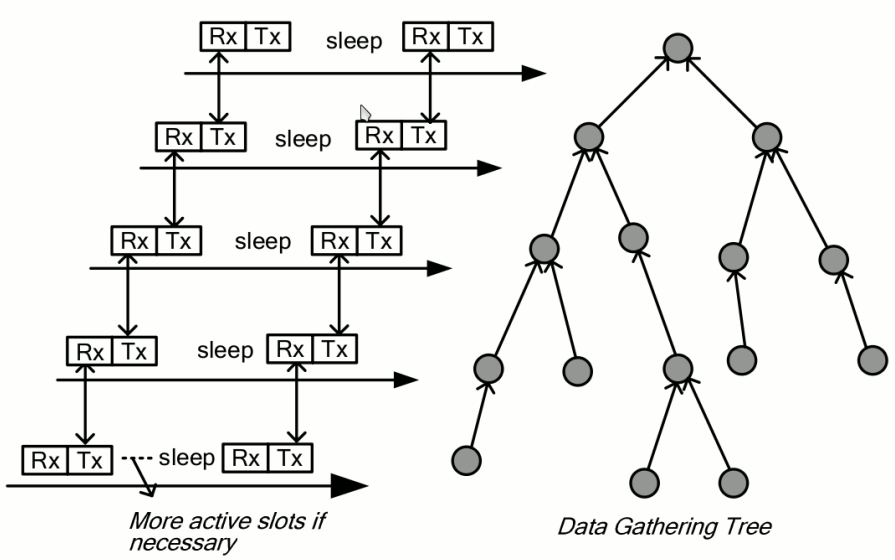


Figure 2.2.7: DMAC creates the staggered wake-up schedule; each intermediate node forwards data just after reception, and in this way DMAC shortens end-to-end delays

the TX state, nodes try to send packets to the next hop and receive ACKs. In the sleep state, nodes power down the radio to save energy

Figure 2.2.7 depicts the tree topology and the corresponding wake-up schedule, referred to as the staggered active/sleep schedule. That is, TX slots to the next node follow immediately RX slots from the previous hop. With such a schedule, nodes receive frames and forward them almost immediately to the next node. Clearly, the TX slot of the previous node and the RX slot of the next node must overlap to allow *rendezvous*. Owing to the fast data forwarding, DMAC reduces end-to-end delays.

Should nodes have multiple packets to send, they increase the duty cycle and prolong TX slots. Next nodes towards the sink increase their duty cycle too. In this way, DMAC adapts the duty cycle to the network load.

Clearly, DMAC must align wake-up times between neighbors to compensate clock drift. However, the wake-up slot alignment is not a part of DMAC, since it applies an existing time synchronization protocol.

Flexible Power Scheduling

Flexible power scheduling [19] (FPS) introduces two-level scheduling: coarse-grain and fine-grain. The former is applied to the network layer to plan radio on/off times. The latter controls the channel access on the MAC layer.

FPS creates a schedule tree rooted at the sink according to nodes demands, i.e.,

data waiting for transmission. FPS spreads the schedule without a global control and without a network-wide initialization phase.

Although the coarse-grain schedule reduces contention, it cannot guarantee the medium is contention free. Thus, FPS needs an underlying MAC layer to handle channel access.

Twinkle

Twinkle [18] improves the Flexible Power Scheduling [19] and introduces partial flows, i.e., they end at any node and not at the root only as in FPS. Thus, Twinkle is not limited to the tree topology and supports a wider range of applications.

Dozer

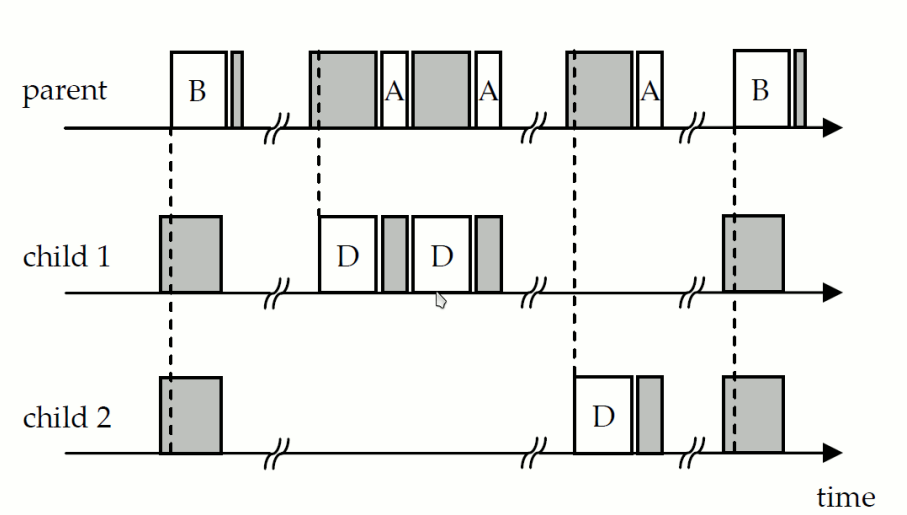


Figure 2.2.8: Dozer: parents send periodically beacons (B) to children in order to synchronize clocks; parents fix upload slots from children; in this example, the parent receives messages (D) from two children and sends acknowledgments (A)

Dozer [9], presented in Figure 2.2.8, is a low duty cycle protocol aimed at the communication from many sensor nodes to a sink. Dozer maintains a forwarding tree rooted at the sink. Each node plays two roles: a parent and a child. Parents periodically send beacons, and children wake up to receive beacons. On beacon reception, nodes synchronize wake-up with the parent and estimate the next beacon time. To compensate clock drift, child nodes wake up earlier by a guard time than the expected beacon time. To estimate guard times, nodes consider worst-case drift.

Beacons are mainly used for two purposes. They compensate clock drift and allow new children to join the network. In the latter case, after sending a beacon, each parent listens shortly for connection requests from new children.

Collisions may arise between nodes, if their schedules overlap. Therefore, parents extend the beacon period by a randomly chosen amount of time. The children must be aware of that random time, or they miss beacons otherwise. As the seed value of the random number generator is included in each beacon, parents and children estimate the same random number.

STEM

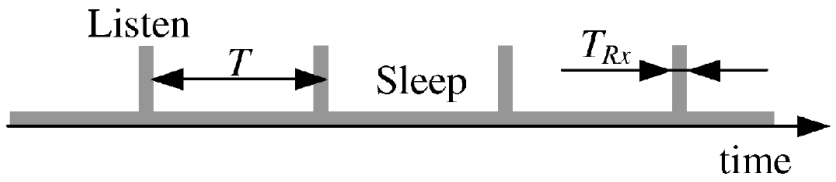


Figure 2.2.9: Receivers with STEM wake up every T period to get wake-up signals, beacons or busy tone

In STEM (Sparse Topology and Energy Management) [44] nodes wake up periodically after T time to receive potential wake-up signals (of T_{rx} length) from senders (see Figure 2.2.9). Because of $T \gg T_{rx}$ STEM results in a low duty cycle and reduces idle listening.

If a source node wishes to send data to a destination (a neighbor), it sends continuously beacons to wake up the neighbor. In the worst case, the source sends beacons over a period T. On receiving a beacon, the destination sends back an acknowledgment, keeps its transceiver in the receive state, and the source sends data.

Nodes with STEM use radios with two frequency bands: one for wake-up signals (beacons) and another for data transmission. If the transmitter allow busy tones, source nodes use them instead of beacons.

Preamble Sampling

The Preamble Sampling[12, 13] technique resembles STEM [44] but applies neither beacons nor busy tones. Figure 2.2.10 presents the solution to *rendezvous* of Preamble Sampling. Sources send preambles of size T_p in the front of every message. Since nodes wake up every T_p time, they detect the preamble transmission and keep on

listening. Sources send data after the preamble and wait for acknowledgments from receivers.

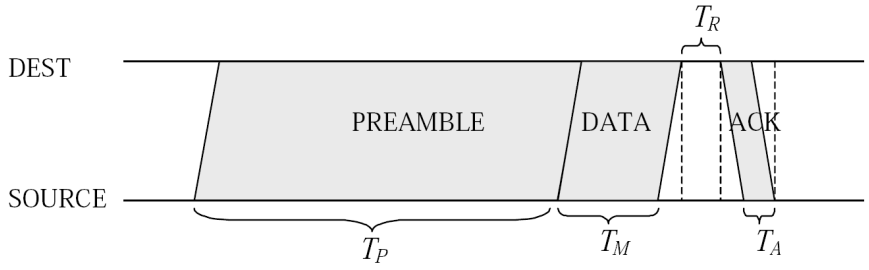


Figure 2.2.10: In Preamble Sampling nodes send a long preamble in front of every message; receivers check the channel periodically and stay awake on preamble detection

B-MAC

Berkeley Media Access Control (B-MAC) [37] resembles Preamble Sampling [12, 13] but addresses different radio characteristics. B-MAC duty cycles the radio through periodic channel sampling, referred to as Low Power Listening (LPL). Each time the node wakes up, it turns on the radio and checks for transmissions. If the node detected a transmission, it stays awake for the time needed to receive the incoming packet. After reception, the node returns to sleep. If no packet is received, a time-out forces the node back to sleep.

The activity detection, i.e., channel assessment (CCA), is critical to achieve low power. Because of false positives of CCA, a node keeps its transceiver to get potential frames, but it receives nothing, causing idle listening. Thus, B-MAC uses the noise floor estimation for both finding a clear channel on transmission and for discovering if the channel is active during LPL.

WiseMAC

Wireless Sensor MAC (WiseMAC) [14] extends Preamble Sampling [12, 13] for infrastructure networks. Such a network consists of access points with unlimited energy and of sensor nodes. Access points learn the sampling schedule of all nodes and start transmissions just at the right time with a wake-up preamble of minimized length T_p (see Figure 2.2.11). The preamble length must be long enough to compensate drift between access points and sensor nodes. Such a preamble is generally much shorter than the sampling period duration used in the Preamble Sampling (T_W in

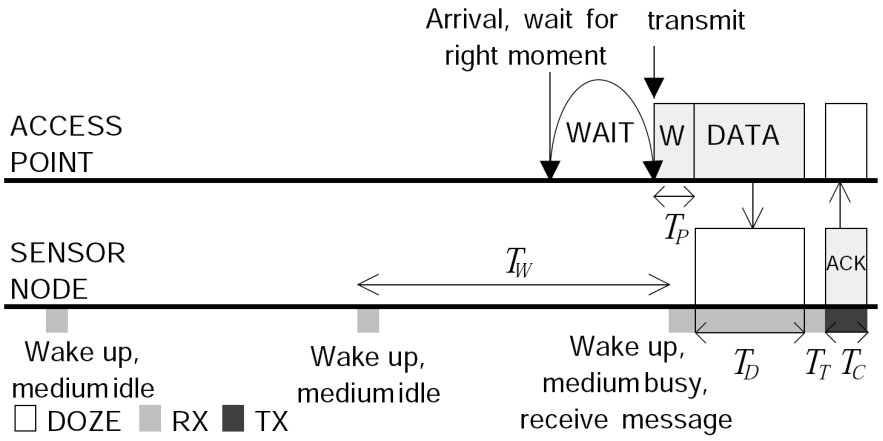


Figure 2.2.11: WiseMAC reduces the preamble length of the Preamble Sampling solution by learning nodes schedules; WiseMAC sends data just before receivers wake up, the preamble is long enough to compensate drift

Figure 2.2.11). In this way, WiseMAC shortens the preamble, reduces idle listening and saves energy.

TICER and RICER

TICER (Transmitter Initiated CyclEd Receiver) and RICER (Receiver Initiated CyclEd Receiver) [26] belong to the group of pseudo-asynchronous *rendezvous* schemes. The following paragraphs introduce both solutions.

TICER

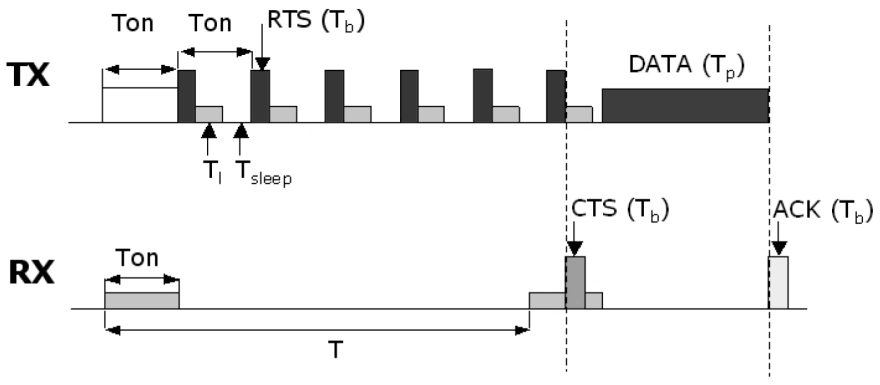


Figure 2.2.12: TICER resembles STEM and Preamble Sampling; sources send periodically wake up frames to synchronize with receivers

TICER resembles STEM [44] and Preamble Sampling [12, 13], as nodes mostly sleep and check periodically for wake-up signals.

Before sending data a source synchronizes with a destination (receiver) by sending Request-to-Send (RTS) frames in a sequence (see Figure 2.2.12). Such frequent sending of RTS may disturb other data transmissions. To solve this problem, the source node checks the channel for duration of T_{on} before sending RTS signals. If the channel is idle, the source sends RTS to the receiver. Then, the source listens for a time T_l to receive CTS (Clear to Send) responses.

On waking up, the receiver immediately gets an RTS frame and responds with a CTS signal (see Figure 2.2.12). On receiving the CTS signal, the source node sends the data packet. After correctly receiving the data packet, the destination node ends the session by sending an acknowledgment (ACK) signal to the source.

RICER

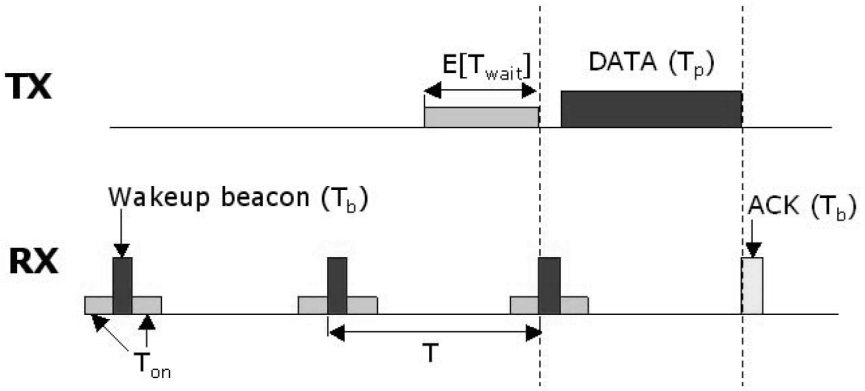


Figure 2.2.13: In RICER receivers carry the synchronization burden by sending wake-up beacons periodically

In RICER receivers synchronize wake-up times with sources by periodically sending wake-up beacons T_b (see Figure 2.2.13). A source node wanting to send data checks the channel for wake-up beacons from the receiver. On beacon reception, the source sends the data packet. The session ends with the ACK signal from the receiver to the source node.

Koala

Koala [34] addresses low duty cycle applications (less than 0.1 %) and introduces *rendezvous* based on the on the Low Power Probing (LPP). Figure 2.2.14 presents

Koala and compares it with the Lower Power Listening (LPL) of B-MAC [37] or TICER [26].

LPP resembles RICER [26], as receivers periodically send short probe messages. Nodes wishing to send data listen continuously for probe messages. On probe message reception, a source sends an acknowledgment and data to the receiver.

Apart from MAC, Koala also considers data distribution. It introduces a simple routing protocol: the gateway collects neighbor-data from all nodes, calculates the routes and spreads them to the network.

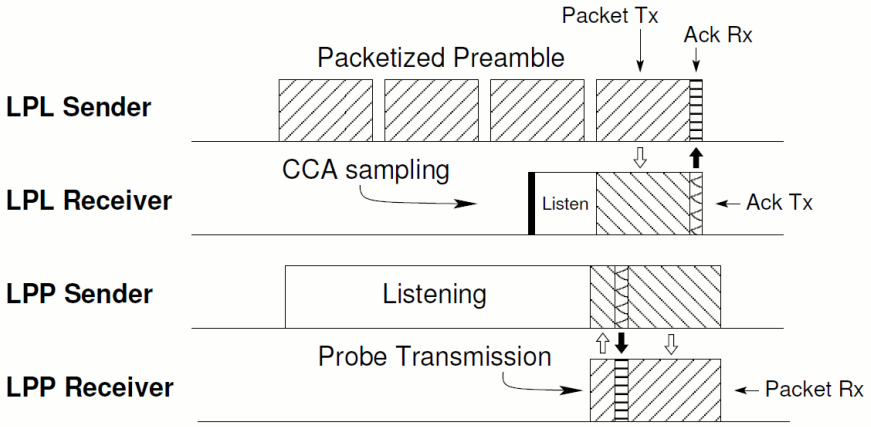


Figure 2.2.14: Koala uses Low Power Probing (LPP), which resembles RICER: senders wait for probe messages from receivers and send data afterwards; Low Power Listening (LPL) is the opposite: senders transmit probe packets

2.3 Quality of Service in This Thesis

2.3.1 Overview

The term *Quality of Service* (QoS) refers to a group of mechanisms that guarantee a certain performance. For example, some approaches guarantee specific transmission reliability expressed in a packet reception rate. Streaming multimedia services often need delays limited to a certain threshold. Some sensor network applications also need QoS guarantee.

A sensor network should provide an accurate view of monitored events. In the best case, nodes should detect all events and provide accurate sensor readings to a sink, referred to as detection reliability in ref. [22]. Any failure in event detection or false sensor reading violate QoS guarantee. However, there are several reasons

for violating QoS. For example, if sensors do not cover certain locations, they will miss some events. Besides, hardware problems with sensors result in false readings. Similarly, any network problems, like congestion or a high packet error rate, cause packet losses, and the sink may miss notices about events.

QoS of sensor networks differs from QoS of traditional networks. Usually, several sensor nodes watch a certain location, resulting in information redundancy. Obviously, when several sensors send notice about the same event to the sink, there is a higher chance the sink receives at least one frame. In addition, when several sensor nodes watch the same area, they can early detect false readings, for instance, by comparison of sensor readings. In this way, they can improve the quality of sensor readings and achieve information accuracy.

This work considers the following QoS aspects. Should nodes detect an event, they must inform the sink about it within a certain time. Several applications, e.g., critical infrastructure protection, need exactly such QoS requirements. For example, if sensor networks check factory facilities for gas leakage, they must inform the sink within a few seconds after detection to reduce the explosion danger. Such QoS requirements involve also remedies to unreliable wireless communication, as the sink must not miss event notices.

2.3.2 Reliable Data Transfer

Wireless communication suffers from various problems that affect reliability of data transfer, for instance, reflection, diffraction, path loss, attenuation, collisions [40]. Thus, sensor nodes may suffer from high bit error rates (BERs). Wireless nodes do not receive frames correctly mainly because of the following problems:

1. On getting the preamble, receivers failed to synchronize with the sender. Thus, they cannot receive frames correctly.

2. The frame exhibit bit errors, i.e., the header or payload checksum is incorrect and the frame is discarded.

Congestion

Apart from bit errors, nodes discards also frames because of congestion. In this case, nodes receive frames correctly, but they do not have enough free space in the RX buffer. As nodes cannot store frames, they discard them. Many research efforts addressed the problem of congestion in sensor networks. For example, Event-to-Sink Reliable Transport (ESRT) [41] detects congestion in the network and adjusts the reporting rate of sensors. COngestion Detection and Avoidance (CODA) [52] detects

congestion additionally through the past and the present channel load. CODA estimates the channel load by sampling it or creating a histogram of received packets. On congestion, nodes broadcast messages towards the sources to alter their sending rates. The congestion problem, however, is not addressed in this work mainly because of the nature of CIP applications. In such scenarios, nodes rarely send data to the sink, and the congestion risk is low. Clearly, several nodes can detect the same event, transmit data towards the sink and pose a congestion risk. However, since the data size is rather small, there should not be congestion problems. Besides, intermediate nodes should discover they received data about the same event and discard redundant frames.

Remedies to Bit Errors

The common remedies to bit errors include:

- Retransmissions
  Senders expects a response from receivers, i.e., whether a frame was correctly received. Should they get a negative acknowledgment or no response, senders transmit the frame again. This group of solutions is dubbed as Automatic Repeat reQuest (ARQ) [28, 27].

- Redundancy in frames
  Senders accept a block of user data and add redundancy to it. Then, they transmit frames with redundancy but do not expect any response from receivers. Owing to redundancy, receivers are able to repair some bit errors. This technique is often referred to as Forward Error Correction (FEC) [27, 28].

The ARQ protocol can be applied either on a hop-by-hop or end-to-end basis. In the first case, ARQ is implemented in the Data Link Layer, and each intermediate node uses acknowledgments and retransmissions. With the end-to-end technique, nodes implement ARQ usually in the Transport Layer, and only sources and destination apply it. That is, source nodes expect ACKs from destination nodes, which are usually located a few hops away. In this case, intermediate nodes on multi-hop paths do not check if the next node receives frames correctly. Clearly, in multi-hop networks with unreliable links the end-to-end approach may not recover from packet losses. Ref. [2] evaluates link layer and end-to-end retransmissions in sensor networks: even in small networks (a few hops only) hop-by-hop outperforms end-to-end. According to ref. [22], link layer retransmissions keep energy costs within reasonable bounds whereas costs for end-to-end approach explode from a certain threshold of the bit

error rate. Therefore, this work examines the ARQ protocol at the Data Link Layer and neglects end-to-end solutions.

Some works examined the problem of packet loss in WSN. In [54] the authors present the empirical RX rate results of nodes using CSMA/CA together with link-layer retransmissions. They considered various traffic loads, from 0.5 frames per second (fps) to approx. 4 fps. The authors present a very pessimistic view of communication in sensor networks. For example, more than 35% of the links exhibit 50% PER and more. Moreover, in such scenarios, ARQ at the link layer does not only improve the RX rate considerably (nearly 50% of links had an RX rate of only 70% at the load 1 fps), but also consumes plenty of energy: anywhere between half and 80% of the communication energy is wasted on repairing lost transmissions. However, ref. [24] and experiments presented in this work show that the RX rate is not as pessimistic as in [54]. According to [24] link-level retransmissions handle approx. 99% of errors. To achieve the remaining 1% nodes should use additional techniques, like erasure codes - a generalization of FEC. Since ARQ recovers from many more packet errors than FEC, the latter is not considered in this work.

Multiple Paths

Another way to provide reliable data transfer over unreliable link is sending data over several paths. In this case, source nodes send the same frames to the destination over different paths. In this way, the destination should receive frames even when some links suffer from connection problems. Nonetheless, although such an approach can provide reliable data transfer, it is not considered in this work, which addresses mainly solutions at the Data Link Layer.

Collisions

Some bit errors in wireless networks stem from collisions. Should two or more nodes transmit frames at the same time, receivers get corrupted data. As previously mentioned, even in applications with a low duty cycle there is a risk of collisions, as many source nodes detect the same event and try to send data towards the sink. To tackle the collision problem, nodes use various remedies. For example, CSMA/CA (Carrier Sense Multiple Access With Collision Avoidance) reduces the collision risk, as nodes postpone transmissions, if the channel is busy. RTS/CTS (Request to Send / Clear to Send) approach based on the MACAW [1] protocol avoids collisions by sending RTS frame before transmission of user data. In this case, the destination replies with a CTS frame, and the source gains permission to send frames. Other nodes receive the CTS frame as well, and postpone transmissions. Since nodes in

CIP scenarios usually send one or a few frames only, the RTS/CTS approach results in a significant overhead and will not provide notable benefits. Therefore, it is not examined in this work.

3 Problem Statement and Solutions

This work addresses sensor network applications that run several years with non-rechargeable batteries. Besides, nodes support certain QoS features, i.e., after event detection they deliver a notice to the sink within a predefined time. Such needs stem mainly from scenarios of critical infrastructure protection. For example, sensor nodes check for gas leakage in factories and notify the sink about it within a few seconds to prevent explosion danger. In addition, similar features must be supported in surveillance applications. For instance, when nodes detect a movable object, they inform the base station in a short time.

The need of a long lifetime with fast and reliable data transfer poses difficult challenges to sensor networks. This chapter introduces the main challenges and briefly introduces solutions to them. The rest of this work explains the solutions presented here in detail and evaluates them.

3.1 Tradeoff: Lifetime vs. 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, i.e., short delays and long lifetimes.

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. [10] examines various schedule approaches of sensors that cover the same sensing area in order to minimize the average EDT. However, the duty cycle of sensors is not addressed in this work.

The previous chapter introduced two main groups of protocols that support low duty cycles (LDCs): pseudo-asynchronous and synchronous. The first ones apply either long preambles or wake-up beacons in front of transmitted data. They are referred to as *Preamble Sampling*. Synchronous protocols, dubbed TDMA here, maintain a wake-up schedule and in this way synchronize active times of nodes.

Owing to long sleep periods they support low duty cycles. The following paragraphs explain how both protocol groups support short end-to-end delays.

3.1.1 Preamble Sampling

The following protocols belong to the Preamble Sampling group: B-MAC, TICER, STEM, and WiseMAC. They were introduced previously in Chapter 2.

With Preamble Sampling, nodes wake up periodically to listen for potential transmissions (see Figure 3.1.1). Nodes send a long preamble or many short frames in front of data. In the worst case, the preamble length equals the sleep period of receivers. After getting the preamble, the receiver stays awake and gets the data frame.

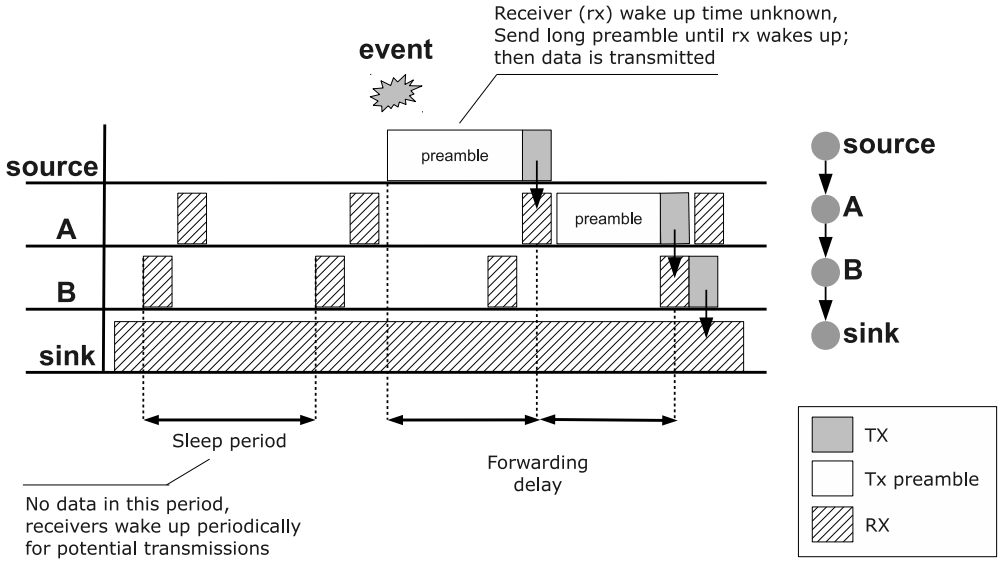


Figure 3.1.1: Preamble sampling (cycled receiver) results in significant end-to-end delays, since each intermediate node waits on average a half of the sleep period before sending data to the next node

Although these protocols were not designed to primarily support short end-to-end delays, they reduce delays by adapting the sleep period. In this case, end-to-end delays d_{EtE} consist of single forwarding delays along the path:

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

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_{sleep}, which is the

worst-case preamble length, the average end-to-end delay of n-hop path is estimated as:

$$d_{EtE} = n \cdot (\frac{T_{sleep}}{2} + t_{frame})$$

3.1.2 Duty-Cycled TDMA

Nodes support a long lifetime by applying TDMA protocols with a low duty cycle. Such protocols keep nodes mostly in the sleep state. If source nodes detect events, they cannot send data immediately to the next node. They wait until the next node is awake before forwarding frames (see Figure 3.1.2). 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, like in Preamble Sampling. End-to-end delays d_{EtE} depends on the sleep period T_{sleep} and equal on average:

$$d_{EtE} = n \cdot (\frac{T_{sleep}}{2} + t_{frame})$$

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

$$T_{sleep} = \frac{d_{EtE}}{n} - t_{frame}$$

Figure 3.1.4 shows sleep periods needed to achieve certain end-to-end delays. Nodes wake up after 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, if 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.

Several protocols, e.g., DMAC [29] and Q-MAC [50] improved the TDMA solution and introduced the staggered schedule (see Figure 3.1.3). It resembles the common practice of synchronizing traffic lights to turn green (wake up) just in time of the arrival of cars, i.e., packets, from previous intersections (hops). Nodes on the path arrange slots in a way that TX slots follow 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 influences end-to-end delays. Obviously,

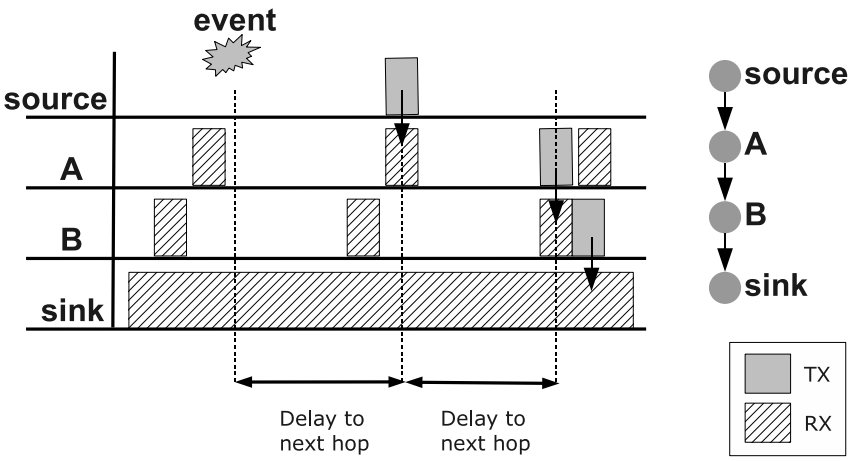


Figure 3.1.2: By applying low duty cycle protocols based on TDMA, nodes mostly sleep and cannot forward data immediately but wait until the next node is awake. It causes significant end-to-end delays

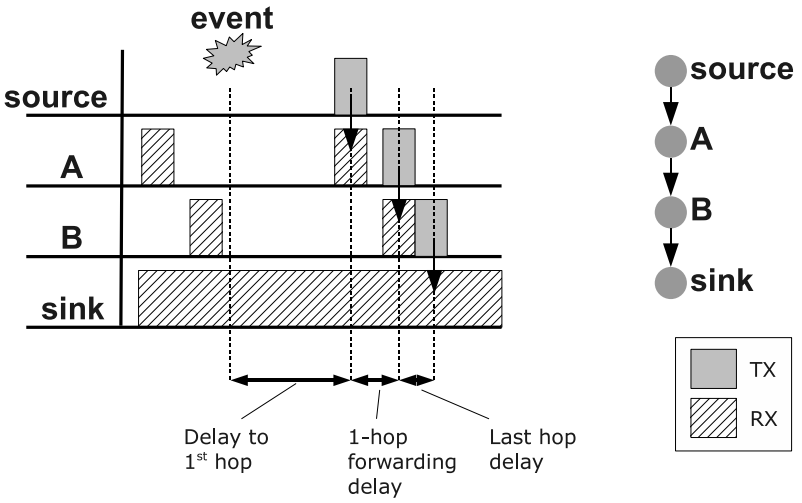


Figure 3.1.3: Nodes with the staggered (aligned) schedule forward frames just after reception and reduce end-to-end delays

the shorter the needed end-to-end delay is, the more often nodes have to wake up to take part in potential data transmissions.

Only the source node waits a long time for the next node to wake up (see Figure 3.1.3). Average end-to-end delays equals to:

$$d_{EtE} = \frac{T_{sleep}}{2} + t_{frame} + (n-1) \cdot (t_{frame} + t_{offset}) \tag{3.1.1}$$

where t_{offset} is the time between the RX slot and the corresponding TX slot on each

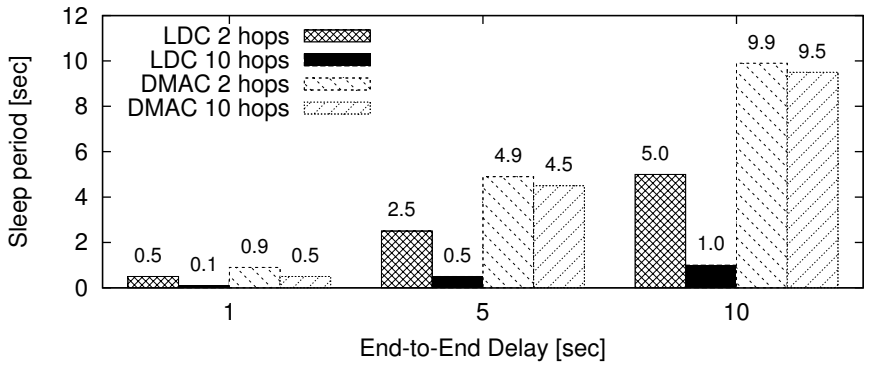


Figure 3.1.4: To support short end-to-end delays nodes with common low duty cycle (LDC) protocols (Preamble Sampling, TDMA) wake up often, especially when the distance between sources and the sink is long
DMAC introduces the staggered schedule and lowers the duty cycle, i.e., nodes wake up more rarely than in common LDC protocols to support short delays; besides, with DMAC the distance to the sink only slightly impacts the duty cycle

node. If the sink does not apply a wake-up schedule, as it is in Figure 3.1.3, the number of hops n is reduced by 2 in Eq. 3.1.1.

Nodes adapt the sleep period T_{sleep} to support certain end-to-end delays d_{EtE}:

$$T_{sleep} = d_{EtE} - t_{frame} - (n-1) \cdot (t_{frame} + t_{offset}) \tag{3.1.2}$$

Figure 3.1.4 compares the duty cycle, i.e., the sleep period, of the staggered schedule and of common LDC protocols: Preamble Sampling and TDMA. As previously mentioned, the distance between sources and the sink only slightly affects 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.

3.1.3 Evaluation and Protocol Selection

Table 3.1 summarizes the main benefits and drawbacks of the solutions that reduce end-to-end delays and provide long lifetimes of sensor nodes. That is, it includes protocols based on Preamble Sampling and on a schedule: a generic TDMA schedule and the staggered schedule.

The main advantage of protocols based on Preamble Sampling stems from their

Table 3.1: Main benefits and drawbacks of solutions that reduce short end-to-end delays at the Data Link Layer

Preamble sampling	TDMA	Staggered schedule
	Benefits	
Small code size Not affected by clock drift	Solves the collision problem	Reasonable wake-up frequency even in multi-hop networks and short end-to-end delays Solves the collision problem
	Drawbacks	
Significant energy consumption on senders because of long preambles Collision problem Frequent wake-ups to support short delays (idle listening)	Clock drift problem Large code size Frequent wake-ups to support short delays (idle listening) Idle listening because of a long time (a few ms) during wake-ups	Clock drift problem Large code size Idle listening because of a long time (a few ms) during wake-ups

simplicity. First, the tiny code size fits into the memory of sensor nodes. For example, B-MAC needs only 4 kB of ROM (Read Only Memory). Second, these protocols do not suffer from the clock drift problem, as they solve it implicitly with long preambles. On the contrary, schedule-based solutions must deal with the clock drift problem. First, it results in idle listening, since nodes wake up earlier than expected transmissions. Second, nodes need extra software to deal with this problem. Besides, schedule-based approaches set up and maintain wake-up schedules. Clearly, it also increases the total software size. Therefore, schedule-based protocols need more memory than the ones based on Preamble Sampling.

Nodes with Preamble Sampling and generic TDMA approaches wake up frequently to support short delays in multi-hop networks. Clearly, it results in excessive idle listening. However, nodes with Preamble Sampling wake up only for a time needed

to detect ongoing transmissions. For instance, since B-MAC adapts the preamble, it needs about 350 µs to detect transmissions. As only a few hardware platforms allow changes in the preamble, most protocols need longer times. They wait a time needed to receive a frame.

As previously introduced, nodes with the staggered schedule wake-up more rarely than other solutions and consequently reduce idle listening. However, all TDMA protocols suffer from idle listening when supporting end-to-end delays, as nodes need a significant time to detect idle channel. Nodes wait a time needed to receive a frame before powering down the radio. It may take even 12 ms, as presented in Chapter 5.

Preamble Sampling causes senders to waste plenty of energy because of transmissions of long preambles in front of each frame. Such preambles are few times longer than data frames. Besides, in some applications (e.g., Critical Infrastructure Protection) many sources detect the same event and send data towards the sink. In this case, long preamble poses a high collision risk, since many nodes try to transmit at the same time. Schedule-based solutions, however, do not suffer from the collision problems, as each node sends data during its timeslots only, and slots of neighbors do not overlap.

Since Preamble Sampling protocols pose a high risk of collisions in CIP applications, they are not considered in this work. Although they provide many benefits, mainly small code size and reasonable lifetimes (see Chapter 7), they may not support reliable communication because of collisions. This work applies also the staggered schedule, since it avoids collisions and does not involve frequent wake-ups when supporting short delays in multi-hop networks. Nonetheless, the staggered schedule suffers from various idle-listening problems, which significantly shorter the lifetime of nodes. The main goal of this work is to examine these drawbacks and provide efficient solutions.

3.2 Idle Listening of Staggered Schedule

This work applies the staggered schedule to support short end-to-end delays and long lifetimes of nodes in multi-hop networks. This section introduces main drawbacks of the staggered schedule.

With TDMA-like protocols, nodes on multi-hop paths to the sink wake up periodically to receive potential data (see Figure 3.2.1). As stated before, the wake-up period of nodes with the staggered schedule equals the supported end-to-end delay (see Eq. 3.1.2). In common applications, nodes rarely detect events and send data

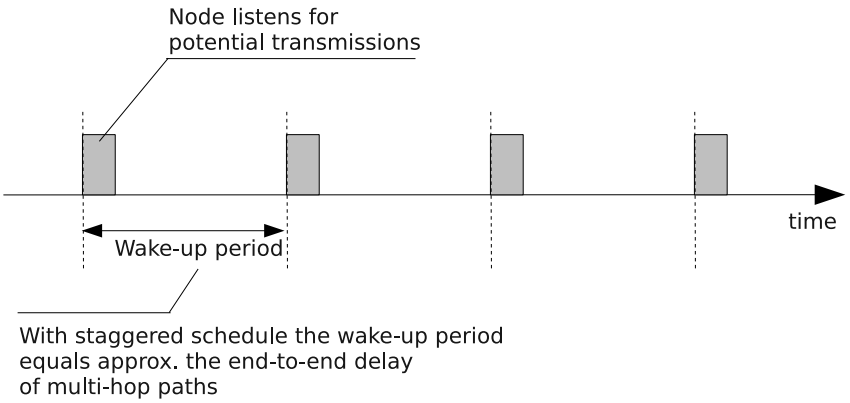


Figure 3.2.1: To support end-to-end delays, each node on a path to the sink wakes up periodically and listens for potential transmissions

to the sink. However, as nodes are not aware when events occur, they keep waking up periodically, even when no data is transmitted. In such cases, nodes wake up to receive data, but no frame arrives. Such slots are referred to as *passive slots*. When sources detect events, they send notices to the sink along the path. Intermediate nodes receive and send such frames in *active slots*.

Frequent wake-ups result in excessive idle listening, as nodes waste energy in passive slots. Clearly, nodes cannot reduce idle listening by applying longer wake-up periods, as it would result in too long end-to-end delays (see Eq. 3.1.2). Common solutions result in unnecessary long idle listening of active and passive slots. Thus, there is still a room for optimization of single wake-up times. That is, nodes can apply shorter active and passive slots.

Figure 3.2.2 presents a typical RX slot of scheduled MAC protocols. First, nodes listen for the guard time to compensate clock drift. Then, they receive a preamble and detect the Start Frame Delimiter (SFD). The frame follows the SFD and is stored in the RX buffer. After that, the transceiver triggers the microcontroller (µC) to handle the received frame by raising the RX interrupt. The µC gets the frame from the RX buffer and reads the contents. Should no frame follow the one just received, the µC powers down the transceiver. Figure 3.2.2 shows also idle listening of both active and passive slots, explained in the following paragraphs.

Long Guard Times

Sensor nodes derive time usually from crystal oscillators, which have certain precision δ, expressed in parts per million (ppm), according to the crystal cut. That

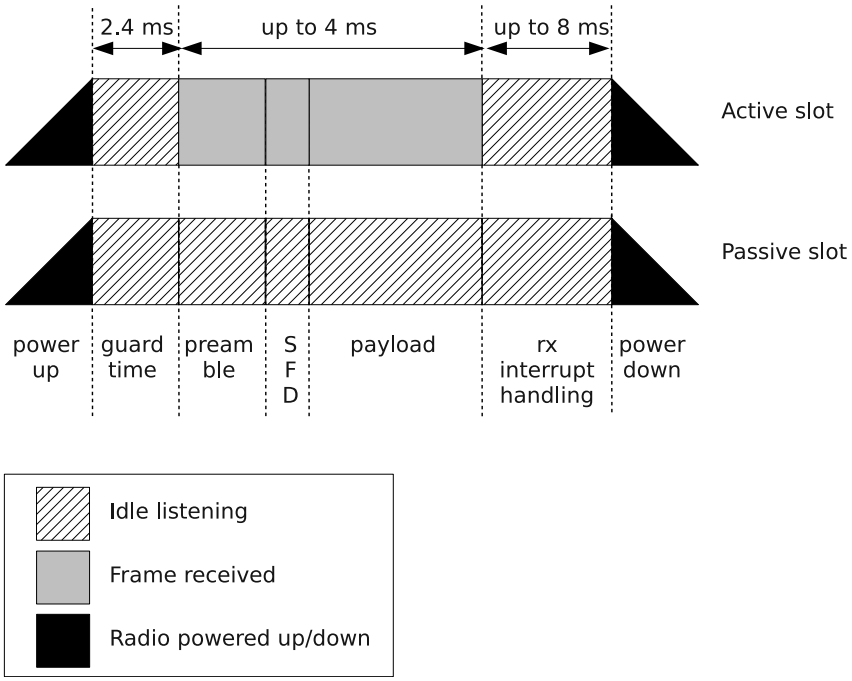


Figure 3.2.2: Idle listening of the staggered schedule in active and passive slots; duration of each phase is based on Tmote Sky nodes

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. Mainly changes of temperature and air pressure cause short-term drift variations.

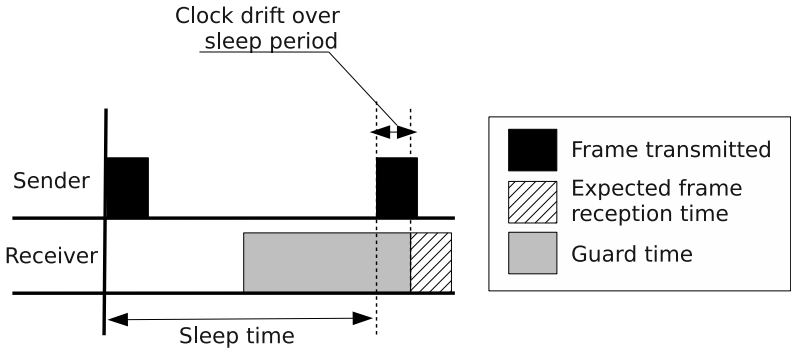


Figure 3.2.3: To compensate clock drift, receivers wake up earlier by guard times

Each scheduled MAC protocol suffers from the drift problem illustrated in Figure 3.2.3. 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 solve this problem, 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. However, short guard times may not compensate drift, and nodes miss some frames. A common solution estimates guard times for worst-case drift. For example, with a sleep period of 1 minute, Tmote Sky nodes use guard times of 2.4 ms. 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. This problem affects active and passive slots of the staggered schedule (see Figure 3.2.2). If nodes apply shorter guard times, they reduce idle listening of both slot types, save energy and prolong the lifetime.

Idle Listening of Passive Slots

As already mentioned, nodes must wake-up periodically to check for potential transmissions and therefore cannot avoid passive slots. Nodes usually detect passive slots indirectly, i.e., they wait the normal time it takes to receive frames (see Figure 3.2.2). However, the time needed to receive a single frame and deliver it to the application is even 14 ms on the Tmote Sky node (see Chapter 5). In other words, nodes wait such a long time in passive slots before powering down the transceiver. It results in excessive idle listening, wastes energy and shortens the lifetime. Besides, in common scenarios there are many more passive than active slots. Thus, the passive slots consume a huge amount of energy, even more than 50% of the total energy (see DMAC results in Chapter 7). It shows that nodes can reduce idle listening and prolong the lifetime, if they early detect passive slots and power down the transceiver.

Idle Listening of Active Slots

In active slots idle listening stems from guard times and the time needed to detect that no frame follows the one just received (see Figure 3.2.2). For instance, protocol headers may contain information about following frames. Thus, MAC protocols get frames from the RX buffer, read headers and decide whether it can power down the transceiver. However, as reading data from the RX buffer can take even a few ms (see Chapter 5), it results in significant idle listening. Nonetheless, it only slightly affects energy consumption, as there are only a few active slots in common applications.

Impact on Lifetime

This paragraph introduces an example application to illustrate the impact of idle listening on the lifetime of sensor nodes. In this scenario, Tmote Sky nodes monitor

an environment, for example, intrusion detection in a security area or a gas leakage risk in a factory. On event detection, nodes must notify the sink within 5 seconds by sending a notice over multi-hop paths. They apply the staggered schedule and wake up about every five seconds (see Eq. 3.1.2). Clearly, the amount of transmitted data affects energy consumption and depends on the event frequency. This evaluation considers three scenarios that differ in the average event period: 1 minute, 1 hour, and 12 hours. To counter the clock drift problem, nodes synchronize wake-up times every 60 seconds and apply guard times. This evaluation considers the slot length of 11 ms (see Figure 3.2.2). The time to get the frame from the RX buffer (RX interrupt handling) was obtained empirically (see Chapter 5).

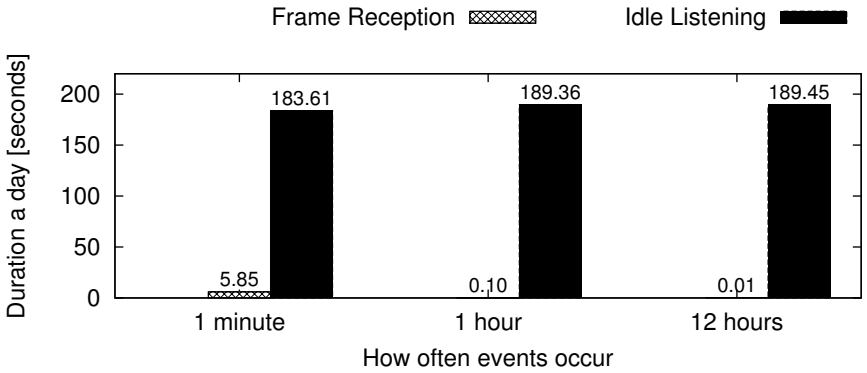


Figure 3.2.4: Idle Listening of the staggered schedule in applications with 5-second end-to-end delays compared with the total frame reception time; the figure presents results for three different values of average event frequency

Figure 3.2.4 presents the idle-listening time T_{idle} of both active and passive slots in this application. Besides, it also shows the total reception time T_{rx}, i.e., the time that each node needs to receive data frames. As there are more passive slots in all scenarios and such slots are the main reason for idle listening, T_{idle} is always significantly longer than T_{rx}. In the worst case, nodes send data every 12 hours, and the number of passive slots is about 8000 larger than of active ones. Therefore, T_{idle} is longer by four orders of magnitude than T_{rx}, i.e., 189.45 s and 0.01 s respectively.

There are only minor differences in the total idle-listening time among all scenarios, and T_{idle} is always longer than 180 seconds. This small variation stems from the different number of active and passive slots in all scenarios. That is, the number of passive slots is higher in scenarios with rare events, since nodes rarely send data. As passive slots cause more idle listening than active ones (see Figure 3.2.2), T_{idle} is longer in scenarios with rare events. Nonetheless, the total energy wasted because of idle listening is similar in all cases, i.e., from 1.12 mAh in a day to 1.16 mAh.

According to the lifetime model introduced in Chapter 7, nodes need about 0.5 mAh a day for other activities in similar scenarios[1]. Thus, nodes waste about $\frac{2}{3}$ of the total energy in idle listening caused by frequent wake-ups. Clearly, it shortens the lifetime of sensor nodes significantly.

These observations show clearly that the reduction of idle listening is crucial to achieve longer lifetimes in applications that support short end-to-end delays. As idle listening wastes about $\frac{2}{3}$ of the total energy, even small improvements can significantly prolong the lifetime.

3.3 Link-Layer Reliability

By applying hop-by-hop solutions to communication reliability, nodes reduce the risk of frame loss in the wireless channel. This paragraph briefly presents CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance) and ARQ (Automatic Repeat reQuest) approaches, and their impact on the energy consumption and lifetime.

CSMA/CA

Several MAC protocols, e.g., IEEE 802.15.4 [20], apply CSMA/CA to deal with collisions. In short, each sender performs the following steps:

1. The node tests, if the channel is free (Carrier Sense).
2. If the channel is free, the node sends the frame after a random time (the lower and upper bounds of the wait time depend on the protocol).
3. Otherwise (channel is busy) the node waits a *back-off time* (BOT), which is a random value within the predefined bounds and checks the channel again (step 1). For example, the BOT can be longer than 100 ms in IEEE 802.15.4.

Receivers do not know the current BOT estimated by the sender. Therefore, especially in scheduled MAC protocols, like the staggered schedule of DMAC, they must listen for potential transmissions during the BOT and cause idle listening.

ARQ

Nodes with the ARQ approach send back an acknowledgment (ACK) on frame reception. If senders do not receive an ACK in a predefined time, they assume the frame was lost and repeat the transmission. Another version of ARQ uses negative

[1] The energy wasted because of self-discharge of batteries is neglected here.

acknowledgments (NACK) instead of ACKs. In this case, receivers detect a frame loss in a packet stream and send NACK to senders. To detect a missing frame, senders include consecutive sequence numbers in each frame transmitted. Receivers detect a frame loss on a missing number.

Idle-Listening with Staggered Schedule

To transfer data reliably to the sink, nodes should apply CSMA/CA and ARQ solutions. However, it results in an extra overhead and increases the energy consumption. The main reason for the extra overhead is idle listening caused by both approaches:

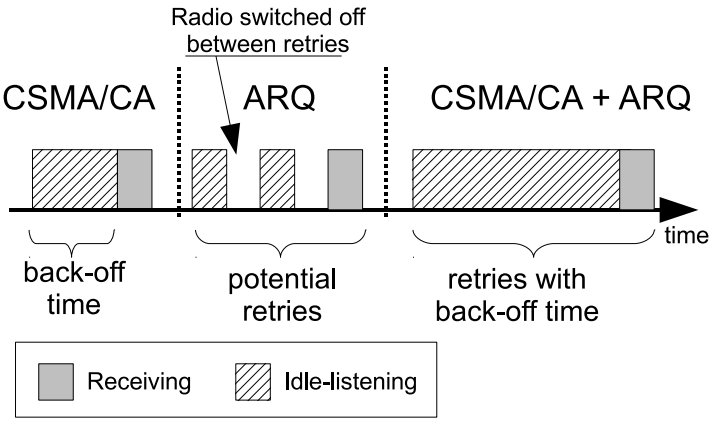


Figure 3.3.1: CSMA/CA and ARQ increase the idle-listening time; with CSMA/CA receivers waits additional back-off time; with ARQ receivers listen during potential retries

- Receivers with CSMA/CA cannot find out during the listen state whether senders do not intend to send data, or they wait BOT before starting a transmission (see Figure 3.3.1). Therefore, CSMA/CA prolongs the idle-listening time of passive slots by the worst-case BOT, i.e., the longest BOT multiplied by the highest number of back-off tries.

- ARQ results in longer idle listening in passive slots as well. Receivers in the listen state cannot find out whether a frame was lost, or senders did not send anything. Thus, receivers listen the time needed to get all retries (see Figure 3.3.1). It increases idle listening of passive slots significantly. Besides, nodes consume extra energy for sending acknowledgments and retries.

3.4 Solutions

This section briefly introduces solutions, explained in detail later in this work, that support short end-to-end delays and reliable link-layer communication in long-living applications. Figure 3.4.1 pictures the solutions in the Open Systems Interconnection (OSI) model.

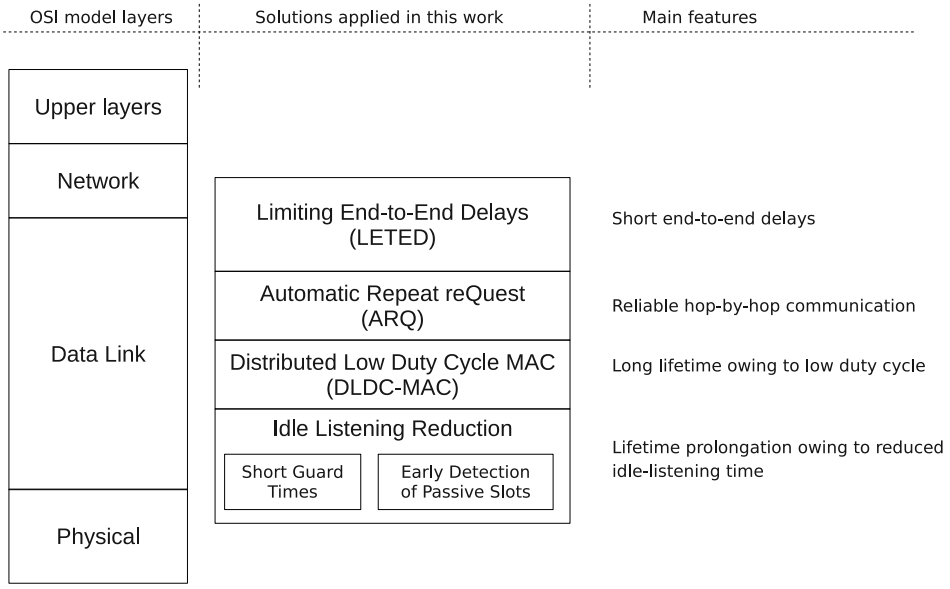


Figure 3.4.1: Solutions to the problem of long lifetime with short end-to-end delays introduced this work

Solutions implemented on many OSI layers influence the lifetime of nodes. PHY specifies the technology used for transmissions, e.g., carrier frequency and modulation, which obviously affects energy consumption. Many approaches of the Medium Access Control (MAC), a sublayer of the Data Link layer, support low duty cycles and prolong the lifetime, as presented previously. The Network layer decides which paths are used for transmissions. For example, to save energy, it can select paths that consume least energy. Nonetheless, effective remedies for long lifetimes of nodes and short end-to-end delays lie in proper wake-up times (see the staggered schedules introduced previously). Therefore, this work addresses mainly the Data Link layer and does not consider solutions at other OSI layers.

There are two major protocol groups that support long lifetimes of nodes and short end-to-end delays, that is, Preamble Sampling and TDMA. This work applies the latter approach and provides solutions that reduce idle listening of TDMA. Although Preamble Sampling is not examined in this work, Chapter 7 evaluates it against the solutions presented here.

To support long lifetimes of sensor nodes, this work introduces a distributed low duty cycle MAC (DLDC-MAC) protocol. As DLDC-MAC does not provide short end-to-end delays, this work applies also LETED (Limiting End-to-End Delays), which is based on the staggered schedule. In short, LETED maintains wake-up schedules along multi-hop paths from sources to the sink. Therefore, it works together with routing protocols implemented at the network layer.

As introduced earlier in this chapter, the staggered schedule results in excessive idle listening in active and passive slots. Therefore, DLDC-MAC and LETED suffer from common idle-listening problems, which waste plenty of energy and significantly shorten the lifetime of nodes. This work addresses this problem and introduces novel solutions that reduce idle listening, mainly at the MAC level (*Idle Listening Reduction* in Figure 3.4.1). First, sensor nodes collect drift samples and predict future drift to neighbors. By doing so, they estimate guard times based on run-time drift and not on worst-case drift. In this way, nodes use short guard times and reduce idle listening. Second, some commercial transceivers provide a feature that notifies the microcontroller just after they started frame reception, i.e., after receiving the preamble and discovering the Start Frame Delimiter (SFD) field. Since it needs cooperation with the PHY layer, Figure 3.4.1 places *Idle Listening Reduction* at the border between Data Link and PHY layers. This work exploits the SFD-discovery feature and introduces a solution that early discovers passive slots, powers down the transceiver and significantly reduces idle listening.

Since this work considers the Data Link layer, it also examines the impact of reliability mechanisms implemented in this layer on the lifetime: CSMA/CA and ARQ. The empirical and analytical results show that CSMA/CA causes mainly energy waste in applications with a low duty cycle but does not increase significantly the communication quality. On the contrary, nodes with ARQ benefit from reliable communication and still work for a long time. However, ARQ achieves such good results in the lifetime owing to the solution that early detects passive slots.

The following sections explain the above-mentioned solutions to long lifetimes of nodes, short end-to-end delays and reliable link-layer communication in sensor networks.

3.4.1 Low Duty Cycle Protocol with Staggered Schedule

This work introduces a distributed low duty cycle MAC (DLDC-MAC) protocol to support long lifetimes. In short, nodes with DLDC-MAC periodically transmit beacons and wake up to receive beacons that are sent by neighbors (see Figure 3.4.2). Obviously, this protocol resembles other TDMA approaches, e.g., S-MAC, Dozer or

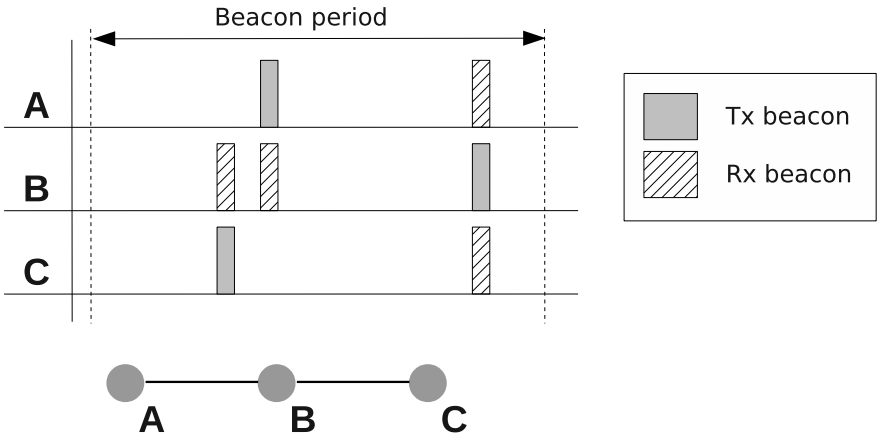


Figure 3.4.2: DLDC-MAC: nodes wake up to send beacons and to receive beacons from neighbors. Data is included in beacons or sent shortly thereafter

IEEE 802.15.4 in beacon-enabled networks. As none of these protocols provided all features needed for LDC applications with fast and reliable data transfer, the DLDC-MAC solution was designed. The main differences from other protocols are:

- 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 receive only parent's beacons. If 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.

- 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 ref. [53], it may use guard times as long as 0.5 second, i.e., longer by two orders of magnitude than the time needed to send a single frame. Second, S-MAC prolongs the active period with extra RTS and CTS frames. Third, S-MAC neglects important TDMA protocol problems, like the overlap problem of two separate wake-up schedules.

- The IEEE 802.15.4 standard does not allow multi-hop communication with beacons. It supports multi-hop networks only without beacons.

In addition, DLDC-MAC supports data replication in sensor networks and handles several TDMA problems. Chapter 4 introduces DLDC-MAC thoroughly and presents the results of a real-world experiment.

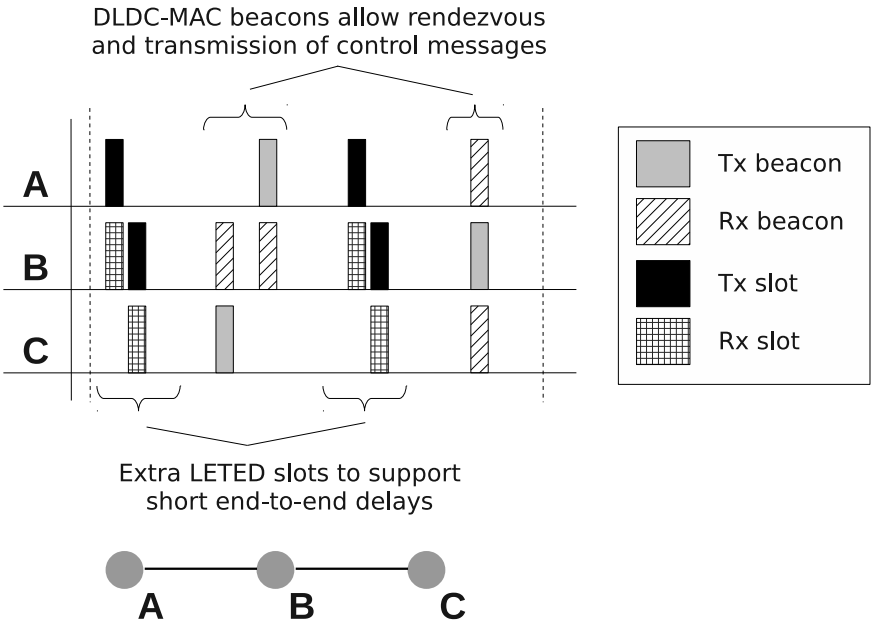


Figure 3.4.3: DLDC-MAC with LETED provide low duty cycles and short end-to-end delays

Obviously, if nodes use only DLDC-MAC to support short end-to-end delays, it will result in often wake-up times and energy waste, as previously introduced in Section 3.1.2. Thus, apart from DLDC-MAC nodes use an extra staggered schedule to provide short delays to the sink. Such a schedule is named in this work LETED (Limiting End-to-End Delays). In this case, nodes wake up to send and to receive DLDC-MAC beacons and for LETED slots (see Figure 3.4.3).

Nodes use DLDC-MAC to send non-time-critical data, e.g., route discovery frames or to set up a new LETED schedule. Therefore, DLDC-MAC works with a low duty cycle and consumes only a fraction of energy. According to the evaluation presented in Chapter 7, 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 an extra LETED wake-up schedule on paths towards the sink. However, nodes create such schedules on demand and only on selected paths. Clearly, if each node sets up a separate wake-up schedule to the sink, it will result in an excessive overhead. Especially nodes close to the sink wake up often, as they keep up schedules of several sources. To avoid such risks, nodes limit the number of schedules. For example, intermediate nodes that maintain schedules of other sources should use them for transmissions of their own data. However, this

work does not examine efficient solutions to set up many wake-up schedules. It is a part of future research efforts.

The LETED approach is discussed thoroughly in Chapter 5.

3.4.2 Idle Listening Reduction

In some applications, the staggered schedule wastes even $\frac{2}{3}$ of energy on idle listening. Therefore, this work addresses this problem and provides solutions that significantly reduce idle listening, save energy and prolong the lifetime. The following paragraphs shortly introduce the major solutions.

Short Guard Times

As previously mentioned, nodes use guard times to counter the drift problem (see Figure 3.2.3). Long guard times used to compensate worst-case drift, e.g., in the Dozer [9] protocol, result in energy waste because of idle listening. Ref. [43] combines hardware and software solutions to achieve accurate times. In short, sensor nodes equipped with two oscillators estimate the local time precisely, that is, with frequency stability of ± 1.2 parts per million (ppm). However, the approach needs dedicated hardware with two oscillators and the calibration of both oscillators. Clearly, such a solution is not feasible in large-scale networks.

There are also other hardware solutions that address the clock drift problem. For example, sensor nodes can use oscillators with higher precision and use short guard times. Such oscillators draw typically too much current or/and are expensive. Therefore, they are not widely used in sensor networks. However, the Maxim DC32kHz [31] oscillator fits well to sensor nodes. It is a temperature-compensated crystal oscillator (TCXO), that is, it measures periodically the temperature and adjust the crystal frequency according to it. By doing so, it provides frequency stability of ± 2 ppm and draws only 1 µA current. In this case, nodes need about 0.24 ms guard times to compensate drift of 1-minute period. Although this TCXO costs only about $3, i.e., a few times more than common oscillators used in sensor nodes, it is rarely used in sensor networks. This TCXO is also a few times larger than common oscillators of sensor nodes. It might be the main reason why Maxim DC32kHz is not widely used in sensor networks.

This work introduces only software solutions to the drift problem, since they can be applied to any hardware platform. Besides, the solution introduced in Chapter 6 results in about 120 µs idle listening for a sleep period of 1-minute. Approaches based on worst-case drift but with accurate TCXO (Maxim DC32kHz) cause 2x longer idle

listening. Nonetheless, this TCXO is a good alternative to the software solution, since it significantly reduces idle listening and does not need extra software.

To deal with the problem of long guard times and idle listening, nodes can apply time synchronization protocols (TSPs). Should a TSP provide accurate time synchronization between neighbors, they can use guard times shorter than the worst case. However, TSPs send extra synchronization (SYNC) frames, consume more energy and shorten the lifetime. Besides, since node must not miss SYNC frames, they use long guard times in this case. Short guard times are used only with data frames. Clearly, because of the overhead of SYNC frames, TSPs are not energy-efficient means to reduce idle listening in applications with a low duty cycle.

Since clock drift is stable over a short time, nodes can estimate future drift precisely and use shorter guard times. For example, Rate Adaptive Time Synchronization (RATS) [16] predicts future drift by applying LR (linear regression). Although it significantly shortens guard times, it needs an emulation of floating-point arithmetic on microcontrollers. When using single precision the truncation error can vary from $\pm$17us to $\pm$17.7ms. When using more accurate, i.e., double-precision operations, OLS may take even 120ms. Besides, floating-point module and operations need extra memory, which is limited on sensor nodes. Thus, these drawbacks limit the use of linear regression approach on sensor nodes.

Several time synchronization protocols address the drift prediction problem. Although they do not refer to it explicitly, nodes can benefit from their drift prediction methods to prolong the lifetime of LDC protocols. Flooding Time Synchronization Protocol (FTSP) [30] introduces the drift prediction based on LR to increase synchronization accuracy. Symmetric Clock Synchronization [46] estimates the relative drift to the reference clock with the weighted moving average filter. The authors use the drift estimation mainly to increase accuracy of the time synchronization protocol. It resembles slightly the approach based on the moving average exploited in this work. However, there are many open issues when adapting the approach [46] to LDC protocols, mainly how to estimate the length of guard times. Sensor nodes using Gradient Clock Synchronization [45] protocol repeatedly collect drift samples from their neighbors. Then, each node updates its logical clock according to the received drift samples. In that way, all nodes converge into a common clock and can predict future drift of this clock.

In this work, nodes predict drift by applying the moving average filter on previous drift samples. First, they need only a few samples for accurate estimations. Second, such prediction works with simple mathematical operations and does not need floating point arithmetic. This work shows that such a solution achieves results as

good as predictions based on linear regression. Therefore, it fits resource-constrained sensor networks.

The experiments presented in Chapter 6 revealed that receivers compensate drift of 98-99% frames with short guard times. To cover drift of all frames, they need guard times longer than the oscillator worst case, as other factors influence drift as well, e.g., jitters in radio start time.

Both approaches, that is, drift prediction and short guard times needed to compensate drift of most frames, shorten guard times 18x when compared to the solution based on worst-case drift. Chapter 6 explains the solution to guard times in detail and provides evaluation results.

Early Detection of Passive Slots

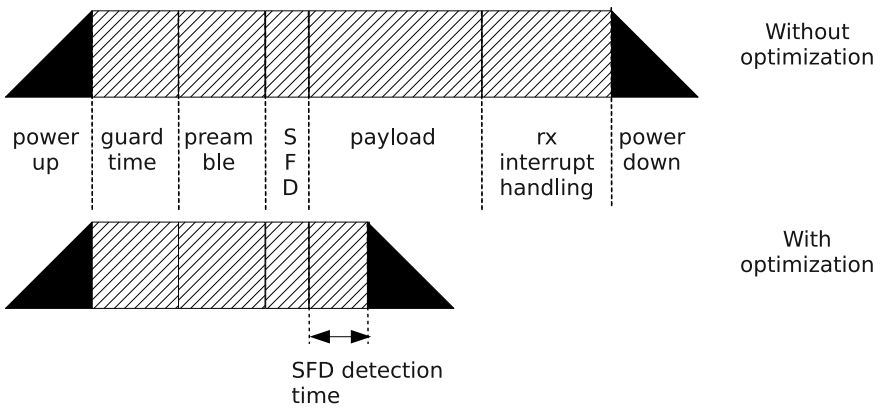


Figure 3.4.4: Owing to early SFD detection, nodes shorten passive slots and save energy

This work introduces solutions that significantly shorten passive slots. First, they reduce the guard times by applying solutions presented in the previous paragraph. Second, nodes detect passive slots early and power down transceiver immediately after. In this case, they exploit the feature provided by some commercial transceivers, e.g., ChipCon CC2420. On SFD reception, the transceiver raises the SFD interrupt on microcontrollers. Thus, should the SFD interrupt not be raised in the expected time, nodes assume no frame arrives and powers down the transceiver (see Figure 3.4.4). Thus, nodes shorten passive slots $t_{passive}$ to:

$$t_{passive} = t_{guard} + t_{preamble} + t_{SFD} + t_{SFD\_detect}$$

where t_{guard} is the guard time, $t_{preamble}$ and t_{SFD} are the RX times of preamble and SFD, and t_{SFD_detect} is the time the µC needs to detect SFD reception, i.e., the

SFD interrupt delay. As nodes need only 100 µs to detect the SFD reception and use short guard times, they reduce passive slots about 13x. Chapter 5 provides more details about the solution.

3.4.3 Reliable and Efficient Link-Layer Communication

The solution presented in this work uses the ARQ protocol to deal with unreliable wireless links. However, nodes apply ARQ to LETED slots only and not for sending beacons. When nodes send data in LETED slots, they expect an acknowledgment (ACK) from the receiver. If they do not receive the ACK, they send frames again. The evaluation of ARQ presented in Chapter 8 revealed that retries only slightly affect the lifetime. For example, ARQ shortens the lifetime by 10% but may improve the reception rate by 20% and more. Such good lifetime results of ARQ stem from the solutions that keep passive slots short, introduced in the previous paragraph. Therefore, although ARQ increases the energy consumption of passive slots four times, it is still relatively small.

As LETED is a TDMA protocol, it does not suffer from the collision risk. However, should slot and beacons overlap because of clock drift, LETED does not prevent collisions in this case. Thus, nodes might use CSMA/CA to reduce the collision risk, but it results in excessive idle listening. Besides, the real-world experiments revealed that nodes should not apply CSMA/CA, if they already use ARQ. In this case, the extra CSMA/CA approach does not significantly increase the reception rate.

Chapter 8 introduces the experiment results of ARQ and CSMA/CA and presents the impact of both solutions on the lifetime.

4 Distributed Low Duty Cycle MAC (DLDC-MAC)

Despite many research efforts in duty-cycled sensor networks there are no protocols that meet three following needs:

1. Low duty cycle with rendezvous
2. Support for decentralized networks
3. Limited end-to-end delays

This chapter introduces DLDC-MAC protocol, presented in works [3, 6], that meets first two needs and serves as a basis for the last one. Next chapter introduces LETED approach for limiting end-to-end delays. Only DLDC-MAC coupled with LETED meet all three needs.

DLDC-MAC resembles other protocols for sensor networks with a low duty cycle. The closest relative of DLDC-MAC is Dozer [9], as it uses beacons and deals with the clock drift problem is a similar way. Since Dozer is limited to tree networks rooted at the sink, it does support decentralized networks. Besides, nodes communicate with parents and children only but not with all neighbors. Therefore, in case of connectivity problems, nodes cannot easily discover alternative paths to the sink, since they are not aware of other neighbors. DLDC-MAC, on the contrary, also supports decentralized networks, i.e., without a permanent sink. Nodes learn about all neighbors and can send data to them any time, provided they are not in the sleep state.

A reliable data storage tinyDSM [36] is an example of decentralized networks. To provide a fast and reliable access to data stored in sensor networks, tinyDSM mirrors sensor readings on several nodes. First, tinyDSM ensures that information is available even if nodes are exhausted. Second, nodes process mirrored data directly and do not need to ask the data owner. For example, such nodes reply to the sink immediately, and in this way support a fast access to the information. Owing to DLDC-MAC, tinyDSM can mirror sensor readings throughout the network and still achieve good results in lifetime.

4.1 Protocol Description

4.1.1 Rendezvous with Beacons

DLDC-MAC solves the rendezvous problems, i.e., the synchronization of wake-up times between senders and receivers, with beacons. Nodes send beacons periodically and wake up to receive the beacons of neighbors, like in Dozer. All nodes have the same beacon period. Even when nodes do not have data (e.g., sensor readings) to send, they still send beacons. Clearly, it results in extra energy overhead. However, the evaluation presented in Chapter 7 reveals that nodes with DLDC-MAC consume only a fraction of energy for sending and receiving beacons.

Nodes mostly sleep and wake up only to receive beacons from neighbors and to send their own beacons (see Figure 4.1.1). In this way, DLDC-MAC synchronizes wake-up times between neighbors, and nodes can send data throughout the network, although they mostly sleep. On receiving beacons, nodes estimate the time of the next beacon by adding the beacon period to the RX time.

Each beacon transmission consists of three phases (see Figure 4.1.2). First, nodes send a beacon, that is, a single broadcast frame to all neighbors. After that, they stay in the RX state shortly. During this time, other nodes can send control frames (e.g., network join, new timeslots). Finally, nodes send awaiting application data to neighbors.

4.1.2 Beacon Schedule Setup

Figure 4.1.1 presents the schedule setup. After powering on, a node is not aware of its neighbors and their beacon times. Therefore, the node listens for the whole beacon period, receives beacons and stores the RX times. Then, the node informs the neighbors it wishes to join the network. The node sends *join-network* frames to each neighbor separately just after they sent their beacons and entered the RX state (see Figure 4.1.2). If two or more nodes send such frames at the same time, the receiver does not get any frame because of collision. Therefore, a node that wishes to join the network sends frames as long as it does not get an acknowledgment from the neighbor. To tackle the collision problem, it does not send frames on every neighbor's beacon. Since *join-network* frames contain the beacon time of the new node, neighbors learn about new wake-up times to receive beacons.

If nodes miss beacons during the schedule setup, they do not detect some neighbors. To solve this problem, nodes may repeat the neighbor discovery a few times a day. However, it results in a significant energy penalty, as evaluated in ref. [3].

Nodes can tackle the above-said problem by collecting the neighbor lists of their

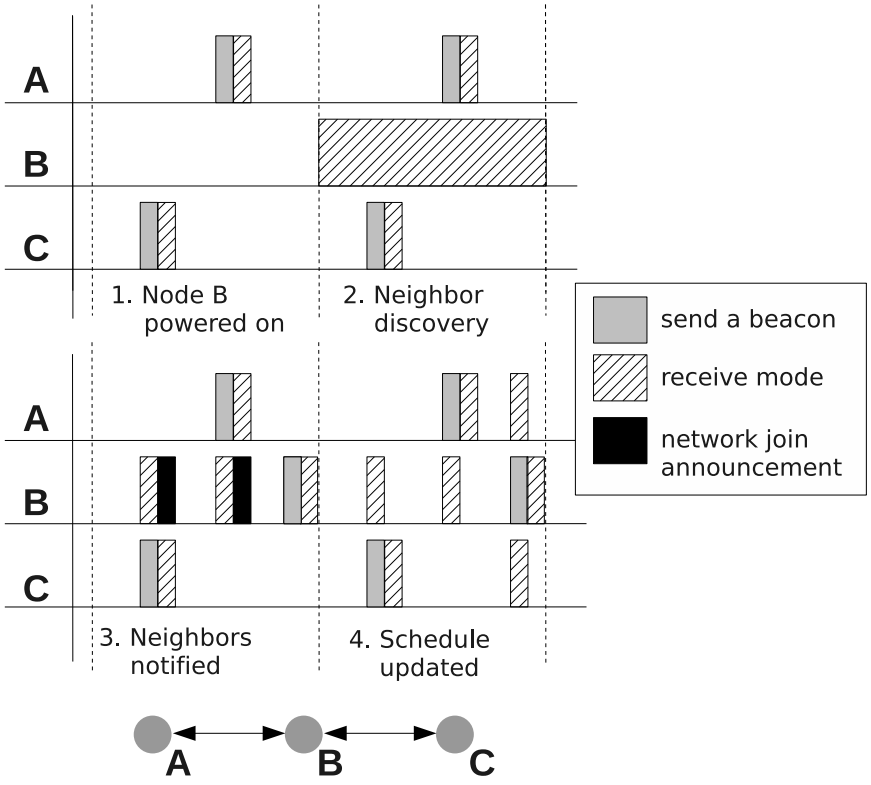


Figure 4.1.1: Nodes send periodically beacons and stay in the RX state to get potential frames from neighbors
Network join: after powering up, nodes listen the beacon period and discover neighbors; after that, they send *network join* commands to the neighbors

neighbors. By doing so, nodes learn about beacon times in 2-hop neighborhood. Then, nodes try to receive beacons from any two-hop neighbor. If they receive such a beacon, they add the corresponding node as a new neighbor.

Nonetheless, this work does not examine the problem of missing beacons during the neighbor discovery. It is a part of future research efforts.

4.1.3 Data Transmission

With Beacons

Nodes include in beacons data of upper layers that is not time-critical. The largest beacon size depends on the underlying physical layer, e.g., 128 bytes for IEEE 802.15.4. If the data exceeds the largest beacon size, the node sends data in separate frames, after sending the beacon and finishing the RX state (see Figure 4.1.2).

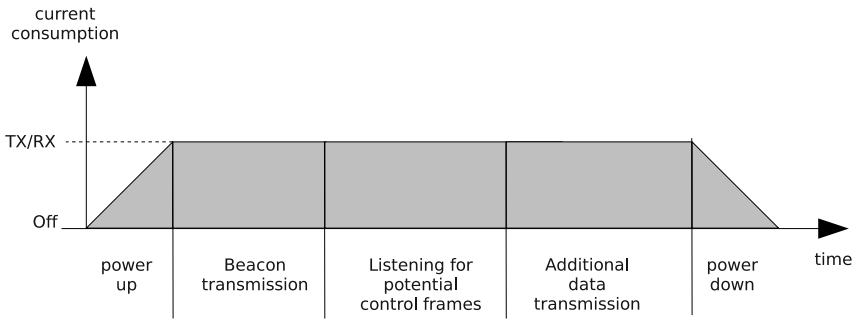


Figure 4.1.2: Single beacon; after sending a beacon, nodes wait a certain time for incoming frames, e.g., network join frames; if awaiting data (from upper layers) do not fit into beacons, nodes send it after the receive slot

Extra Time Slots

Sending data with beacons result in data forwarding delays in multi-hop networks, as nodes cannot forward data immediately after receiving. Nodes wait for their beacon time to forward data. In the worst case, nodes wait almost the whole beacon period before forwarding. If the network should support short delays, nodes arrange extra data slots. Next chapter presents the details of limiting end-to-end delays (LETED).

4.2 Solutions to Wireless Problems and Clock Drift

4.2.1 Clock Drift and Missed Beacons

Because of clock drift, nodes may wake up too soon or too late to receive beacons. In that case, they miss beacons, and lose wake-up synchronization with neighbors. To avoid such a risk, nodes compensate clock drift by waking up earlier than the expected beacon time (see Figure 4.2.1). The waiting time for the beacon is referred to as a guard time. Chapter 6 introduces main solutions to the guard time estimation and evaluates their impact on the lifetime.

Sensor nodes calculate guard times for each neighbor separately, according to clock drift and the last successfully received beacon of this neighbor. If nodes miss a beacon, they prolong the guard time for the next RX try. Obviously, such an approach causes a longer idle-listening time and increases the energy consumption. However, by doing so, the synchronization of wake-up times works even when nodes miss several beacons, e.g., because of short-term connection problems.

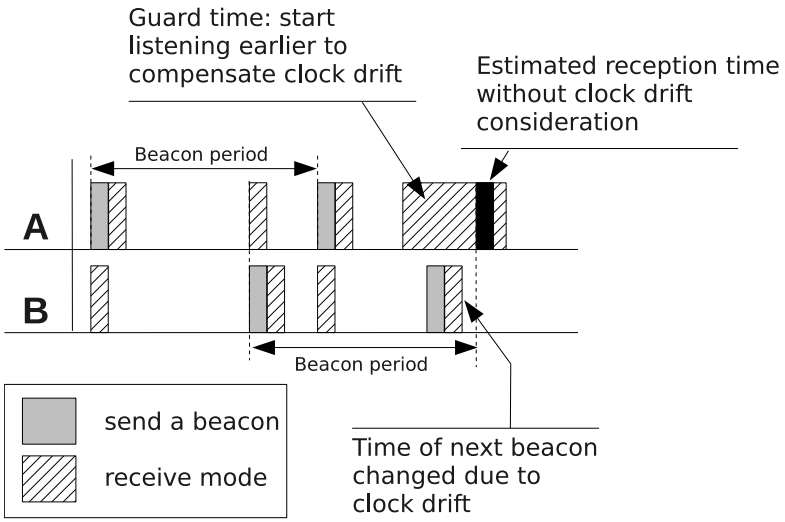


Figure 4.2.1: Solution to the clock drift problem; each node calculates drift since the last received beacon (guard time), starts listening for the next beacon earlier and stays awake longer by the guard time

4.2.2 Asymmetric Links

With asymmetric links, nodes send data to neighbors but do not receive anything. Ref. [54] empirically evaluates the performance of packet delivery in dense sensor networks. For example, more than 10% of link pairs have significant asymmetry. By default, DLDC-MAC ignores links that are permanent asymmetric. However, when links are asymmetric only temporarily, DLDC-MAC does not discard them. In this case, it just prolongs guard times in order to receive beacons after the connection becomes symmetric again.

4.2.3 Link Failures

Nodes detect broken links, when they do not receive several consecutive beacons from a neighbor. However, after detecting a broken link, nodes do not mark the neighbor as not-working immediately, since the link may suffer from short-term problems. For example, this section introduces experiment results with sensor nodes deployed indoors. Because of a high interference rate with other wireless devices, some links stopped working for a short time, but were available again afterwards. Therefore, DLDC-MAC marks such neighbors as not working temporarily. To preserve energy, nodes stop waking up to receive beacons from such neighbors. After some time, they try to receive beacons again. If they still do not receive beacons, they assume the neighbors as permanent not-working. Clearly, after nodes did not receive several

beacons, they use long guard times to receive beacons again. However, with the drift prediction approach, presented in Chapter 6, they keep guard times relatively short in this case.

The number of missed beacons after neighbors are marked as not-working depends on the scenario, on transmission conditions, etc. This holds also true for the number of beacon periods that need to pass before trying to reconnect to a temporarily not working neighbor.

4.2.4 Beacon Overlap Prevention

Because of clock drift, beacons of different nodes move relatively to each other. Finally, the beacons overlap leading to collisions. To deal with this problem, nodes monitor times of their own beacon and of neighbors. When the time difference between any two beacons is smaller than a threshold, the affected node shifts its beacon. To prevent frequent beacon changes, nodes find the longest unoccupied period in the beacon schedule of 2-hop neighborhood.

Nodes notify neighbors about beacon shift in advance. Because of unreliable links, however, some neighbors may miss such notices. Therefore, nodes shift beacons a few periods after they started announcing it. In this time, they expect to receive acknowledgments from neighbors.

4.2.5 Hidden Terminal Problem

The solution to beacon overlap prevents the hidden terminal problem as well. The problem arises if two or more nodes are out of their communication range, meaning they are unaware of their transmissions. These nodes, referred to as hidden terminals, may send data at the same time and cause collisions on receivers.

A similar problem arises in DLDC-MAC, if any two neighbors of a node send beacons at the same time. However, as nodes monitor beacon times of all neighbors, they discover the risk of hidden terminals. In this case, the time difference between two beacons is smaller than a threshold. The node prevents the risk by requesting one of the neighbors to shift its beacon time.

4.2.6 Collision Avoidance

Collisions might occur only if the times of beacons overlap. Nodes avoid such risks by shifting beacons, as introduced previously. Therefore, DLDC-MAC does not need approaches to collision avoidance, like CSMA/CA.

4.3 Experiment

The goal of the experiment was to test whether the DLDC-MAC works on a real hardware platform, and deals with the unreliable wireless channel. The following paragraphs give more details about the experiment.

4.3.1 Implementation

Hardware and Operating System

The DLDC-MAC version considered in this experiment runs on the TinyOS [25] operating system and Tmote Sky sensor nodes. Later on the protocol was designed again in a cross-platform manner and works with various operating systems [8]. Table 4.1 presents the size of DLDC-MAC implementation, which fits into a limited memory of Tmote Sky. DLDC-MAC uses only 14 kB of flash memory, leaving more than 30 kB for an operating system, other protocols and applications. Besides, it uses more than 2 kB of RAM for transmit and receive queues. Should it be necessary, DLDC-MAC can use smaller queues and occupy less memory.

Table 4.1: The size of DLDC-MAC implementation for TinyOS

	Flash	RAM
DLDC-MAC with TinyOS	25.8 kB	4.9 kB
TinyOS alone with radio driver	11.5 kB	0.4 kB
DLDC-MAC alone	14.3 kB	4.5 kB

Precise Estimation of RX Time

To calculate next wake-up times DLDC-MAC needs RX times of beacons. The more precise RX times are, the shorter are guard times, and the less energy is consumed while listening for beacons. An obvious way to determine RX times is to read the timer register after a frame was received. However, such a solution may cause non-deterministic delays, as it involves raising an interrupt and handling it.

The DLDC-MAC implementation exploits the opportunity provided by the Tmote Sky design. When the transceiver receives the Start Frame Delimiter (SFD) of a new message, it sets a microcontroller pin to one. The microcontroller stores the current timer counter in a register. Then, the software reads the register after receiving the complete message and determines the RX time. In that way, it does not have any

delays caused by software execution. As Tmote Sky uses a 32 kHz oscillator, the reception time is estimated with precision of about 30 microseconds.

In this experiment, DLDC-MAC compensates drift by applying guard times based on worst-case drift between neighbors.

4.3.2 Experiment Setup

In this evaluation, ten Tmote Sky nodes were placed in an office environment (see Figure 4.3.1). To allow a multi-hop experiment, and resemble a more realistic scenario, the transmitter output power was reduced to -25 dBm, decreasing the radio range to a few meters. Node 1 served as a sink and was permanently connected to a logging computer. Other nodes periodically sent statistics to the sink, and it forwarded them to the computer.

Some nodes, e.g., nodes 6 and 7, could not reach the sink directly and sent frames over multi-hop paths. All nodes applied a simple routing protocol to support multi-hop communication. The protocol resembles Greedy Perimeter Stateless Routing (GPSR) [23]. That is, only nodes closer to the sink forwarded received frames. However, GPSR uses position information to decide whether nodes are closer to the sink and should forward frames. In this scenario, nodes used the hop count as position information, i.e., only nodes with the hop count smaller than the hop count of the previous sender forwarded frames towards the sink. Since the network topology may have changed, the sink issued topology-update frames every 10 minutes. Such frames contained the hop count from the sink. Each node that received such a frame incremented the hop count value, updated its distance to the sink and forwarded the frame.

In this scenario, nodes used the DLDC-MAC protocol with a beacon period of one minute.

4.3.3 Results

Beacon Overlap and Clock Drift

Figure 4.3.2 shows beacon times observed by node 3. The beacon of node 3 is always at 0 seconds. As expected, beacon times of neighbors were not constant and changed according to relative drift between nodes. Therefore, nodes shifted beacons to avoid the overlap risk. For example, after two days, the beacon time of node 2 was close to the beacon of node 3. Thus, node 3 shifted its beacon and prevented the collision risk.

Although nodes shifted beacons and avoided the overlap risk, it does not prove

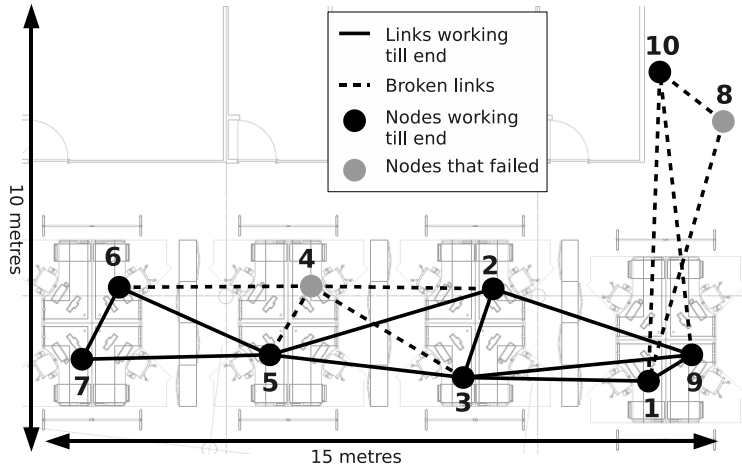


Figure 4.3.1: Topology of the sensor network after 14 days of the office experiment (node 4 and node 8 broke down during the experiment, probably because of a loose contact with batteries)

the solution will always work without problems. The evaluation considered only ten nodes that worked a short time. In common scenarios there are many more nodes, and they work months or years. Nonetheless, the experiment shows that DLDC-MAC code run on Tmote Sky nodes without problems. Besides, the DLDC-MAC implementation was coupled with LETED solution and tested on the OMNeT++ network simulator (see Chapter 5). This large-scale experiment considered 155 nodes, which worked about 3 months. During this time nodes did not encounter any problems with DLDC-MAC.

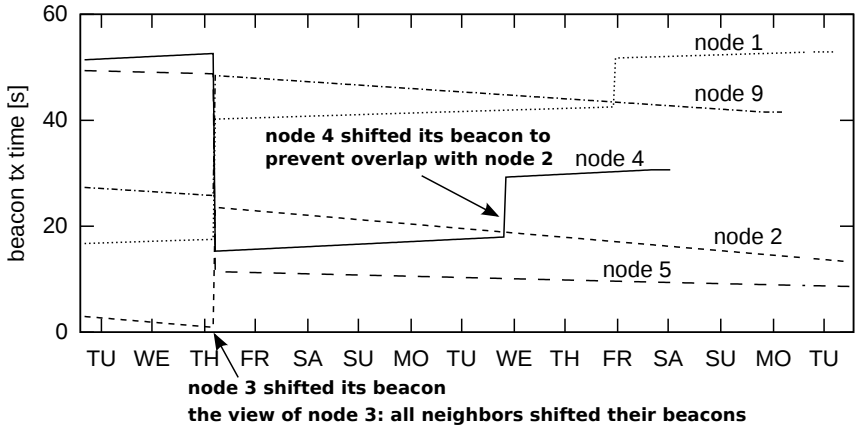


Figure 4.3.2: Relative beacon times observed by node 3, i.e., the beacon time of node 3 is always at 0 seconds;
Because of clock drift, the beacon times were changing, and nodes shifted their beacons to avoid the overlap risk

Robustness Against Link Failures

The nodes used the IEEE 802.15.4 standard with 2.4 GHz frequency band. As there were other wireless devices using the same frequency band during the experiment, like WLAN access points, laptops, and other sensor nodes, they interfered with the sensor network.

Figure 4.3.3 presents the amount of received beacons by node 3 during the experiment, separately for each neighbor. As expected, at weekends and during nights, when there was almost no WLAN communication, node 3 received almost all beacons. However, during weekdays, node 3 missed many beacons, especially from node 9, because other wireless devices influenced transmissions. However, communication with node 2 was always reliable, even during the day, and was not affected by WLAN interference. The distance between nodes 2 and 3 was much smaller than between nodes 3 and 9. Therefore, the received signal from node 2 was much stronger and not affected considerably by other transmissions. The RX signal strength of node 9 was much smaller that from node 2, as node 9 was located far away. Therefore, WLAN affected the RX signal of node 9, leading to missed beacons.

The amount of received beacons from node 4 and node 9 was sometimes 0% during the day. Such poor results stem from the way DLDC-MAC handles link failures. Nodes assume a neighbor as not working temporarily, if they miss 10 successive beacons. In that case, they stop receiving beacons from this neighbor and try again after an hour. Thus, after missing 10 beacons, node 3 stopped receiving beacons from nodes 4 and 9 for an hour, presented as 0% of received beacons in Figure 4.3.3. After this time, node 3 reconnected to nodes 4 and 9, and received beacons again.

Although node 3 missed plenty of beacons during the interference periods, it stayed synchronized with the neighbors. That is, node 3 still received beacons from the affected nodes after the interference periods. It shows that DLDC-MAC synchronization of wake-up times works even in the presence of unreliable links. However, since nodes consider worst-case drift in this experiment, node 3 used long guard times to compensate clock drift for 1-hour period. As it results in excessive idle listening, nodes ought to exploit the clock prediction approaches presented in Chapter 6. In this way, they shorten guard times and reduce idle listening.

Direct vs. Multi-Hop Communication

Direct communication over long distances may lead to a high packet loss rate, as the RX power decreases with the distance from the transmitter. Thus, frames transmitted over long distances are not received correctly, if the noise level is higher than the reception power. In such cases, a multi-hop communication can result in a

higher reception rate than direct transmission. Such a case was observed during the experiment.

Communication between nodes between nodes 3 and 9 exhibits the problem stated above. Node 3 missed many beacons from node 9 during the daytime (see Figure 4.3.3). The following analysis presents what would happen, if node 9 did not send directly data to node 3 but through node 1. First, node 1 received almost all beacons from node 9, even during the day (see Figure 4.3.4)[1]. Second, node 3 received almost all messages from node 1, apart from the last two days of the experiment (see Figure 4.3.3). Therefore, node 3 would receive almost all messages from node 9, if they were sent not directly but through node 1. In that case, multi-hop communication achieved a higher packet delivery rate than direct communication.

In addition, the example of node 10 confirms that direct communication over a long distance may result in a high packet loss rate. Figure 4.3.4 shows that many frames of node 10 did not reach the sink. Since node 8 stopped working at the beginning of the experiment, node 10 sent data directly to the sink. Obviously, the reception signal from node 10 at the sink was very weak and WLAN devices affected transmissions.

Node 7 could reach node 1 with a multi-hop communication only. Although node 7 was at least three hops away from node 1, its end-to-end reliability was higher than reliability of node 10, which could directly reach node 1. The reasons for that are obvious. First, node 1 received several copies of data from node 7, as the packets were forwarded through multiple paths. Second, the links on the way from node 7 to node 1 were reliable, since the distances between forwarding nodes were relatively small. As a result, a multi-hop communication from node 7 to the sink was more reliable than a direct communication from node 10.

The above observations reveal a need of cooperation between layers, especially Layer-2 (Data Link Layer) and Layer-3 (Network Layer) to achieve good results in a packet delivery rate. That is, both layers can reject weak direct links, if there is a multi-hop connection available. In this case, L2 may send control frames to a neighbor over a multi-hop path and not directly as is the custom. By doing so, L2 can increase reliability of communication with neighbors.

[1] The figure presents the results of end-to-end communication, typically multi-hop; however, in the case of node 9 it was direct communication because of the small distance between nodes 1 and 9

Lost Synchronization

After ten days, node 1 did receive data from node 10 (see Figure 4.3.4), as they lost the synchronization of wake-up times. The reason for this problem is the following. As previously mentioned, on missing a beacon, nodes prolong the guard time in order not to miss following beacons because of not compensated clock drift. Communication with node 10 resulted in a high beacon miss rate. Such a high rate was not expected before the experiment, and therefore guard times were limited to 40 ms. However, clock drift to node 10 was exactly 40 ms after 2.2 hours. Therefore, when node 1 did not receive any beacons from node 10 after 2.2 hours and more, it used the longest guard time (40 ms). However, in this case, node 1 needed a longer guard time to compensate drift of such long periods. Clearly, because of too short guard times, node 1 could not reconnect to node 10 and lost the synchronization.

These observations show that this version of DLDC-MAC needs further adaptations, if it should support scenarios with links that are unavailable for several hours but need to be used anyway. DLDC-MAC currently discards such links, because of the following reasons. First, if a link was not working over a long period, it should be discarded so that routing protocols find an alternative path w/o the affected link. Second, if a node keeps waking-up to receive potential beacons from a not-working neighbor, it consumes a significant amount of energy.

However, owing to the drift prediction approaches, introduced in Chapter 6, nodes use short guard times to compensate drift. Therefore, they can reconnect to neighbors after long periods of unreliable communication and do not suffer from excessive idle listening.

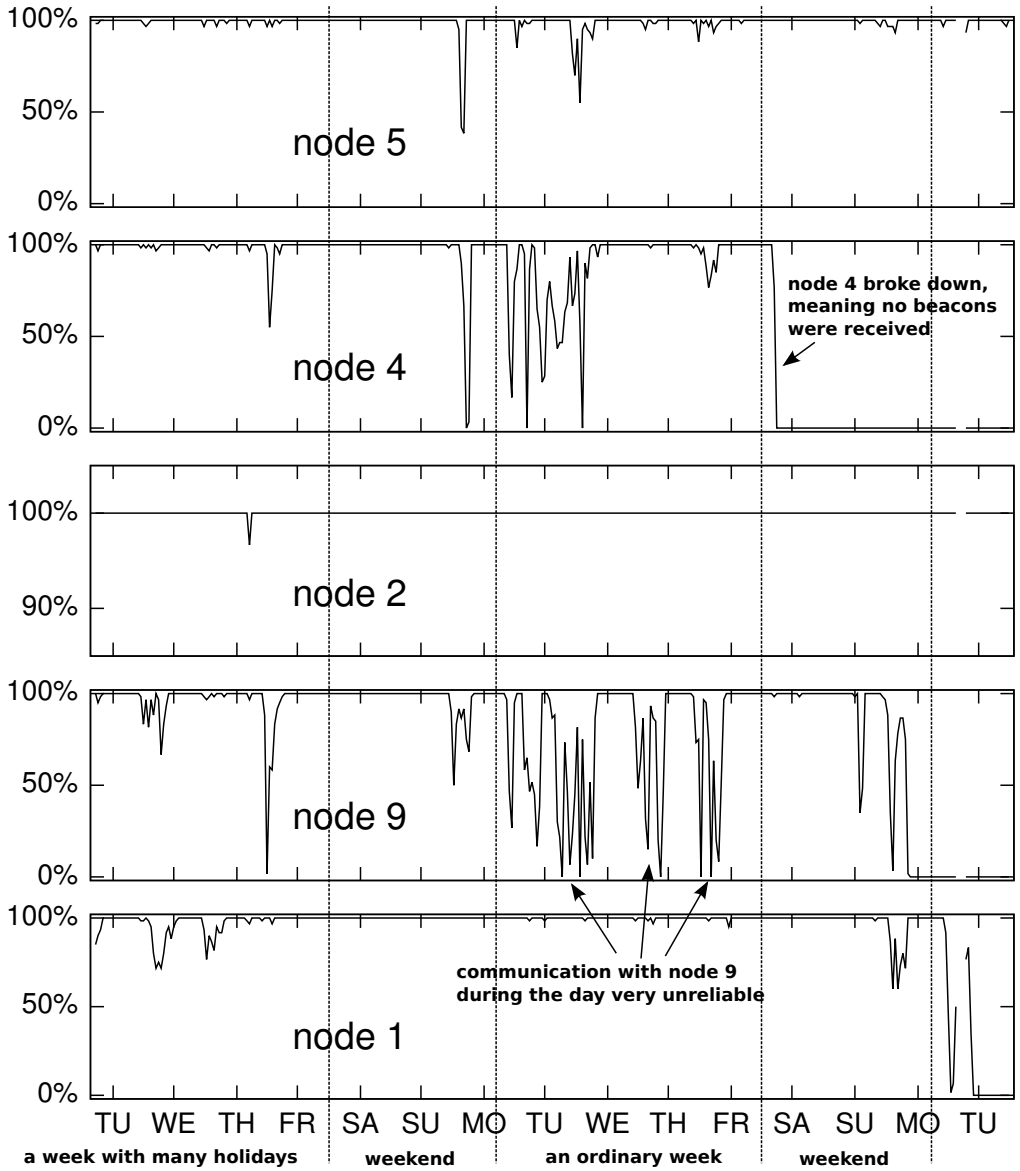


Figure 4.3.3: The amount of received beacons by node 3 from all neighbors during the experiment; because of WLAN interference, node 3 missed lots of beacons from node 4 and node 9 during the daytime; midnight is marked with the ticks on the X-axis

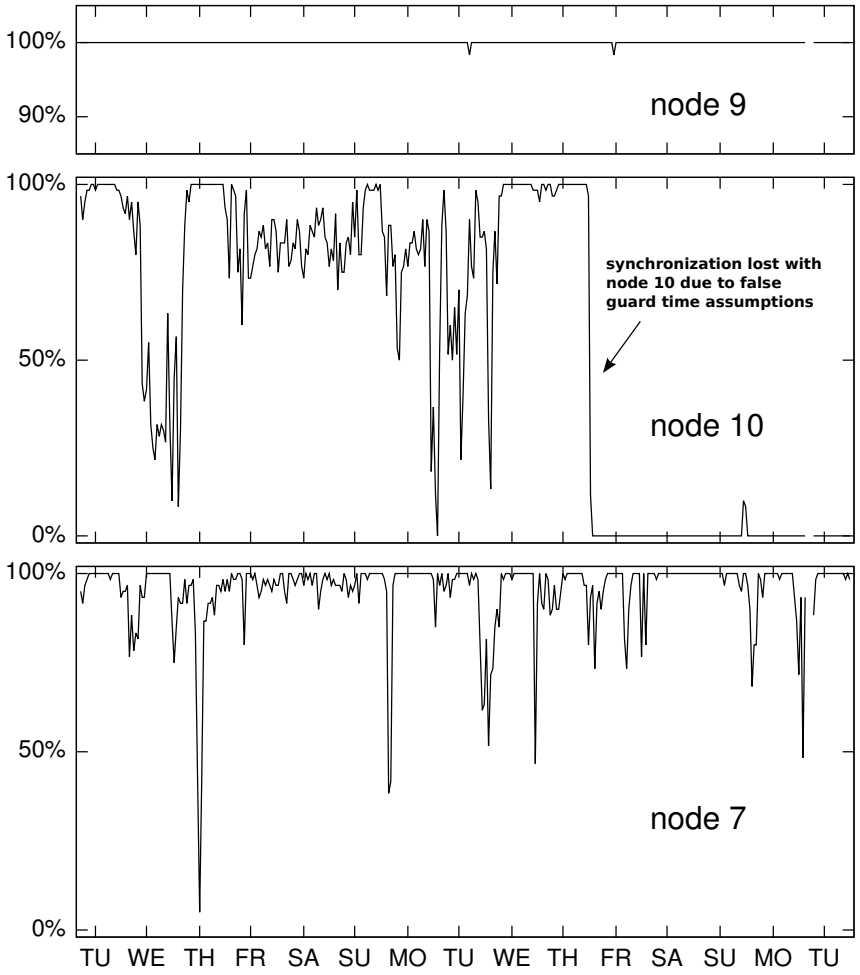


Figure 4.3.4: Data received by node 1 from other nodes; the multi-hop connection to node 7 (at least 3 hops away) was more reliable than the direct connection to node 10 (neighbor)

5 Limiting End-to-End Delays (LETED)

This chapter introduces LETED, i.e., a set of solutions that limit end-to-end delays in sensor networks. LETED adapts the staggered schedule provided by DMAC [29], Q-MAC [50] and ref. [10]. The staggered schedule was introduced briefly in Chapter 3. LETED handles also drift problems neglected in previous works. As LETED needs an underlying MAC protocol, this work couples it with DLDC-MAC. Nonetheless, LETED works with other MAC approaches as well.

LETED was partly introduced in ref. [7] but later simulative evaluations revealed some drawbacks of this protocol. This chapter presents LETED with novel improvements, and the solutions to idle listening avoidance (ILA) [4].

5.1 Overview

As already mentioned, LETED adapts the staggered schedule, i.e., nodes on the path to the sink settle a wake-up schedule in a way that it limits end-to-end delays (see Figure 5.1.1). In short, each TX slot follows immediately the RX slot from the previous node. Therefore, nodes send messages just after reception.

In applications with a low duty cycle most wake-ups are idle. That is, nodes wake up but do not receive anything. However, they must wake up to take part in potential data forwarding. Otherwise, they cannot support short end-to-end delays. Such idle slots are referred to as passive in this work (see Figure 5.1.1).

Nodes with LETED start transmissions exactly at TX slots, i.e., they do not apply CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) or similar solutions, which may postpone transmissions. Because of the scheduling approach presented here, nodes usually do not need such means, since the schedule is a TDMA approach and inherently avoids contention. However, because of clock drift, slots may overlap and cause a collision risk, as introduced in Section 5.5. Nonetheless, extra medium access means result in longer guard times and cause excessive idle listening. Chapter 8 evaluates CSMA/CA and confirms that it results in a significant energy penalty in low duty cycled networks. Therefore, LETED does not use such medium

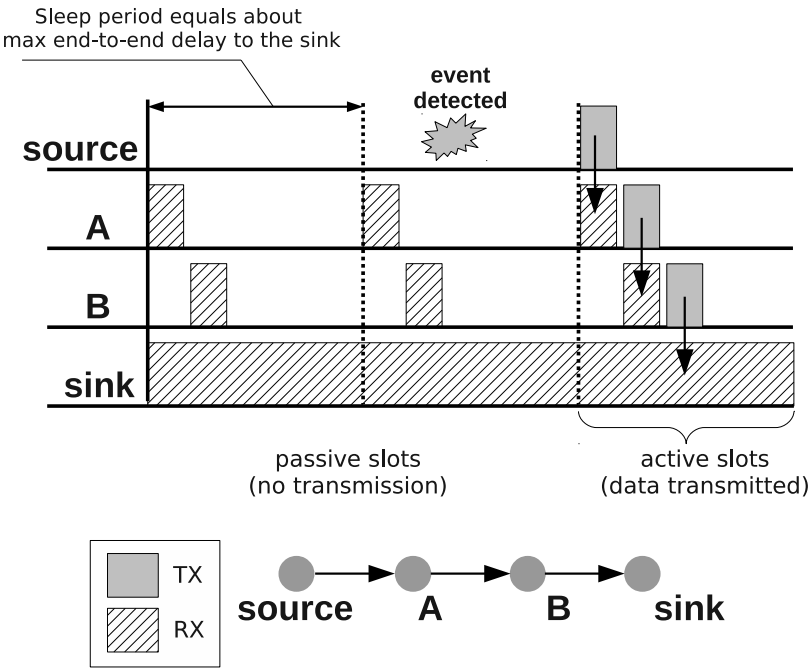


Figure 5.1.1: LETED is based on the staggered schedule; in applications with low duty cycles, most slots are passive, i.e., nodes do not send anything but listen for potential transmissions

access means.

To deal with unreliable wireless links, nodes use the ARQ [27] 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. Chapter 8 presents the ARQ performance in duty-cycled wireless networks.

5.2 Schedule Setup

The schedule setup involves cross-layer cooperation among the application, the network, and the MAC layer (LETED and DLDC-MAC in this case), as depicted in Figure 5.2.1. 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 LETED, and it sets up new time slots with the next node. If the routing protocol does not have a route to the sink, it discovers a new path and triggers LETED.

Considering the hop distance to the sink, provided by the routing, LETED calcu-

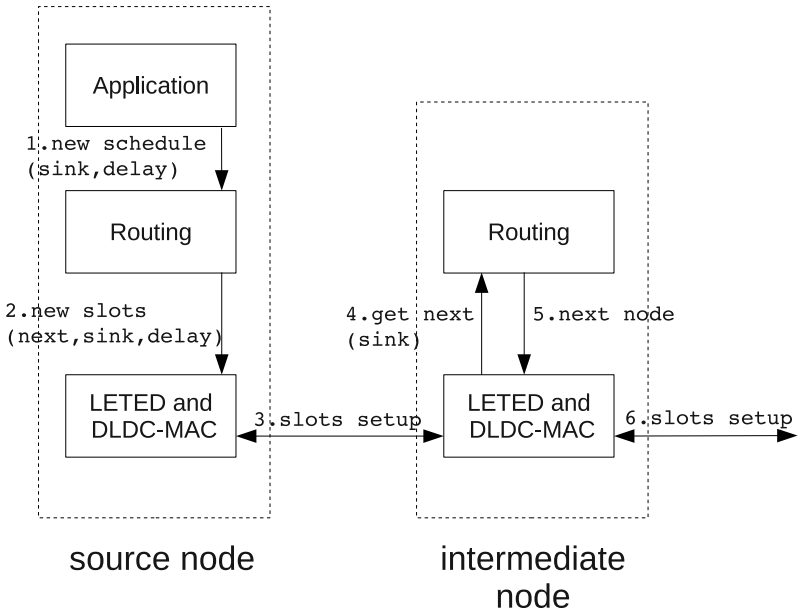


Figure 5.2.1: Setup of a new wake-up schedule; cross-layer approach among the application, the network layer (Routing) and the data link layer (LETED coupled with DLDC-MAC)

lates how often nodes must wake up to support d_{EtE}. As presented in Figure 5.1.1, on event detection, the source does not send a notice at once but waits for the next TX slot. To support d_{EtE} the source node needs TX slots every T_{slot} time, as previously introduced in Chapter 3. That is, since intermediate nodes cause extra delays, the source estimates the total forwarding delay $d_{forwarding}$ and the slot period T_{slot} as:

$$T_{slot} = d_{EtE} - t_{frame} - d_{forwarding} \tag{5.2.1}$$

$$d_{forwarding} = (n-1) \cdot (t_{frame} + t_{tx\_offset}) \tag{5.2.2}$$

where n is the number of hops to the sink, t_{frame} is the expected frame size, and t_{tx_offset} is the time between RX and TX slots on intermediate nodes. All nodes on the path apply the same value for t_{tx_offset}, explained later in Section 5.6.

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 (see Figure 5.2.1). Since the source already discovered the route to the sink, nodes find next hops immediately, for example, they look up the routing table.

In this work, LETED benefits from the underlying DLDC-MAC and sends control frames, i.e., new TX times and ACKs, piggybacked in beacons.

5.3 Guard Times

Since LETED is a TDMA protocol, it suffers from the drift problem (see Chapter 3). That is, receivers may wake up too late because of clock drift and miss frames. Therefore, LETED applies the same solution as DLDC-MAC, i.e., based on drift prediction introduced in Chapter 6, 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 use guard times long enough to compensate drift.

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

5.4 Slot Synchronization

5.4.1 Problem Statement

Because of clock drift, timeslots of different nodes move relatively to each other (see Figure 5.4.1). If 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. Therefore, nodes need to apply a solution that prevents slots from moving or synchronizes them again.

5.4.2 Primary Solution

This solution demands an accurate estimation of multi-hop drift. Nodes adapt repeatedly the schedule according to relative drift to the source, i.e., the timeslots

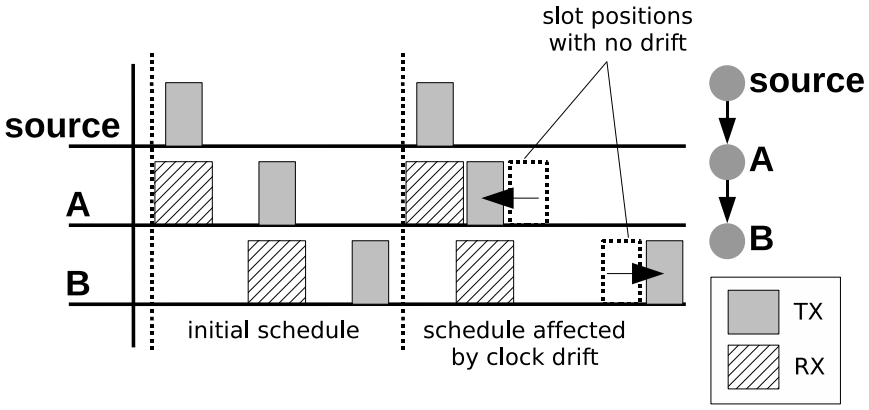


Figure 5.4.1: Because of clock drift, slots of different nodes move relatively to each other; in this example drift(A) < drift(source) < drift(B)

remain stable relatively to the source TX slots. As a result, the end-to-end delay remains constant, and timeslots 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.[1] Each node with a schedule sends to the next node its relative drift to the source repeatedly, piggybacked in DLDC-MAC beacons. On receiving relative drift to the source of the previous hop, nodes add to it drift of the sender (neighbor). In that way, each node estimates relative drift to the source.

Nodes shift LETED slots in the following way:

1. After an RX timeslot finishes, nodes calculate the time of this slot rx_{next} in the next beacon period as:

$$rx_{next} = rx_{now} + T_{beacon} + \delta_{src} \cdot T_{beacon} - g \tag{5.4.1}$$

   where T_{beacon} is the beacon period and, g is the guard time to the sender (neighbor). Clearly, nodes adapt the schedule according to relative drift to the source δ_{src}. The estimation of guard times is based on the MADC solution introduced in Chapter 6.

2. Nodes handle TX slot shifts in a similar way, i.e., they estimate the slot time in the next beacon period as:

$$tx_{next} = tx_{now} + T_{beacon} + \delta_{src} \cdot T_{beacon} \tag{5.4.2}$$

[1] DLDC-MAC measures run-time drift to neighbors each time a beacon is received

where tx_{now} is the time of the slot just finished.

5.4.3 Improvement

Risks of Previous Approach

The solution introduced in the previous paragraph works as long as nodes estimate exact drift to the source. However, drift estimation may suffer from various errors. For example, because of limited memory nodes should not store high precision values, leading to truncation errors.

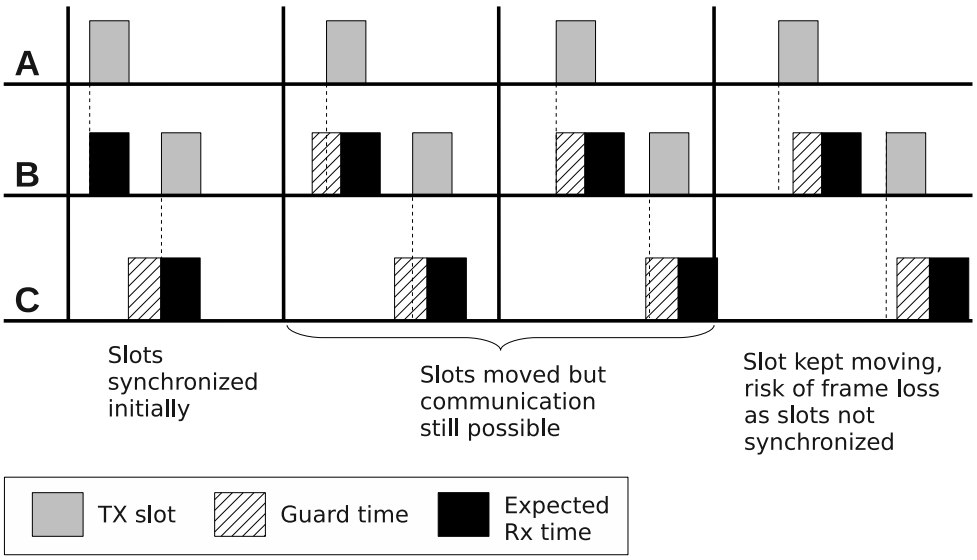


Figure 5.4.2: Nodes shift slots according to relative drift to the source. Because of errors in multi-hop drift estimation, they shift slots by different times. Thus, slots of senders and receivers drift away.

Because of potential errors in drift estimation, nodes may still suffer from the drift problem (see Figure 5.4.2). Nodes on multi-hop paths shift slots according to relative drift to the source, i.e., to node A in this case. Because of errors in multi-hop drift estimation, however, slots of senders and receivers move apart. Minor time differences between TX and RX slots are compensated with guard times. However, since slots keep moving apart, 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 5.4.2).

Solution

To solve this problem, nodes adapt the previous solution as follows. 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 $t_{expected}$:

$$t_{diff} = t_{expected} - t_{rx}$$

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.

To deal with the packet loss problem, nodes apply the ARQ protocol when sending SYNC frames. That is, if senders do not receive an RX acknowledgment, they send the SYNC frame again. Moreover, only on ACK reception senders shift the corresponding TX slots. Otherwise, upon a SYNC frame loss only senders would shift TX slots, but RX slots of receivers would not be changed, leading to loss of wake-up synchronization.

The simulations with the LETED protocol confirmed that the solution based on SYNC frames solves the problem of slot synchronization. Nonetheless, nodes may still lost wake-up time synchronization in unlikely cases, e.g., after long periods with broken links or when clock drift significantly changed. Therefore, it may be necessary to use slightly longer guard times for sending SYNC frames. However, the theoretical possibility of such risks and potential improvements will be examined in future work.

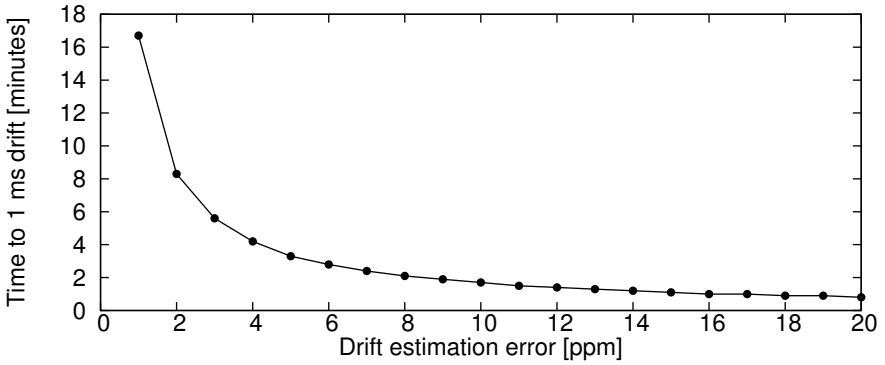


Figure 5.4.3: LETED slots move by 1 ms, because of errors in drift estimation, after the time depicted here

Clearly, the frequency of frame transmission depends on the scenario, e.g., the accuracy of drift estimation or changes in external conditions that affect drift. Figure 5.4.3 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 transmissions of SYNC frames, e.g., because of variable delays in software execution and on transceivers. Since delayed transmissions result in drift estimation errors, nodes should include TX timestamps in SYNC frames. By doing so, such 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 transmissions 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.

5.5 Overlap Risk

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 (see Figure 5.5.1. Besides, because of clock drift, LETED slots overlap with beacons of DLDC-MAC as well. Clearly, on timeslot overlap nodes may not receive data because of collisions. Therefore, nodes with LETED need to apply a new solution that tackles the overlap problem of LETED slots and beacons.

5.5.1 Overlap Detection

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 5.5.2 presents the problem. There are two independent routes to a sink, i.e., 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, if their slots overlap, as depicted in Figure 5.5.4, they do not detect it, and collisions occur. A similar case presents Figure 5.5.3 but both paths, i.e., 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

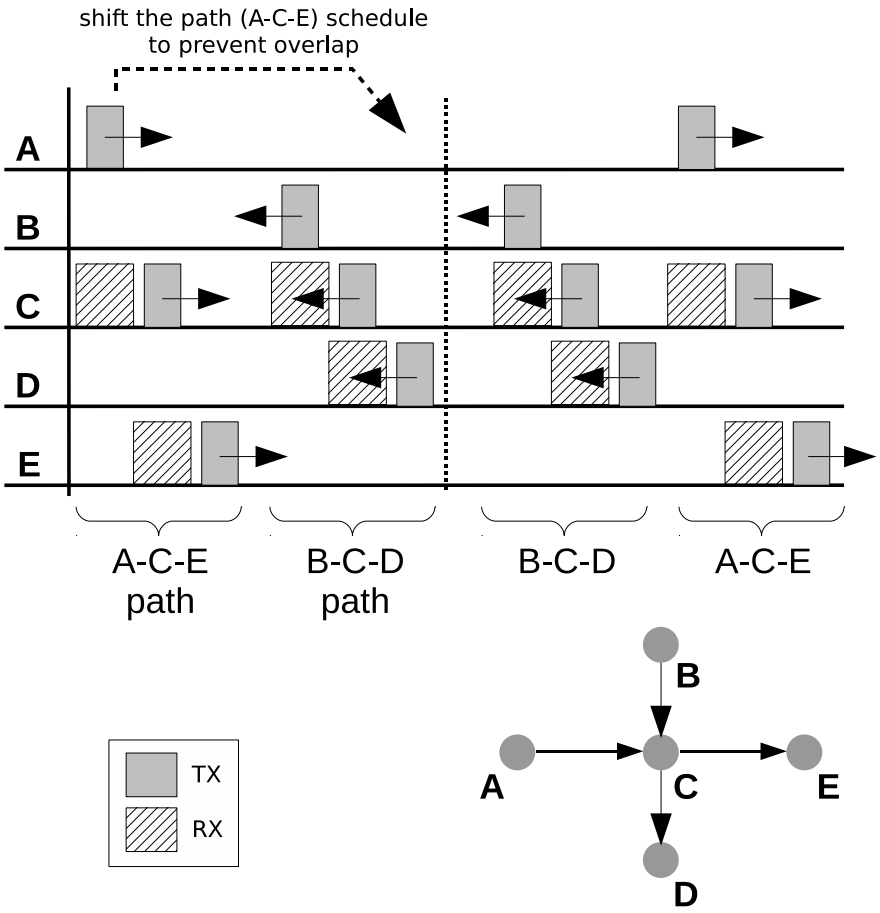


Figure 5.5.1: In LETED, schedules of different sources may move relatively to each other; in the case of overlap risk, one of the schedules is shifted

not receive data. Besides, if nodes send frames again 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 5.5.5 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.

The following paragraph presents a solution to the overlap problem, previously introduced in ref. [7]. However, simulation runs revealed some drawbacks of this approach. Therefore, this work introduces a simpler but a robust solution to the overlap problem based on the ARQ protocol.

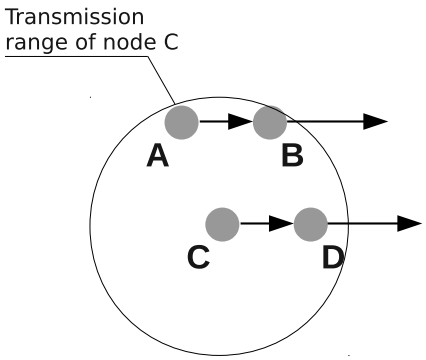


Figure 5.5.2: Two independent LETED paths are within their transmission range. As nodes do not learn about wake-up schedules of other paths, they cannot detect overlap risks between independent paths

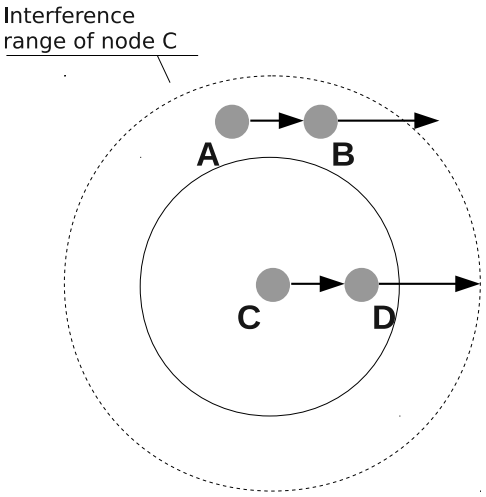


Figure 5.5.3: Although both paths are out of their transmission range, they are within their interference range and affect each other

5.5.2 Previous Solution

Timeslot - Timeslot Overlap

Nodes repeatedly collect information about LETED slots and beacons from nodes in two-hop neighborhood. By doing so, they should detect all overlaps within their transmission range. If they detect an overlap risk, they trigger the source node of the shorter path[2] to change its schedule by sending a shift request. The schedule change of the shorter path involves less effort. The request contains the time offset

[2]the network layer may provide the path length; otherwise, the node sends a path-length query to the source nodes.

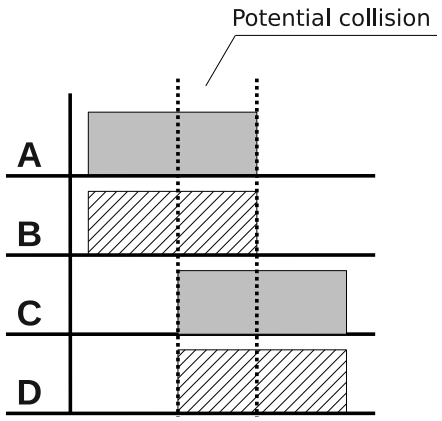


Figure 5.5.4: Schedule of paths A-B and C-D. Since nodes A and C transmit at the same time, there is a collision risk on receivers, on nodes B and D

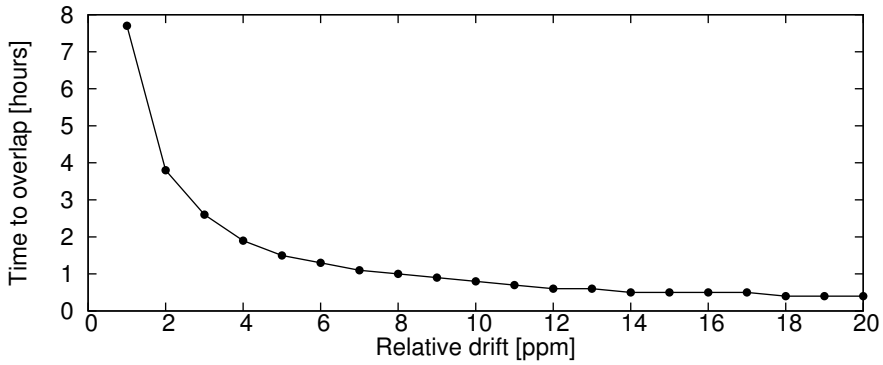


Figure 5.5.5: Average time to slot overlap, beacons or LETED slots, of nodes having four neighbors and a schedule supporting 10-second delays for different clock drift values

and relative drift of both affected schedules so the source can estimate the needed shift time.

On 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, which includes 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 5.5.1 presents the case when the nodes shift the affected slots to a later time.

Timeslot - Beacon Overlap

Beacons and timeslots may move relatively towards each other because of clock drift. If a node discovers that a beacon and a timeslot may overlap, it triggers the beacon sender to change its beacon time. Obviously, the beacon sender can be the node itself. Nodes shift beacons and not timeslots, since it results in less overhead.

Drawbacks

Although the solution solves the overlap problem, it suffers from the following drawbacks:

- Nodes may detect an overlap risk too late. First, slot shift takes a significant time, even several beacon periods, as a shift request must reach the source, but there is no wake-up schedule in this direction. Therefore, each node that forwards the request waits on average half a beacon period before sending the request. Second, if the slot shift would cause another overlap risk, nodes send back a negative acknowledgment, as introduced in Section 5.2. Clearly, it increases the total shift time as well. As a result, nodes may shift slots too late and do not prevent the overlap risk.

- To detect an overlap risk, nodes use a threshold time $t_{overlap}$. That is, if the gap between two slots is smaller than $t_{overlap}$, nodes start shifting slots. On the one hand, $t_{overlap}$ should be long enough to start slot shifts early enough. On the other hand, if nodes use too long $t_{overlap}$, they shift slots too often. To estimate a reasonable $t_{overlap}$, nodes need an exact time of slot shift. Unfortunately, nodes cannot estimate it exactly because of unpredictable delays in sending shift requests, as previously mentioned.

- With this solution nodes collect repeatedly wake-up schedules from 2-hop neighborhood. Clearly, dense networks or frequent updates results in many transmissions and waste energy.

5.5.3 ARQ-Based Solution

This approach aims to deal reasonably well with the overlap risk but remain 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 probably will 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, as introduced Chapter 4.

Each node detects an overlap by comparing start and finish times of slots stored in the slot table. As 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 tries to receive ARQ retries afterwards. Besides, if all affected slots are RX slots, the node switches the transceiver into the listening state for the time of both slots.

Table 5.1 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 use the 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.

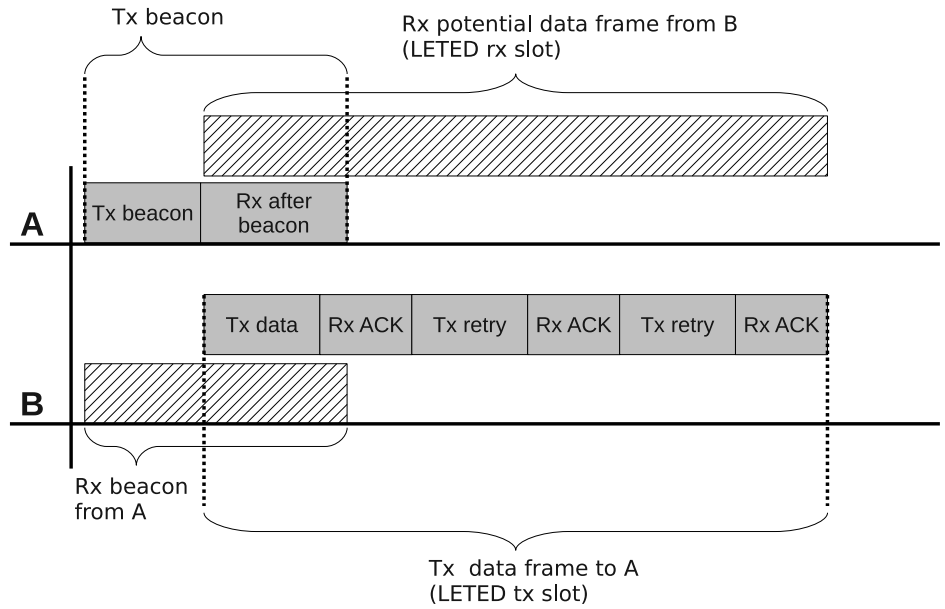
Table 5.1: Priority of slots in the ARQ-based solution to the overlap problem; in the case of overlap nodes skip or shorten the slot with a lower priority

Slot type	Priority
Beacon TX	4
Beacon RX	3
LETED TX	2
LETED RX	1

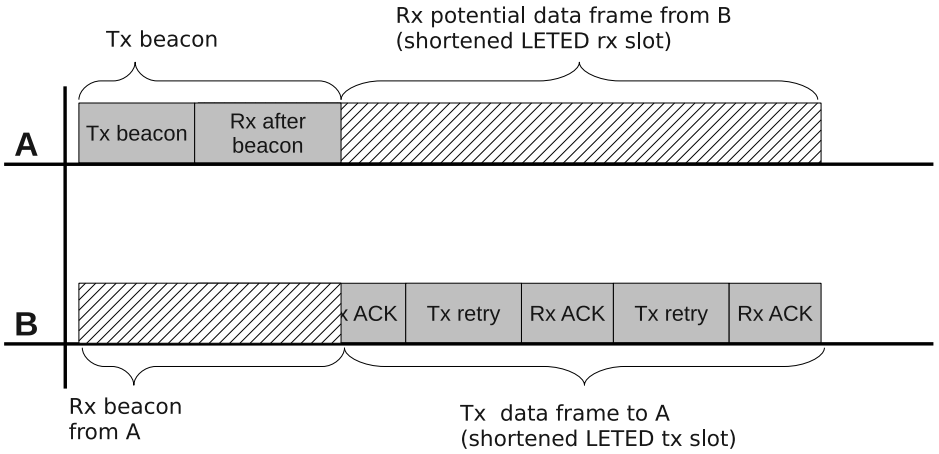
Figure 5.5.6 illustrates handling of slot overlap. In this case, node A detects an overlap of TX beacon and LETED RX slot. According to the slot priorities (see Table 5.1), nodes favor beacons over LETED slots. However, in this case, the beacon covers only the beginning of the LETED slot, the remaining ARQ retries are not affected (see Figure 5.5.6a). Therefore, the nodes do not skip the LETED slots but only shorten it (see Figure 5.5.6b). As a result, node A can receive data, although a LETED slot overlaps with a beacon.

5.6 Offset Between RX-TX Slots

As already stated, on intermediate nodes TX slots should follow almost immediately the corresponding RX slots to keep forwarding delay small (see Figure 5.1.1 and



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



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

Figure 5.5.6: Joining and shortening of slots on overlap risk

Eq. 5.2.2). However, the offset between TX and RX slots t_{tx_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 smallest t_{tx_offset} must compensate drift arisen between two consecutive slot shifts, i.e. during the beacon period T_{beacon} of DLDC-MAC. Thus, the smallest t_{tx_offset} is estimated as:

$$t_{tx\_offset} = T_{beacon} \cdot \delta_{worst} \tag{5.6.1}$$

where δ_{worst} is worst-case drift to the next node, specified for each oscillator type (e.g. ±20 ppm for Tmote Sky nodes). For example, Tmote Sky using DLDC-MAC with 1-minute beacon period needs t_{tx_offset} of at least 3 ms.

5.7 Topology Change

A routing protocol may change a source-sink path, referred to as re-routing, e.g., because of 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, like the 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 (see Eq. 5.2.1 and 5.2.2). Therefore, the application with rigid end-to-end delay requirements sets up a new schedule for the new path, i.e., starting from the source.

5.8 Energy Savings: Idle Listening Avoidance

This paragraph introduces a solution to the idle-listening problem of LETED. The solution exploits features of available transceivers, reduces the idle-listening time and prolongs the lifetime significantly.

5.8.1 Idle Listening of Software Solution

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 5.8.1a shows an active RX slot with a common software approach. After getting the preamble[3] and the following Start Frame Delimiter (SFD), nodes receive the payload. Then, the payload is delivered to the application, i.e., usually the transceiver drives the RX pin high, and the microcontroller (µC) triggers the RX 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

[3]Receivers use preambles to detect a new frame, the start and the end of frames and to synchronize bits and symbols

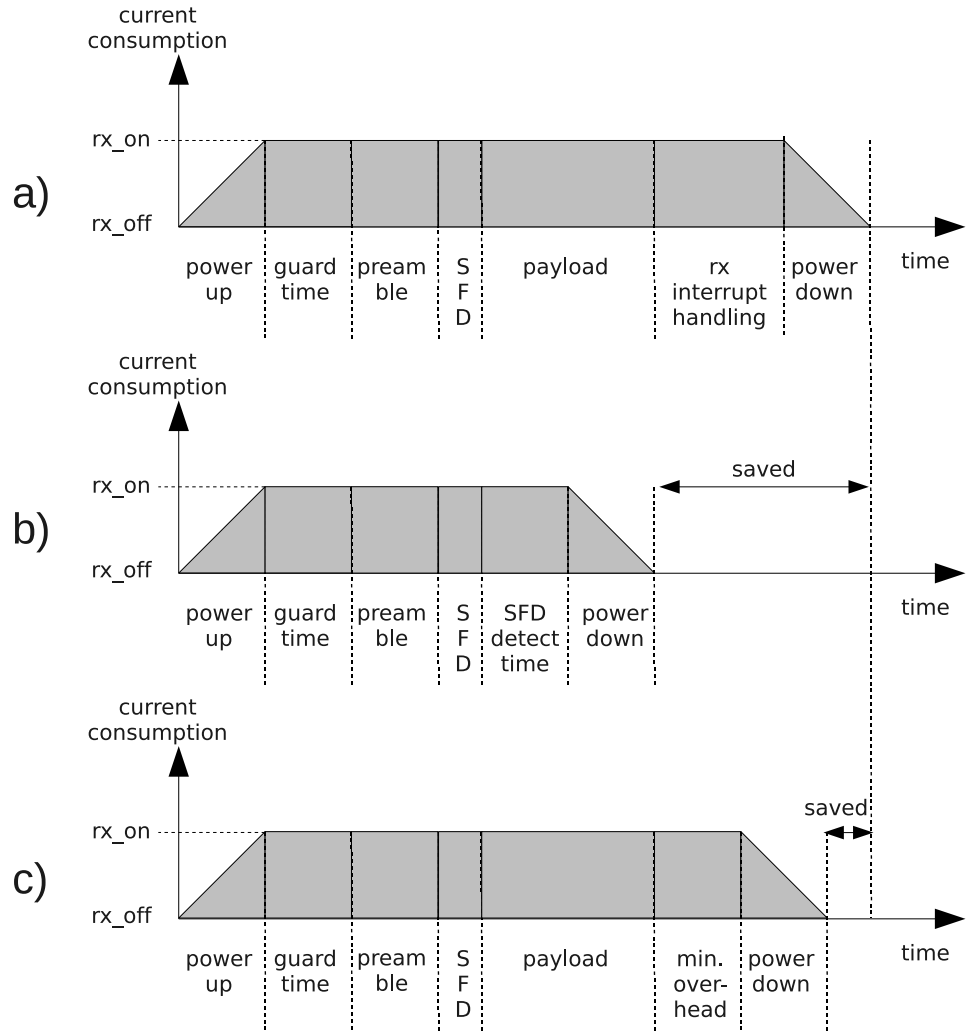


Figure 5.8.1: a) General receive slot slot (software only);
b) Shortened passive slot with ILA and ASIC
c) Active slots with reduced idle listening (ASIC only)

it to the application. Finally, the application calls an OS function to switch off the transceiver.

Figure 5.8.2 depicts the current consumption of various RX slot phases measured with an oscilloscope connected to a 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, i.e., 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:

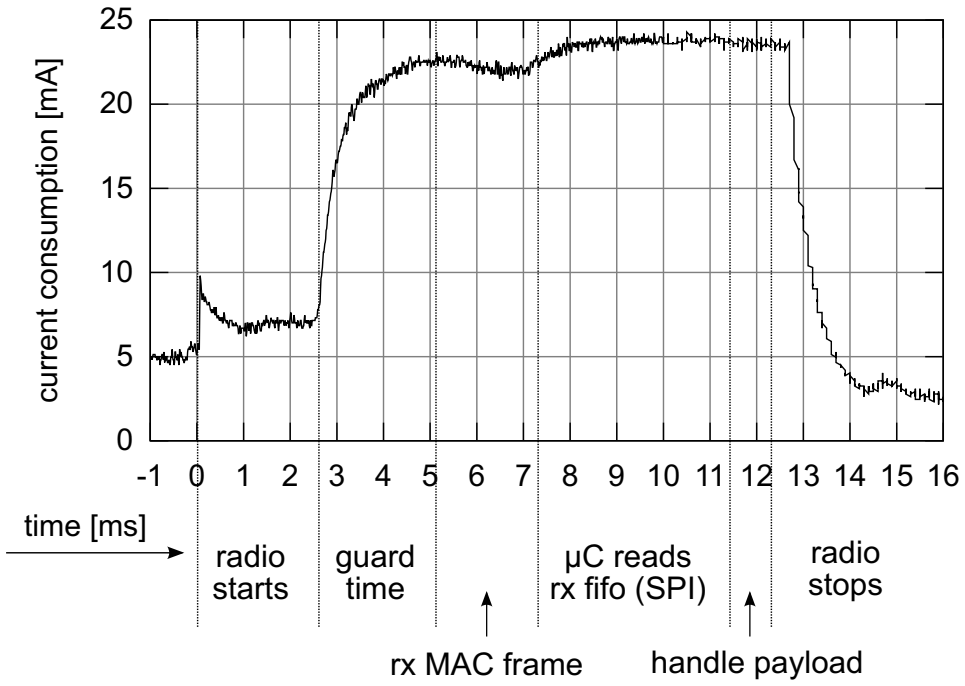


Figure 5.8.2: RX slot of Tmote Sky sensor node (oscilloscope output: average from 2 samples); MAC frame 62 bytes, data rate 250 kbps

- *Passive slots*

  Applications running on sensor nodes detect that a packet is received, when the operating system (OS) calls an RX routine, i.e., 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 until the OS calls the RX routine. The application powers down the transceiver, when this time interval has expired w/o any RX interrupt (see Figure 5.8.1a). However, handling of RxINT and getting a frame from the RX buffer may last much longer than the frame reception (see Figure 5.8.2). Besides, if the underlying protocols use frames of various lengths, the application considers the longest frame when waiting for RxINT. Obviously, indirect detection of idle RX slots causes excessive idle listening.

- *Active slots*

  After frame reception, the transceiver stays in the receive state until the µC powers it down, e.g., with ChipCon CC2420 transceiver [47] the µC writes a special command to a strobe register. Before software powers down the transceiver, it reads and handles the frame payload during ISR to learn whether other frames will follow. After that, it signals the µC to power down the

transceiver (see Figure 5.8.1a). Of course, if no frames follow the one just received, the transceiver should be powered down 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 powers down the transceiver. Thus, the software solution causes idle listening also in active slots.

5.8.2 Experiments

Some commercial transceivers offer extra features that can be exploited with LETED. For instance, ChipCon CC2420 [47], used in Tmote Sky [33] sensor nodes, captures the exact time of SFD and raises an interrupt when SFD is received.[4]. Nodes with LETED can exploit this feature to detect passive slots in an early stage and shorten idle listening. Next paragraphs present the solution in detail. This section presents the empirical results of Tmote Sky. First, it evaluates the time needed to handle SFD interrupt in the operating system. Second, it examines the frame reception handler.

The experiments were carried out on Tmote Sky sensor nodes running TinyOS [25] operating system. Tmote Sky consists of CC2420 transceiver (compliant to IEEE 802.15.4 standard) and MSP430 microcontroller (running with a frequency of 1 MHz in the experiments).

In the following experiments the times of SFD and of the RX interrupt are estimated by reading the timer register. However, reading the register takes some time as well and therefore can influence the measurements. Thus, the delay caused by reading of the timer register was estimated before other experiments. First, the initial value of the timer register t_{start} was saved. The sensor node read the timer register 1000x in a loop (a TinyOS function). The average time t_{read_timer} needed to read the timer register was estimated as:

$$t_{read\_timer} = \frac{t_{end} - t_{start}}{n} \tag{5.8.1}$$

where t_{start} is the timer register value before the loop, t_{end} the value just after the loop, and n equals to the number of timer read operations ($n = 1000$ in this example).

On the evaluated hardware, the average time needed to read the timer register equals to about 26 µs, which is less than a timer tick (the timer runs with 32 768 kHz frequency, i.e., with a tick speed of 30.518 µs). Thus, the following experiments

[4] 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

neglect the timer read delay, as it does not influence the measurements considerably.

SFD Detection Interrupt

This experiment measured the time T_{INT_SFD} needed from the SFD reception to the SFD interrupt handling in TinyOS (see SFD detect time in Figure 5.8.1b). The examined node received 1000 messages and collected as many T_{INT_SFD} samples. When CC2420 receives SFD of a new frame, it drives SFD pin high. Then, the µC captures the current timer value t_{SFD} and stores it in a register. In that way, the µC captures the SFD reception time precisely, that is, without any delay caused by software execution. Moreover, after SFD reception, the µC raises an SFD interrupt, and TinyOS executes the appropriate handler. The time t_{int} was captured in this handler. In that way, for each received frame the pair of timestamps $<t_{SFD}, t_{int}>$ was collected, and the time needed to raise SFD interrupt T_{INT_SFD} estimated as:

$$T_{INT\_SFD} = t_{int} - t_{SFD} \tag{5.8.2}$$

In this experiment, the time needed to raise the SFD interrupt was 3 ticks (approx. 91 µs) for all 1000 received messages.

As stated before, Tmote Sky raises another interrupt after the transceiver received the last byte of the frame. However, Tmote Sky uses the same pin and the same interrupt for SFD detection and for frame reception. In the first case, it detects a rising edge of the pin, and a falling edge in the latter case. Thus, raising a frame reception interrupt takes as long as SFD detection, i.e., about 91 µs.

RX Interrupt Overhead

This experiment estimated the time T_{RxINT} of the RX interrupt handler. It it the time that elapsed from the frame reception on CC2420 transceiver to the function call of TinyOS (see *rx interrupt handling* in Figure 5.8.1a). As the time needed to retrieve frame from the RX buffer depends on the frame size, two frame sizes were evaluated: 42 and 127 bytes. In this experiment 400 T_{RxINT} samples were collected for each payload size.

Like in the previous experiment, the µC captured the SFD reception time t_{SFD} for each frame. Moreover, the time t_{rx_TinyOS} was captured each time TinyOS executed the function for handling of received frames. TinyOS calls this function after it handles RxINT and reads the frame payload from CC2420[5]. The RX interrupt

[5]the µC raises an interrupt after receiving the first payload byte and not the whole payload. Thus, TinyOS starts reading payload bytes, while the payload is still being received. In that way, the

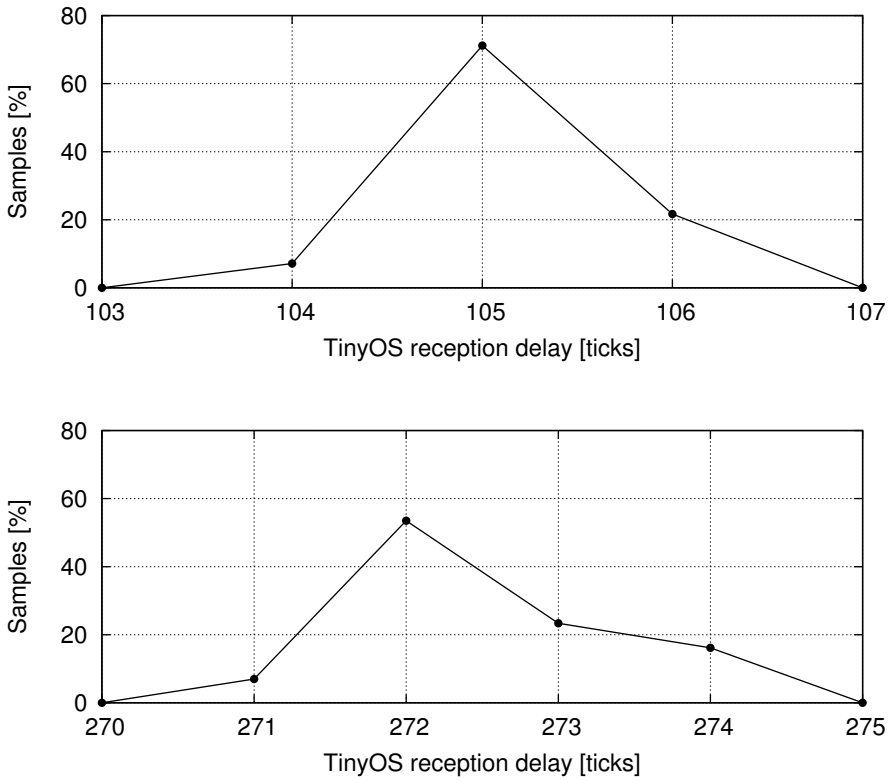


Figure 5.8.3: Frame reception delay on Tmote Sky with TinyOS with different payload size: 42 bytes (top) and 127 bytes (bottom)

handler overhead T_{RxINT} was estimated as:

$$T_{RxINT} = t_{rx\_TinyOS} - t_{SFD} - T_{frame} \tag{5.8.3}$$

where T_{frame} is the frame length in time units (about 44 and 133 ticks by 250 kbps data rate for both frame lengths).

Figure 5.8.3 depicts the experiment results. For the payload of 42 bytes, the shortest RxINT handler took 104 ticks (3.17 ms) and the longest 106 ticks (3.23 ms). However, the time was significantly longer for 127-byte payload: from 271 ticks to 274 ticks (8.29 to 8.35 ms). The reason for this is the time the µC needs to read data from the RX buffer of the transceiver. On Tmote Sky the µC gets frames from the transceiver using SPI (Serial Peripheral Interface Bus) with 510 kHz SPI clock. With such a clock frequency, the µC may receive 127 bytes in 2 ms, if the bytes are read one after another. However, the µC on Tmote Sky waits from 50 µs to 170 µs before getting another byte, which causes such long reading from the RX buffer, i.e.,

time needed for delivering frame to the software is shortened.

more than 8 ms for 127-byte frame instead of 2 ms.

5.8.3 Idle Listening Avoidance (ILA) Solution

The previous paragraphs introduced the experiments with Tmote Sky sensor nodes. It turned out that Tmote Sky handles the SFD interrupt in about 91µs. Moreover, handling the RX interrupt can take more than 8 ms. It is caused mainly because of a long time needed to read received frames from the RX buffer with SPI. Considering these results, this work introduces a solution that reduces idle listening of passive and active slots, referred to as Idle Listening Avoidance (ILA). Obviously, reduced idle listening prolongs the lifetime of nodes. The solution 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, it assumes that no frame arrives in this slot (see Figure 5.8.1b). The SFD detection time includes guard time, preamble, SFD itself, and the SFD interrupt handler. Since the detection time is short on Tmote Sky, less than 100 µs, nodes quickly power down the transceiver and shorten 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 5.8.1c). 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 the µC must retrieve it, which takes a few ms. Clearly, the node can already switch the transceiver to the idle state in the RX interrupt handler. Then, it will get the frame from RX buffer while the 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. Since it takes a few ms to start the transceiver, the node may miss the frame. Therefore, in this work nodes do not apply the solution to active slots.

5.8.4 ASIC Solution

The optimal solution for idle-listening reduction involves the use of an application-specific integrated circuit (ASIC), which causes the shortest delay in SFD detection and switching off the transceiver. Such a circuit shortens idle listening in the following way:

1. Passive slots
   Like in the ILA solution, ASIC should switch off the transceiver immediately if SFD is not received within a desired time (see Figure 5.8.1b). 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 quickly, i.e., 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 5.8.1c). In this case, nodes with ASIC solution switch 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 for comparison reasons, neglecting open issues.

5.8.5 Evaluation

This section examines the solution to Idle Listening Avoidance based on CC2420 transceiver and compares it with software and ASIC approaches. As stated before, LETED causes idle listening both in active and passive slots. The following formulas estimate idle listening according to the solution applied, i.e., ILA, software or ASIC.

Passive slots

Idle listening of nodes with LETED based on the software solution $T_{idle\_software}$ equals:

$$T_{idle\_software} = t_{guard} + t_{preamble} + t_{SFD} + t_{max\_frame\_len} + t_{rxINT}$$

where t_{guard} is the average guard time used to compensate drift, $t_{preamble}$ and t_{SFD} are the times needed to receive the preamble (4 bytes in IEEE 802.15.4) and the SFD

field (1 byte). Moreover, nodes cannot usually determine the length of the potential frame and therefore listen for the time needed to receive longest frame supported $t_{max\_frame\_len}$ (127 bytes in IEEE 802.15.4). Finally, nodes wait the time t_{rxINT} needed to retrieve the frame from the buffer, approx. 8 ms for 127-byte frame.

With the ILA solution, nodes still have to wait the time needed to receive a preamble and SFD. However, after that, they switch off the transceiver after the time needed to detect SFD t_{SFD_detect} (about 91 µs on Tmote Sky node). In this case, idle listening equals:

$$T_{idle\_ILA} = t_{guard} + t_{preamble} + t_{SFD} + t_{SFD\_detect}$$

ASIC solution cause similar idle listening to ILA, but the SFD detection time t_{SFD_detect} is shorter than the time of CC2420 transceiver.

Active slots

In active slots, nodes should switch off the transceiver immediately after they received and examined a frame. However, it takes time t_{rx_post} to get a frame from the RX buffer and thus idle listening of active slots equals t_{rx_post}: about 8 ms with software or ILA solutions, and a few µs with ASIC.

Results

To determine idle listening caused by LETED, this evaluation uses the energy consumption model introduced later in this work in Chapter 7. Besides, the evaluation also uses the hardware and scenario parameters, like energy consumption, from Chapter 7. For example, in this scenario an event occurs once an hour, which determines the number of active slots.

Figures 5.8.4 depicts the results of three solutions (Software, ILA, and ASIC) applied to LETED for various end-to-end delays. Clearly, the shorter is the guaranteed end-to-end delay, the more receive slots are needed, and the longer is the total idle-listening time. For example, with end-to-end delay of 5 seconds, the software approach causes 163 seconds of idle listening a day. In this case, the ASIC solution decrease 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.

Figures 5.8.5 present the corresponding energy gain of ILA and ASIC against the software solution owing to idle listening reduction. For example, nodes with ILA consume approx. 0.9 mAh/day less energy than the software solution with 5-second delays. According to the lifetime evaluation presented in Chapter 7, nodes w/o ILA

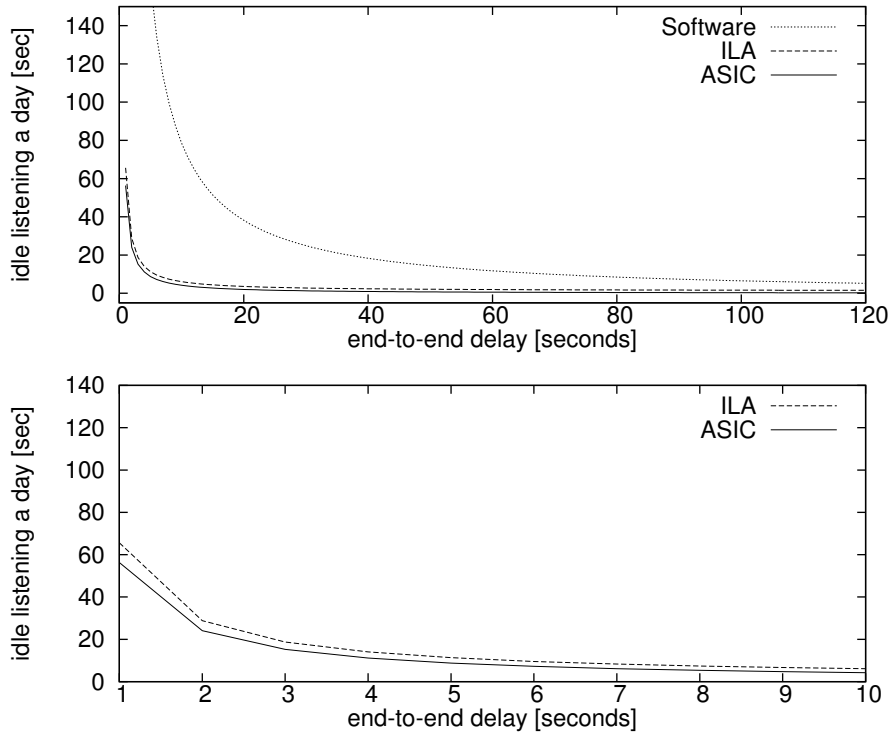


Figure 5.8.4: Idle Listening caused by LETED of three various solutions: Software, ILA (Idle Listening Avoidance) based on ChipCon 2420 and with a dedicated hardware (ASIC);
The diagram at the bottom zooms the results of ILA and ASIC

(DMAC approach there) consume approx. 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, the 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, with end-to-end delays of 5 seconds, the energy gain of ASIC is larger by only 0.008 mAh/day than ILA gain, which is less than 1% of total energy consumption.

5.9 LETED Evaluation

LETED with DLDC-MAC was implemented as cross-platform software and tested with OMNeT++ [49] simulator. This section introduces the simulation environment and discusses the results.

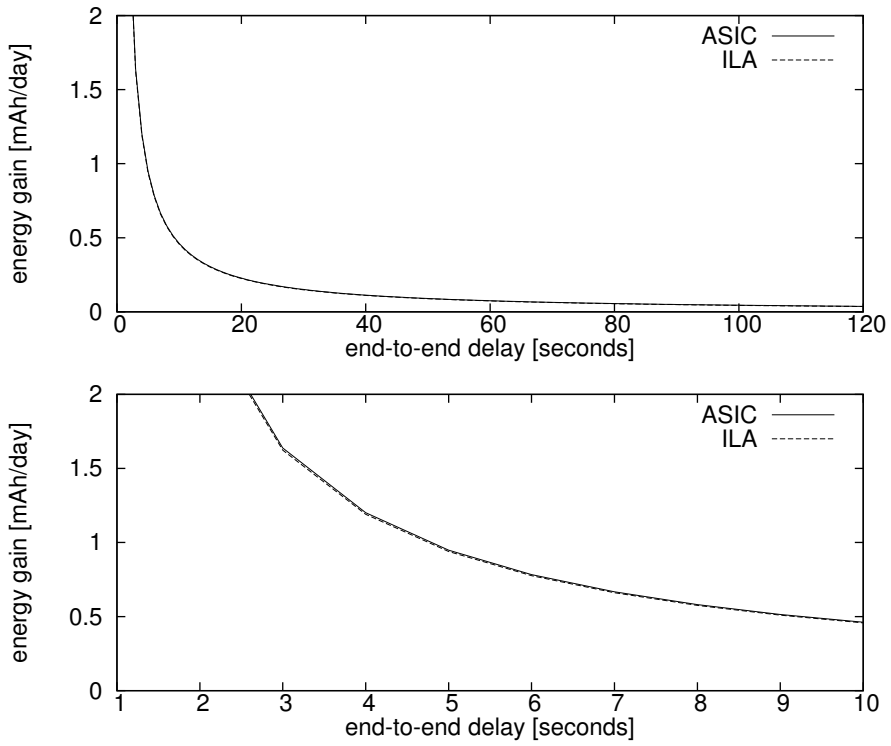


Figure 5.8.5: Energy gain of solutions to Idle Listening Avoidance based on hardware: ILA (with CC2420 transceiver) and ASIC (based on dedicated hardware);
The diagram at the bottom shows zoomed results

5.9.1 Simulation Environment

All simulations introduced in this paragraph were carried out with OMNeT++, a discrete event simulator, which gained in popularity in the last several years. OMNeT++ consists of modules written in C++. Owing to the modular design, the simulator can be easily extended with new models and features. For example, Mobility Framework (MF) provides several models for mobile wireless communication. For example, it simulates wireless channel at the physical level by considering signal strength, noise level, etc. 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, i.e., sensor nodes do not move. Another extension (INET) provides protocol models for TCP, IPv4, IPv6, Ethernet, IEEE 802.11 b/g, OSPFv4 and other protocols. Thus, OMNeT++ allows also simulations of complex heterogeneous networks, e.g., a sensor network connected with gateways to a Local Area

Network (LAN) or to the Internet.

This work utilizes another extension [17] that integrates Reflex [51] operating system (OS) with OMNeT++. Owing to this extension, all applications 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 Reflex OS but also on any other OS, if they provide suitable adapters. Such a cross-platform design for sensor networks was introduced previously in [8].

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

Integration with Reflex OS

Reflex OS is integrated with OMNeT++ with a *coroutine-based* model, i.e., the module code runs in its own thread and usually consist of an infinite loop with send and receive calls. 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 the 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 by the application previously. For example, if the MAC layer sets 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.

Bit Error Simulation

Mobility Framework (MF) takes 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 [40]:

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

where P_{tx} is the transmission power, λ is the wavelength, r is the distance between the transmitter and the receiver, and α is 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 [40]. Then, it estimates the Signal to Interference and Noise Ratio (SINR). The model considers the constant thermal noise parameter, defined in a configuration file, and the noise caused by consecutive transmissions of other nodes. It this way, MF discards frames upon collision, as consecutive frame transmissions result in noise levels higher than the frame RX power. Considering the SINR, the simulator estimates the bit error rate (BER) of the frame, according to the modulation used. For instance, the BER of binary phase-shift keying (BPSK) equals:

$$BER = \frac{e^S}{2}$$

$$S = \frac{-SNIR \cdot bandwidth}{bitrate}$$

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

$$P_{ok} = (1 - BER)^l$$

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}.

Clock Drift

The OMNeT++ simulator 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, OMNeT++ was extended to change the local time of nodes according to their 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 a node reads the system time, the simulator calculates the local time t_{local} of nodes according to their drift parameter:

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

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

5.9.2 Network Setup

Table 5.2: 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

The protocol stack of evaluated nodes depicts Figure 5.9.1. The application used services of the routing, mostly sending data over a multi-hop network. In this scenario, nodes used the AODV [35] routing protocol. As stated above, nodes used LETED and DLDC-MAC to limit end-to-end delays. All layers were implemented as a cross-platform application in ANSI C. Owing to the adaptation layer, software was tested and evaluated with OMNeT++.

In this scenario, the application was running only on the sources and on the sink. The intermediate nodes used only communication protocols, i.e., DLDC-MAC, LETED, and AODV.

On sources, 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 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 included the current simulation time, which the sink used to estimate the total delay of multi-hop communication.

The LETED implementation used two new approaches to the overlap problem. First, it used extra synchronization frames to keep LETED slots arranged along the path (see Section 5.4.3). Second, it allowed that LETED slot overlap but reduced the

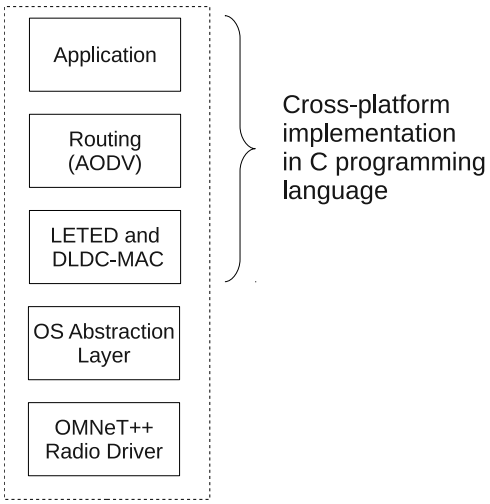


Figure 5.9.1: Protocol stack of evaluated nodes; owing to the cross-platform design, it runs on various operating systems; the OMNeT++ adapter allows the execution in the simulator

risk of frame loss in overlaps case by applying the ARQ protocol (see Section 5.5.3).

As stated above, OMNeT++ simulates frame loss according to the path loss model. Table 5.2 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 [47], 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 networks were evaluated with OMNeT++ simulator. Both setups are introduced in the following paragraphs.

Small Network

Figure 5.9.2a presents the small network evaluated with OMNeT++ simulator. Nodes 0 and 6 served as sources, they sent data to the sink, to node 5. As the nodes had to guarantee 10-second end-to-end delays, they applied a LETED schedule depicted in Figure 5.9.2b. Since nodes 3, 4 and 5 were on two gathering paths, i.e., 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 approx. 100 ms forwarding delay of a single hop. Thus, with a 5-hop

Table 5.3: 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

path to the sink, the nodes had to wake-up every 9.5 seconds to guarantee 10-second delays (details in Section 5.2).

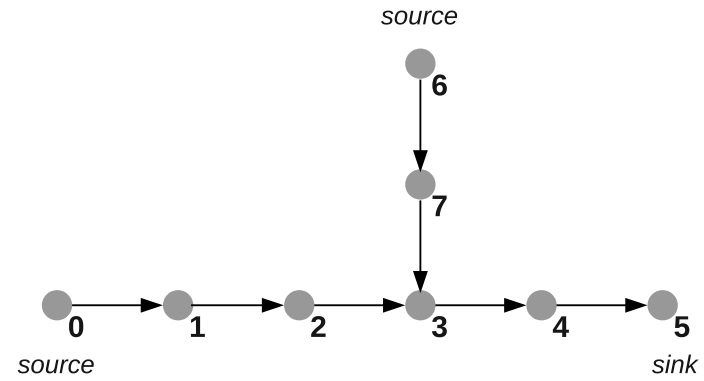
The thermal-noise parameter was adapted to get a PER of approx. 10%, which is higher than the PER of 4% observed in the outdoor experiment presented in Chapter 8. The reason for a higher PER is to test whether LETED works in worse conditions than expected. However, with such a PER the LETED protocol did not work correctly without the ARQ protocol. The problem stems from the way the slot synchronization works. In multi-hop networks, synchronization frames do not reach the sink because of bit errors. As a result, nodes close to the sink are not synchronized and miss frames transmitted in LETED slots (details in Section 5.4.3). 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 5.3). Each scenario was simulated 3 months.

Figure 5.9.3 shows the graphical user interface of OMNeT++ simulator with the network topology used in the evaluation.

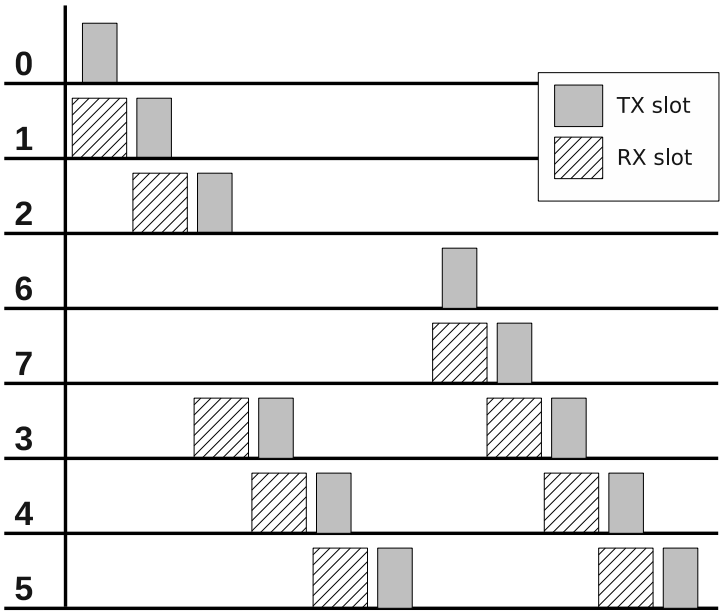
Large Network

The small network introduced previously considers a high PER, but the number of nodes is small. In such a setup, beacons and slots rarely overlap, and nodes do not handle it as often as 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 does 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 5.9.4 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



(a) Two sources (nodes 0 and 6) send data periodically to the sink (node 5)



(b) Nodes repeat the depicted schedule every 9.5 seconds to guarantee 10-second end-to-end delay

Figure 5.9.2: Topology and wake-up schedule of the evaluated network

Figure 5.9.4). 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 efficiently support 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 data dissemination from many sources is a part of future research.

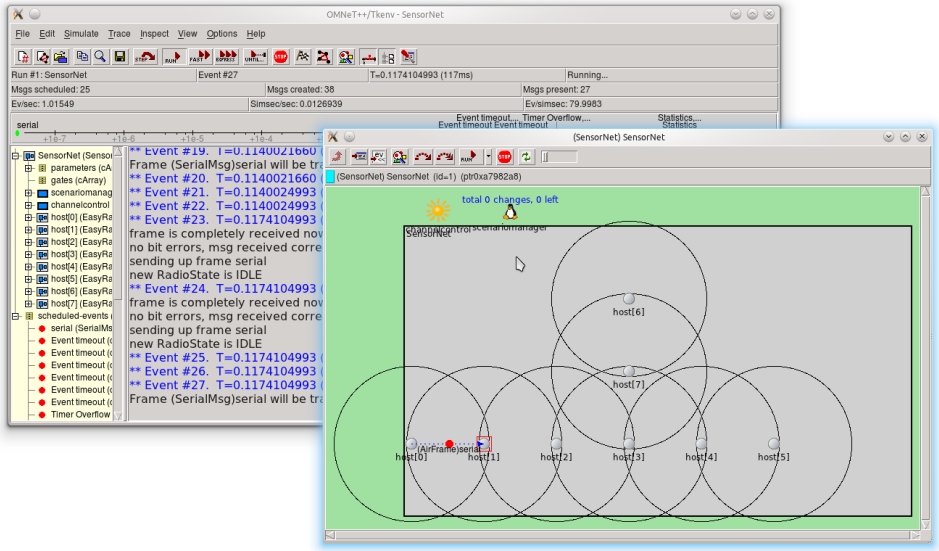


Figure 5.9.3: Screenshot of OMNeT++ simulator running the small-network experiment; the circles show the transmission ranges of nodes

In this case, nodes also support 10-second end-to-end delays, like in the small network. To counter the frame loss problem, nodes use the ARQ protocol with 1 retry.

5.9.3 Results

Packet Error Rate

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 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 that nodes missed because of slot overlap.

Figure 5.9.5 presents PER values of three scenarios in the small network. As expected, the nodes did not suffer from the frame loss risk in S1, i.e., with a PER of 0%. In two other scenarios the PER was close to 10%.

Figure 5.9.6 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, i.e., from 0% to 3.35%. In this scenario, links

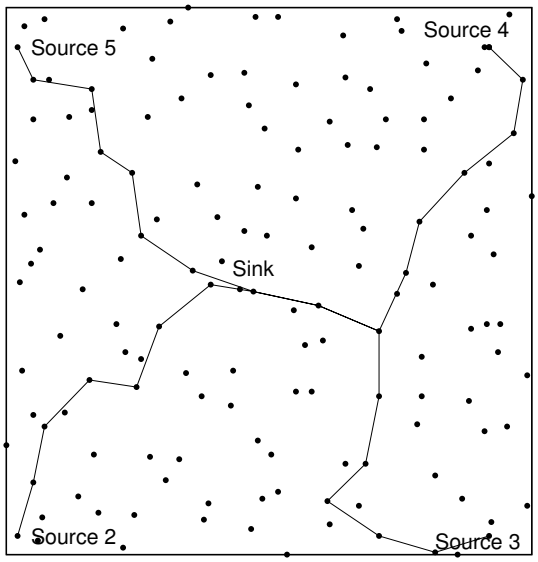


Figure 5.9.4: Evaluated network of 155 nodes
Four sources send data to the sink along the routes depicted in the figure

worked more reliably than in the real-world experiments introduced in Chapter 8. For example, node 50 had the worst-case PER of 3.35%, but the average PER of the outdoor experiment was about 5%. Thus, the LETED results of the large-network experiment are slightly better that in the real world.

Small Network

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

During all simulations, sources had to deliver event notices to the sink within 10 seconds. Figure 5.9.7 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, i.e., in approx. 10 seconds in this case. Bit errors are the second reason for frames reaching the sink too late. In this case, nodes use 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 experiment, nodes sent retries in the same slot and

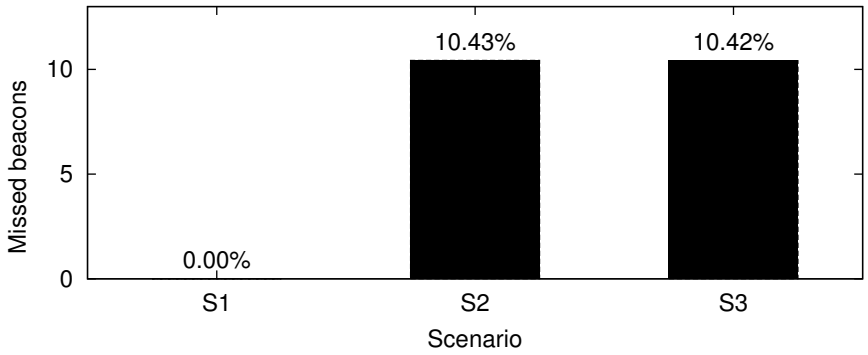


Figure 5.9.5: Small network: the total number of missed beacons among all nodes in three scenarios; as nodes did not apply ARQ for beacon transmission, it shows the average PER of links

ARQ did not influence end-to-end delays. Therefore, the sink received some frames too late because of skipped slots. This observation explains the results presented in Figure 5.9.7. 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 5.9.7 includes only frames received by the sink. However, here the success rate means the number of event notices received on time. Therefore, the success rate must exclude missed frames. Figure 5.9.8 presents the number of missed frames in all simulations. To obtain the success rate, the number of frames on time (see Figure 5.9.7) 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 transmissions were not affected by bit errors in S1, the sink missed 0.25% packets (see Figure 5.9.8). As stated above, the sink missed some frames, as nodes skipped some slots to reduce the overlap risk. It 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 in S1, i.e., 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 because of 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 in S1. Figure 5.9.9 shows that nodes in S3 skipped fewer slots than in S1 and 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 5.9.9 shows the average number of slots skipped, partly used and joined,

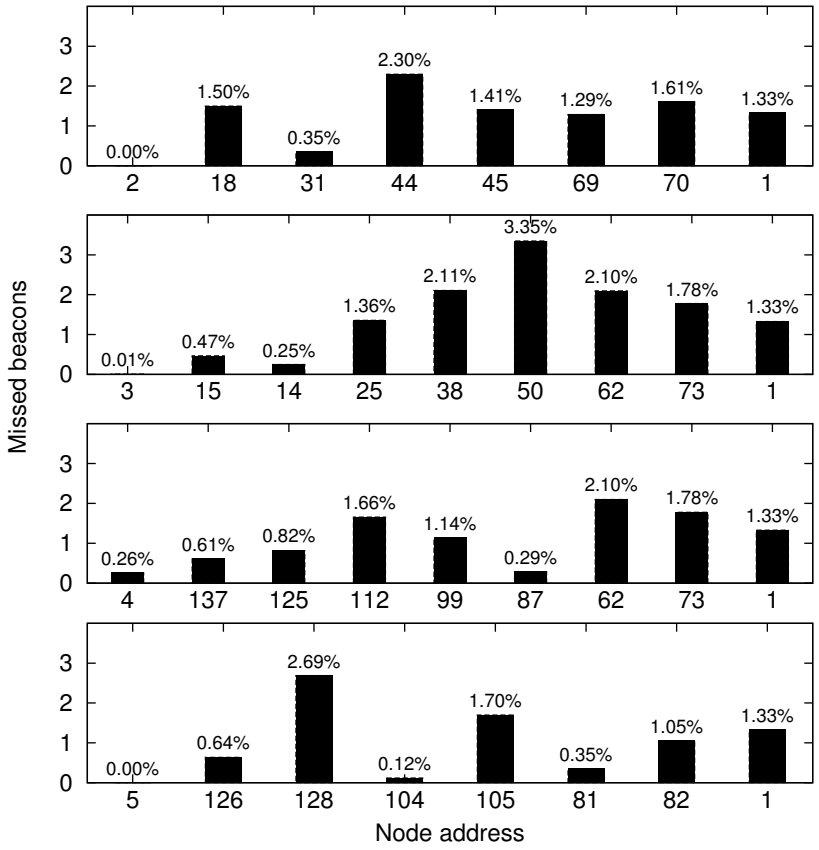


Figure 5.9.6: Large network: the average number of missed beacons (corresponds to the PER of links) for each route separately

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 it a similar number 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 in previous runs, when they were skipped. Therefore, the number of partly used slots in S3 is higher than in S1 and S2, 0.34% vs. 0.20%. Clearly, if more slots were used partly in S3 than in the previous runs, fewer 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 vs. 0.72% in S1 and S2.

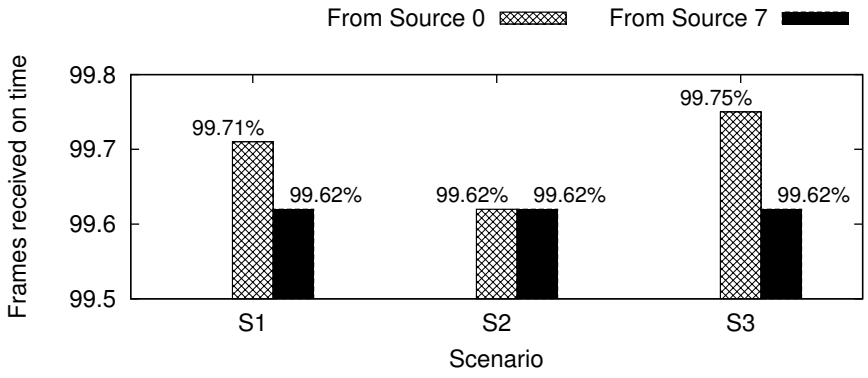


Figure 5.9.7: Small network: the number of frames (event notices) the sink received on time; it does not include missed frames

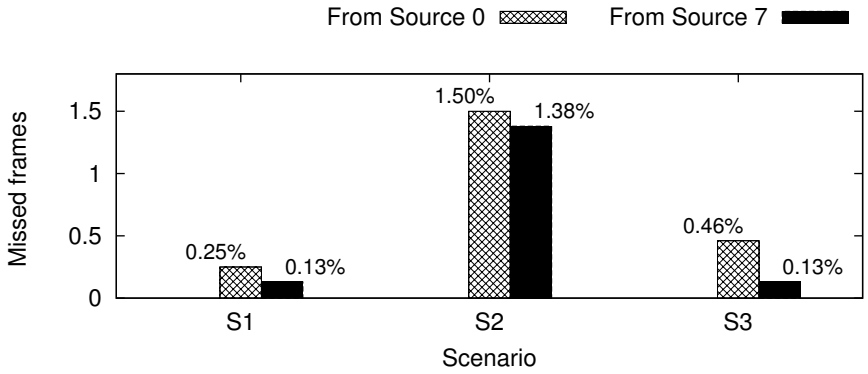


Figure 5.9.8: Small network: the total number of event notices that the sink did not receive, i.e., end-to-end packets, for each source separately

Large Network

Although nodes in the large network (LN) scenario suffered from a smaller PER than 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 5.9.10). In the worst case, it received 96% frames on time. Clearly, the main reason for worse PER results of LN is the higher overlap risk because of more neighbors than in SN. Besides, as nodes applied only one ARQ retry, it resulted in short LETED slots. Should beacons and LETED slots overlap, nodes skipped the latter ones and did not use them partly (details in Section 5.5.3). In this case, nodes send awaiting frames in the next slot, i.e., after about 10 seconds, and the sink received it too late. In the SN scenario, nodes skipped about 0.20% slots because

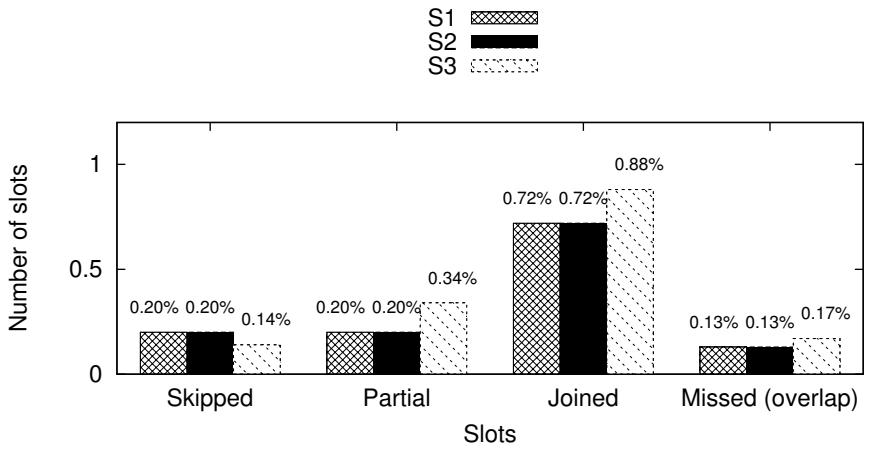


Figure 5.9.9: Small network: the number of slots skipped, used partially and joined upon an slot overlap risk among all nodes, for each test run separately

of overlap (see Figure 5.9.9). However, in LN nodes skipped more slots than in SN, i.e., from 0.45 to 0.65%.

Short LETED slots and a higher overlap risk affected also the total number of missed events. In the worst case, the sink missed 1.53% frames (see Figure 5.9.11), which is similar to the number of events missed in SN with one retry. However, in the latter case nodes suffered from a higher PER, i.e., 10% instead of 1-2%. As in LN nodes missed more slots because of overlap than is SN, it caused a higher frame loss rate. For instance, nodes in LN missed even 5x more frames because of overlap than in SN, i.e., 0.72% (see source 4 in Figure 5.9.12) and 0.13% (see Figure 5.9.9) respectively.

To achieve better performance, nodes might use the following solution. Should nodes use only a few ARQ retries, they prolong LETED slots by sending ARQ retries later and not consecutively. In overlap cases, 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: the success rate of all routes is more than 95%, i.e., the total number of frames received on time and not missed.

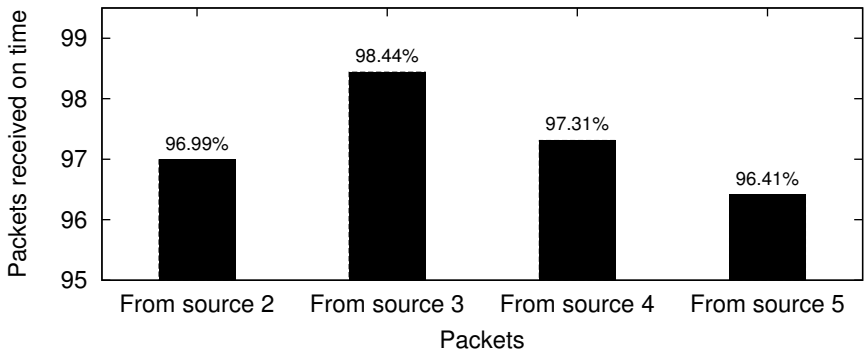


Figure 5.9.10: Large network: the amount of frame the sink received on time, i.e., within 10 seconds

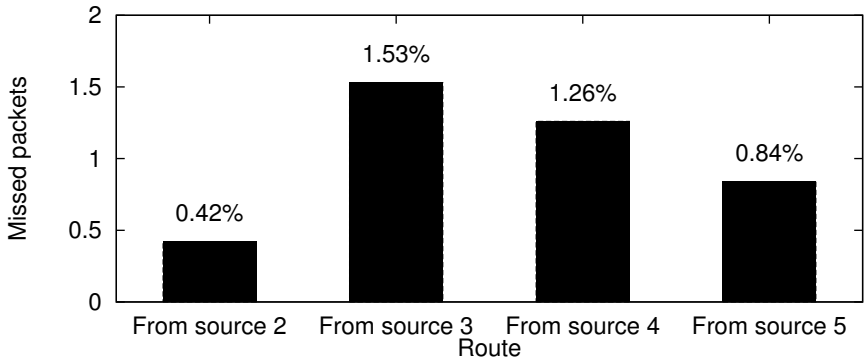


Figure 5.9.11: Large network: missed event notices

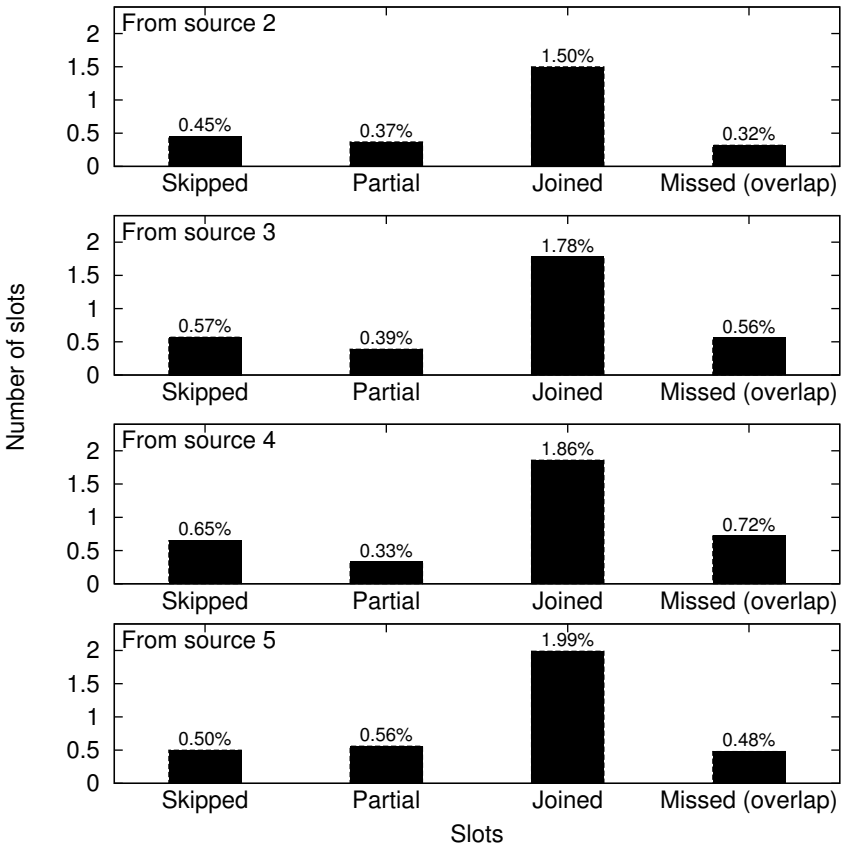


Figure 5.9.12: Large network: slots statistics for each route separately

6 Efficient Solution to Clock Drift Problem

Sensor nodes estimate the current time with crystal oscillators. The frequency of oscillators is time variable. Changes of temperature, air pressure, or electric supply voltage cause short-term variations of the oscillator frequency. Long-term variations are caused by oscillator aging. Besides, the same oscillator type may run with different frequencies. Thus, clocks of sensor nodes run at different speeds, referred to as clock drift.

Clearly, clock drift may lead to several problems in communication protocols. For example, nodes with TDMA approach may send and receive data at different times because of clock drift. Sensor nodes deal with the clock drift problem in scheduled protocols by applying guard times, e.g., they wake up earlier than the expected RX time to compensate drift to the sender. For example, nodes with DLDC-MAC wake up earlier by the guard time each time they expect to receive a beacon (see Chapter 4).

Since guard times result in idle listening, and shorten the lifetime, this chapter examines clock drift in sensor networks. First, it presents the results of a long-term drift experiment with sensor nodes running indoors and outdoors. Second, based on the experiment results, it provides solutions that keep guard times short. In this way, sensor nodes reduce idle listening, and work for a long time. Major parts of this chapter were introduced previously in [5].

6.1 Drift Experiment

6.1.1 Overview

The main goal of the experiment was to collect enough clock drift samples to evaluate various solutions to the drift problem. The experiment considered relative drift between sensor nodes, including all causes affecting it. For example, it included the time needed to power up the transceiver, which may not be constant, and the varying time of software execution. As clock drift depends on environmental conditions, e.g.,

temperature, pressure, the evaluation considered two types of nodes: with constant and with changing temperatures. That is, some nodes were placed indoors, and others outdoors. The latter ones were exposed to sunlight and varying temperatures. However, the sink temperature was constant throughout the experiment. In this way, drift measurements for various temperature differences between the sensor nodes and the sink were collected.

In this experiment, ten Tmote Sky nodes periodically sent beacons to the sink, once a minute, with the current temperature. A half of nodes were placed indoors, and another half outdoors. The sink was connected to a logging computer. On receiving a beacon, the sink recorded the RX time using its local hardware clock to get precise results. The sink delivered the RX time, with the sender address and the temperature, to the logging computer.

The sink was constantly powered up, and all nodes were located in a close vicinity of the sink (i.e., one-hop network). In this way, the sink could receive beacons of all nodes. Since the sink forwarded RX times to the computer, it did not have problems with the limited storage capacity, and recorded measurements for two weeks with a high frequency. The sink recorded drift samples with a frequency once a minute for each node, resulting in about 200,000 drift samples.

6.1.2 Results

Tmote Sky nodes use crystal oscillators of 32.768 kHz frequency with drift of ± 20 ppm. Thus, theoretically nodes need a guard time of approx. 80 ticks (2.4 ms) to compensate worst drift of a 1-minute period. However, the sink received some frames (less than 1%) with drift few times worse (more than 300 ticks) than the theoretical worst case. It shows that not only the inaccuracy of crystal oscillator affects relative drift, but also other factors (e.g., jitter in times needed to power up radio, software execution time, etc.).

As expected, the drift distribution of nodes working indoors appears Gaussian (see Figure 6.1.1). However, it does not hold true for outdoor environments (see Figure 6.1.2). The reason for that is the influence of the changing temperature on clock drift. The temperature of indoor nodes was constant, about 25 °C. However, the temperature of outdoor nodes changed from about 20 °C (during nights) to more than 50 °C (during daytime). As the sink was not exposed to sunlight, its temperature was constant, about 25 °C. Because of temperature variation, drift among the nodes and the sink changed. Therefore, the drift distribution of outdoor nodes is not symmetric and wider than the distribution of the indoor environment (compare Figures 6.1.1 and 6.1.2).

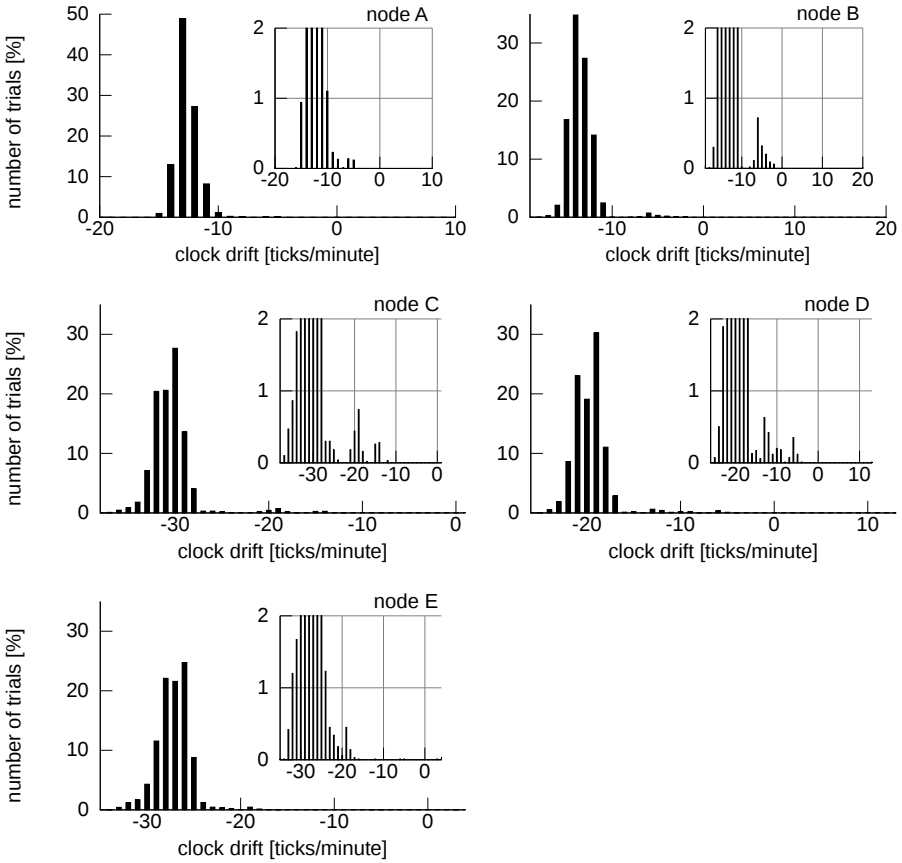


Figure 6.1.1: Drift distribution among indoor nodes; magnification in the right upper corners; 1 clock tick = 30.5 µs

The experiment revealed that nodes receive most messages in a small drift window. For example, the sink received on average 98% frames of indoor nodes in a window of 10 ticks. It is 30x smaller than the worst case including all factors affecting drift. Besides, worst-case oscillator drift results in 8x longer guard times. This holds true also for outdoor environments with changing temperature. In that case, 99% messages were received within a drift window of 40 ticks (see Figure 6.1.3). However, to receive the remaining frames (less than 1%) nodes needed much longer guard times, i.e., more than 300 ticks.

6.2 Solutions to Guard Times

Figure 6.2.1 shows a general TDMA approach with receivers responsible for dealing with clock drift. That is, sender and receivers agree on the next communication time, and keep their radios powered down until then. To compensate drift, receivers wake

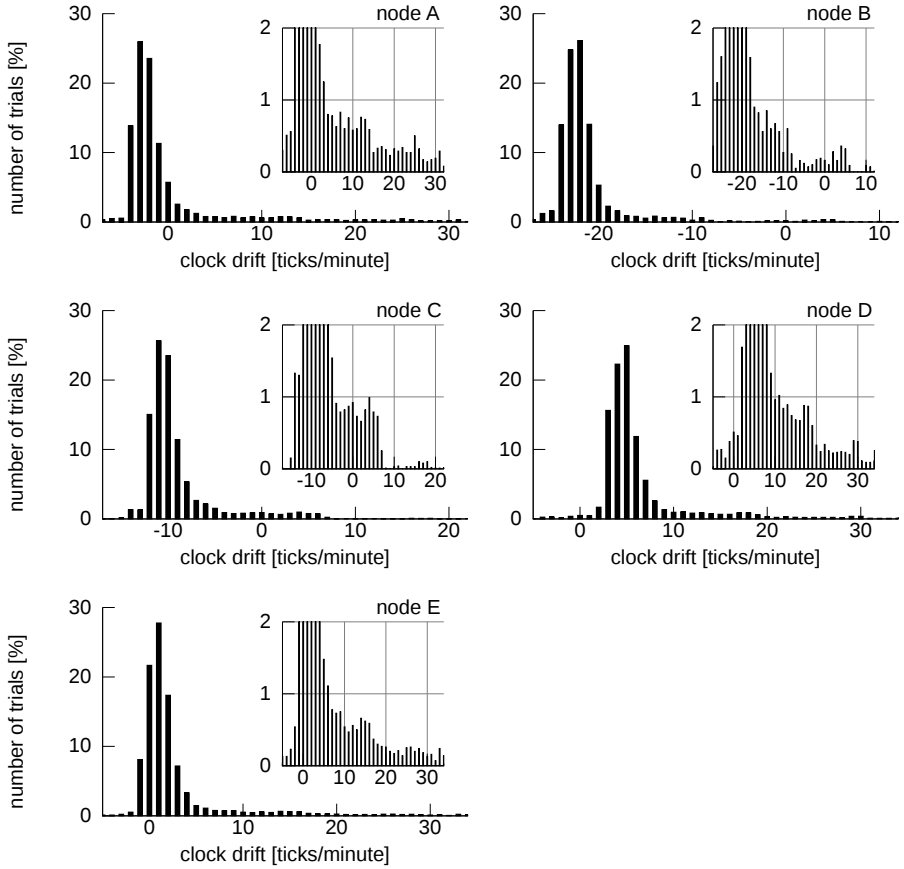


Figure 6.1.2: Drift distribution among outdoor nodes; magnification in the right upper corners

up earlier by the guard time, and listen until the frame arrives. Figure 6.2.1 also presents three major solutions to the drift problem based on guard times, introduced in the following.

6.2.1 Worst-Case Guard Time

With this solution, nodes use guard times that compensate worst-case drift since the last synchronization. For example, the crystal oscillator used in Tmote Sky [33] has the drift parameter of 20 ppm (parts per million). In the worst case, the oscillator drifts ±20 microseconds in a second against the perfect clock, and 40 microseconds between senders and receivers. In order not to miss frames because of clock drift, receivers estimate worst-case drift of the sleep period, and use it as a guard time. That is, they wake up earlier by this time. Some LDC protocols, like Dozer, use this approach. Since drift to senders should not be worse than the worst case, receivers

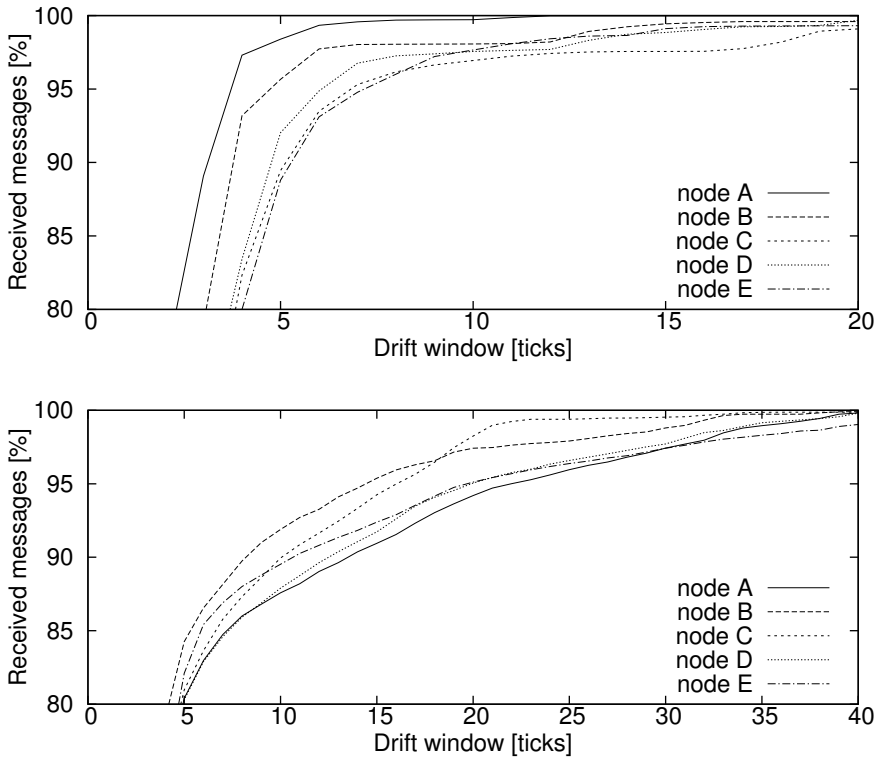


Figure 6.1.3: Amount of received frames in various drift windows, i.e. with guard times of various length; indoors (top) and outdoors (bottom)

will not probably miss frames because of clock drift. As depicted in Figure 6.2.1, the receiver expects the frame at a wrong time, shifted by the drift, and uses a long guard time to compensate drift. Tmote Sky needs about 24 ms guard time for a 10-minute sleep period, which is 8x longer than the frame length.

Despite long guard times, receivers can still miss some frames because of other factors that influence drift, e.g., jitters in the transceiver, and not only inaccuracy of the oscillator. The drift experiments revealed that some frames were received with drift larger than the theoretical worst case. Therefore, even when nodes consider the worst case drift parameter of the oscillator, there is no guarantee that such guard times compensate drift of all frames. Besides, guard times based on worst-case drift are typically unnecessarily long, and cause excessive idle listening (see the evaluation in Section 6.3).

6.2.2 Static (Short) Guard Time

As previously mentioned (see experiment results in Section 6.1.2), nodes receive most frames within a short drift window. It provides a new solution to the guard time

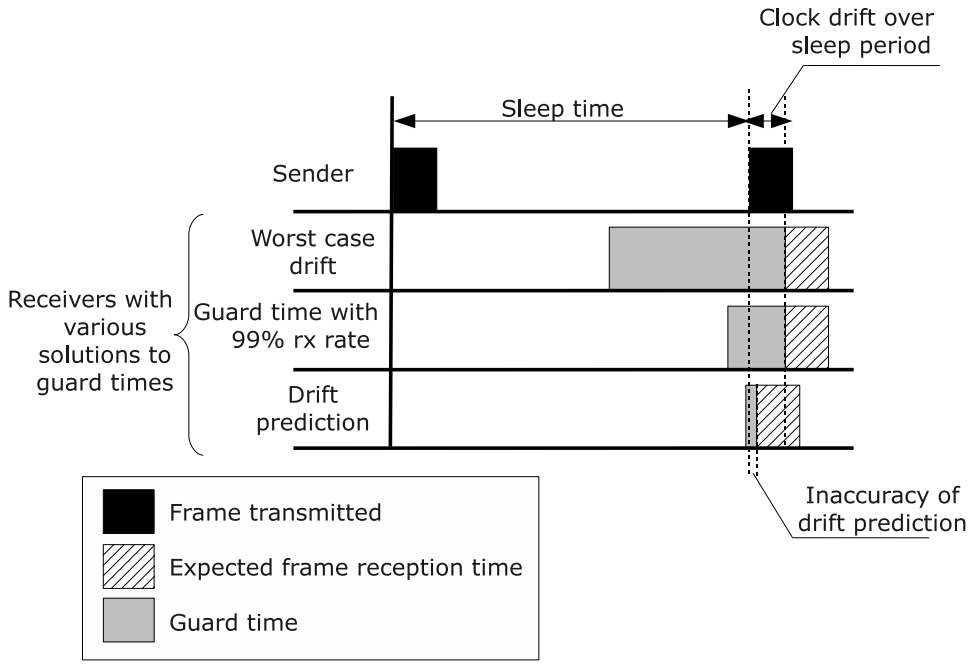


Figure 6.2.1: Three various solutions that estimate guard times

estimation. Should nodes deliberately give up the reception of some frames, they shorten guard times significantly. In this case, nodes estimate the length of guard times before deployment. For instance, Tmote Sky nodes need 10 ticks guard times to compensate drift of 98% frames indoors. Such guard times are 30x shorter than with the approach based on worst-case drift of oscillator. In this way, nodes reduce idle listening and prolong the lifetime. Figure 6.2.1 compares this solution to the approach based on worst-case drift. In both cases the expected RX frame is shifted by drift. However, with this solution guard times are shorter.

The solution works well in 10 out of 10 examined Tmote Sky nodes, indoors and outdoors. It would probably work well with other node types, since they use similar oscillators. However, as other oscillators may have slightly different drift characteristics, there is a need to find the static value of guard times needed to receive the desired number of frames, but still keeping guard times short. In addition, large-scale drift experiments should be carried out to evaluate the solution on more than 10 nodes, and on different hardware platforms.

6.2.3 Drift Prediction: Linear Regression

With solutions based on drift prediction, nodes estimate future drift to senders. To compensate prediction errors, nodes use guard times, which are shorter than the

solution based on worst-case drift (see the evaluation in Section 6.3).

Since clock drift is stable over a short time [30, 16], nodes predict drift by using linear regression. For example, with beacon-based protocols, they apply the ordinary least squares (OLS) method on previous drift samples and beacons. Nodes store n previous samples $< b, rx >$; b is the beacon sequence number (included in the beacon) and rx the beacon RX time. The next beacon reception time, based on the prediction of future drift, equals:

$$t_{next} = \beta_1 \cdot b + \beta_0 \tag{6.2.1}$$

where b is the expected beacon sequence number, β_1 and β_0 OLS parameters:

$$\beta_1 = \frac{n\sum_{i=1}^{n} b_i rx_i - \sum_{i=1}^{n} b_i \sum_{i=1}^{n} rx_i}{n\sum_{i=1}^{n} b_i^2 - \left(\sum_{i=1}^{n} b_i\right)^2} \tag{6.2.2}$$

$$\beta_0 = \bar{rx} - \beta_1 \bar{b} \tag{6.2.3}$$

where $\bar{b}$, $\bar{rx}$ are the average values of the beacon sequence numbers, and the corresponding RX times of n previous beacons.

Clearly, t_{next} may not be accurate enough, and nodes miss beacons. Thus, ref. [16] introduces confidence bands (guard times) around t_{next}:

$$t_{next} \pm [\Delta \cdot SE(t_{next})] \tag{6.2.4}$$

where $SE(t_{next})$ is the standard error of the predicted value, and Δ the scaling factor obtained empirically, e.g., during an initialization phase, to compensate prediction and estimation errors.

This work extends the OLS approach by defining Δ as the factor needed to compensate drift of a predefined number of frames, referred to as RX rate. The rationale behind this is the fact the majority of frames are compensated with short guard times, i.e., with small Δ. On the contrary, drift compensation of all frames needs excessive long guard times, and increases idle listening. Therefore, nodes using OLS can deliberately give up the reception of some frames, and use short guard times, reducing idle listening. The experiments revealed that Δ does not vary among sensor nodes, and can be either discovered during the initialization phase, or estimated before deploying the nodes. For example, the value of Δ needed to receive 99% frames varied from 2.7 to 3.1 among outdoor nodes.

To achieve high OLS precision, nodes collect previous drift samples, and must not miss any beacon. Thus, they use the worst-case guard times for some beacons, and cause significant idle listening. However, evaluation based on the empirical drift samples revealed that OLS still predicts future drift accurately, if it uses short guard

times and misses some beacons. For example, if nodes miss 1% beacons, the standard deviation of drift prediction was approx. 1 clock tick (30.5 µs) for 1-minute sleep period.

Although the adapted OLS shortens guard times, it causes several problems when applied on sensor nodes. OLS needs an emulation of floating-point arithmetic on microcontrollers. When using single precision, the truncation error can vary from ±17 µs to ±17.7 ms, according to [16]. When using more accurate, i.e., double-precision operations, OLS may take even 120 ms [16]. Besides, floating-point module and operations need extra memory, which is limited on sensor nodes. Thus, these drawbacks limit the use of linear regression approach on sensor nodes.

6.2.4 Moving Average Drift Compensation (MADC)

This paragraph introduces a novel approach to drift compensation for low duty cycle (LDC) protocols. Although some works applied the moving average filter to oscillator drift, e.g., Symmetric Clock Synchronization [46], they use it for different purposes, and do not address the problem of LDC protocols.

The main goal of this solution is to predict drift accurately, but with a small overhead in calculations, storage capacity, etc. MADC (Moving Average Drift Compensation) resembles the linear regression approach, but uses simpler mathematical operations to estimate future drift.

Drift Compensation

Nodes with MADC estimate future drift and its uncertainty in order to calculate guard times for the following beacon, like the approach based on linear regression (see the previous paragraph).

Nodes predict future drift δ_{avg} to senders by applying the moving average filter on the previous n beacon RX times rx_i:

$$\delta_{avg} = \frac{\sum_{i=1}^{n}(rx_{i+1} - rx_i)}{n \cdot T_{beacon}} \tag{6.2.5}$$

where i denotes the sample index starting from the oldest one. In this work, sensor nodes approximate Eq. 6.2.5 using 32-bit integers, which are accurate enough for MADC. Then, nodes estimate the next wake up time t_{next} to receive a beacon:

$$t_{next} \quad = \quad t_{last} + T_{beacon} \cdot \delta_{avg} \tag{6.2.6}$$

As run-time drift may vary from the predicted value δ_{avg}, nodes compensate it by

using guard times based on the drift jitter κ (expressed in ppm) as:

$$t_{next} \pm (m+1) \cdot \kappa \cdot T_{beacon} \tag{6.2.7}$$

where m is the number of consecutive missed beacons. According to Eq. 6.2.7, nodes use longer guard times after they miss a beacon.

Initialization Phase

If the drift jitter κ does not vary among nodes, as in the drift experiment presented previously, nodes estimate it before deployment. Otherwise, nodes must discover empirically the jitter needed to receive a predefined number of frames, e.g., during the initialization phase. An algorithm for the jitter discovery that does not need storing previous drift samples consists of the following steps:

1. Γ-array contains RX counters for predefined jitter values from κ_{min} to κ_{max}; each node fills the array with zeros at the beginning.

2. Nodes use the worst-case guard time to receive as many beacons as possible. On beacon reception, nodes increment the counter for all jitters κ that compensated current drift according to Eq. 6.2.7., i.e., $\Gamma[\kappa]$++

3. After receiving the predefined number of beacons Φ, each node finds $min(\kappa) : \frac{\Gamma[\kappa]}{\Phi} \geq \psi$, where ψ is the desired RX rate.

6.2.5 Evaluation of Drift Prediction Approaches

The evaluation considers the indoor and outdoor drift samples collected during the experiment. Both approaches, linear regression and moving average, were implemented as a standalone program. It iterated through drift samples, and provided the results.

Precision of Drift Prediction

This paragraph evaluates precision of both prediction approaches: MADC and OLS. It compares the standard deviation σ of drift prediction, i.e., the difference from real drift to the predicted one.

In the indoor environment, there were only minor differences between MADC and OLS (see Figure 6.2.2). On nodes A, C and D both approaches achieved similar results. OLS predicted drift more accurate on node E, whereas MADC was better

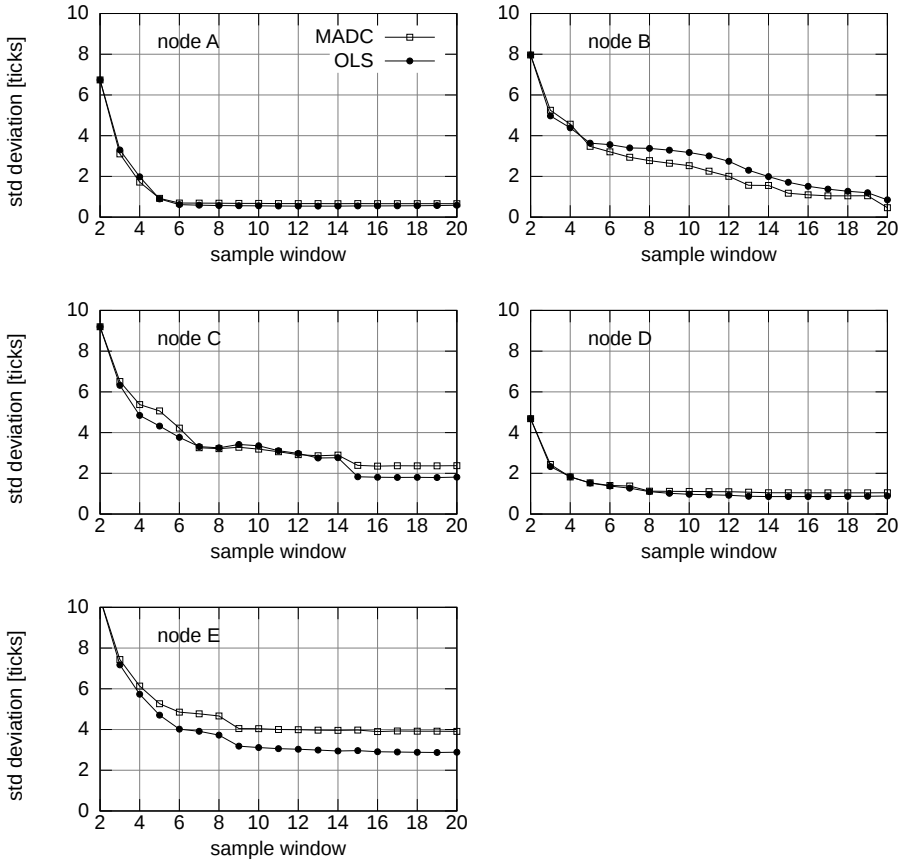


Figure 6.2.2: The prediction accuracy of moving average (MADC) and ordinary least squares (OLS) on indoor nodes

on node B. However, these differences are negligible, i.e., σ difference was smaller than a clock tick.

For small sample windows, MADC and OLS achieved similar results in the outdoor environment (see Figure 6.2.3). However, on 3 out of 5 nodes, accuracy of OLS prediction was worse with the increasing number of considered drift samples. For example, σ on node E with 7 previous samples was 2 ticks. Changing the number of previous samples to 20 resulted in almost 5 ticks σ. Since clock drift is stable over a short time [30, 16], nodes predict it precisely using linear regression. In other words, linear regression achieves precise results, if drift was stable in the past. In some cases (nodes A, D, E), however, since drift was not stable over long periods (10 minutes and more), OLS did not predict drift precisely. On the contrary, the moving average filter does not suffer from this problem. Therefore, MADC achieved high prediction accuracy on all examined nodes, indoors and outdoors. Besides, on some outdoor nodes, MADC predicted drift more accurate than OLS. For example, on node D σ

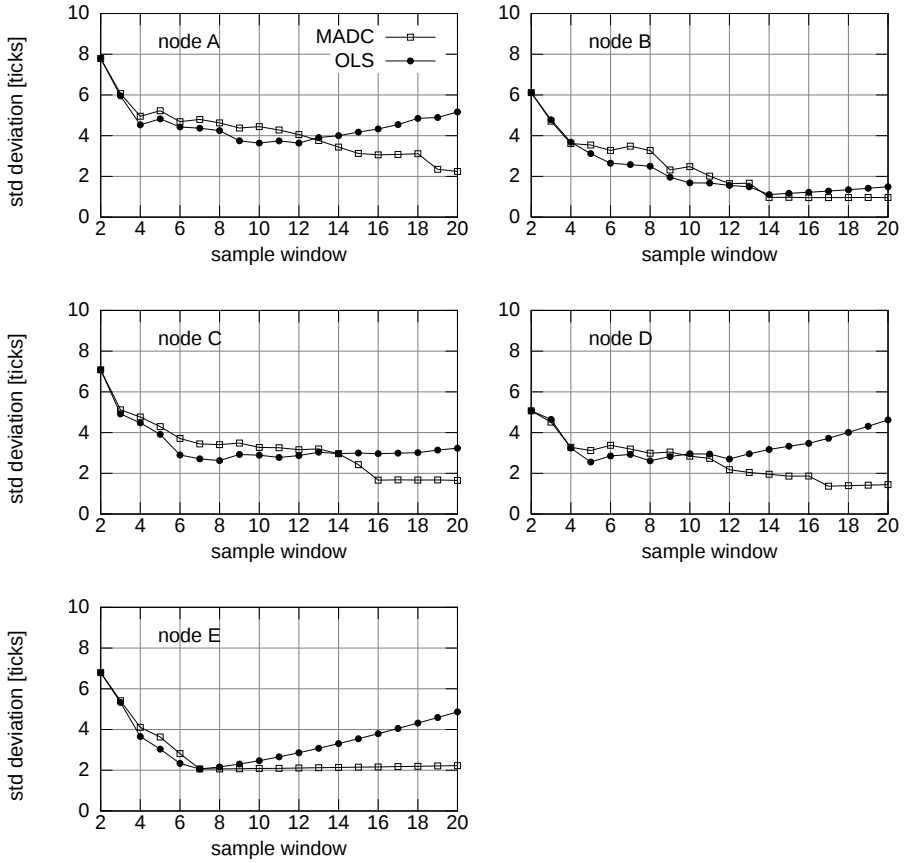


Figure 6.2.3: The prediction accuracy of moving average (MADC) and ordinary least squares (OLS) on outdoor nodes

of MADC was approx. 1 tick with 17 and more previous samples, whereas σ of OLS was 3 ticks and more (see Figure 6.2.3).

Although MADC achieves similar results to OLS indoors, and even better results on some nodes outdoors, it works with fewer resources and operations than OLS. Table 6.1 depicts the number of operations needed for a single drift prediction. MADC works well with integer values, whereas OLS needs floating-point arguments for mathematical operations, causing longer computations.

Impact of Samples History

Ref. [16] claims there is the best history window size of OLS that provides the most accurate drift prediction, i.e., the history window size of 8 previous samples (1-minute sampling period). However, the drift experiment presented in this work did not confirm the statement. On the contrary, there is no best history window in

Table 6.1: Complexity of a single drift prediction with n previous drift samples. MADC works with integer values, whereas OLS needs floating-point (FP) operation, i.e., it takes longer, consumes more energy and needs FP arithmetic module.

	ADD	SUB	MUL	DIV	SQRT
MADC	n+1	n-1	4	1	0
OLS	7n-2	3n+4	5n+5	4	1

indoor environments (see Figure 6.2.2). In general, the more samples nodes consider, the more accurate the prediction is. For example, on node B σ of OLS with 3 previous samples was 5 ticks. Doubling the number of previous samples decreased σ by a half. Only on some outdoor nodes there was the best sample window. For example, node A achieved the best result with 10 previous samples, but node E with 7 samples (see Figure 6.2.3).

Prediction accuracy is higher with the increasing number of past drift samples. However, there is a limit of prediction accuracy, i.e., any increase in the number does not improve the prediction. For example, the indoor node A predicted drift accurately (σ of approx. 1 tick) with only 6 previous samples (see Figure 6.2.2). Any increase in the number of considered samples did not improve prediction accuracy.

Initialization Phase

Table 6.2 presents the values of jitter windows (MADC) and scaling factors (OLS) that compensate drift of 99% messages. These values were obtained by considering all drift samples.

After 10 hours of initialization phase, MADC and OLS discovered jitter windows/scaling factors precisely, i.e., within ± 1 tick margin (see Figure 6.2.4). However, after 20 hours the estimated jitter windows of MADC got worse. This sudden change is caused by large clock drift on some nodes during 20-25 hours. It shows that nodes may overestimate the jitter windows, which leads to longer guard times. However, this problem is not addressed in this work.

Idle Listening

The longer guard times are, the fewer messages are missed because of not compensated drift. However, long guard times increase idle listening. Figure 6.2.5 presents the relation between the RX rate and idle listening of all indoor and outdoor nodes.

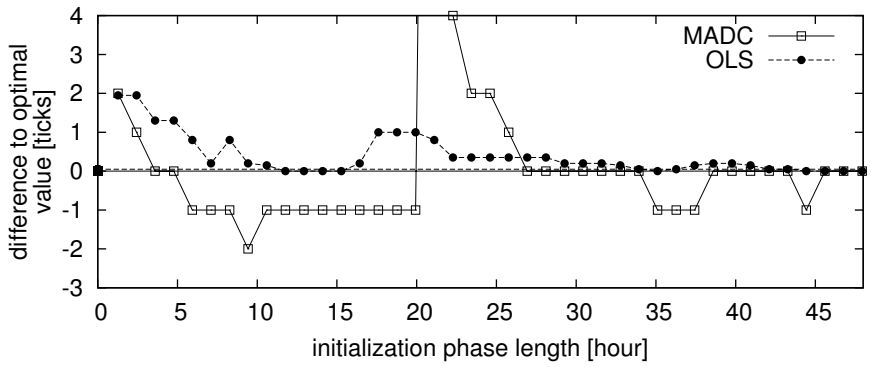


Figure 6.2.4: Time needed to estimate jitter windows/scaling factor; average among 5 outdoor nodes

Table 6.2: Jitter windows (MADC) and scaling factors (OLS) in ticks of Tmote Sky needed to compensate drift of 99% messages; 1 tick is approx. 30.5 µs; the table shows integer values rounded up

	node				
	1	2	3	4	5
indoor MADC	2	2	4	3	4
outdoor MADC	5	6	4	4	4
indoor OLS	1.7	1.4	1.8	1.1	2.2
outdoor OLS	3.1	3.0	3.0	2.9	2.7

Both approaches cause less than 1.5 tick (45 us) idle listening per frame when supporting 80% RX rate and less. Nodes achieve higher RX rates when using longer guard times, and it results in a slightly longer idle-listening time. For example, MADC achieves 99% RX rate and causes only 3 ticks of idle listening indoors. The approach based on the worst-case guard times causes on average 150 ticks of idle listening. However, it compensates drift of all frames.

OLS causes shorter idle listening than MADC. For instance, with 99% RX rate OLS causes 1.8 ticks idle listening indoors (MADC 3 ticks), and 3.6 ticks outdoors (MADC 4.3 ticks). Thus, although MADC may predict future drift more accurately than OLS, especially outdoors, it causes longer idle listening. The reason for that is the estimation of guard times. OLS uses guard times of various lengths, according to the standard error of prediction (see Eq. 6.2.4). In other words, if nodes predicted drift accurately in the past, they use short guard times. On the contrary, MADC uses guard times of fixed-length, even when the prediction was accurate. As a result, it uses longer guard times than OLS for some frames, and causes longer idle listening.

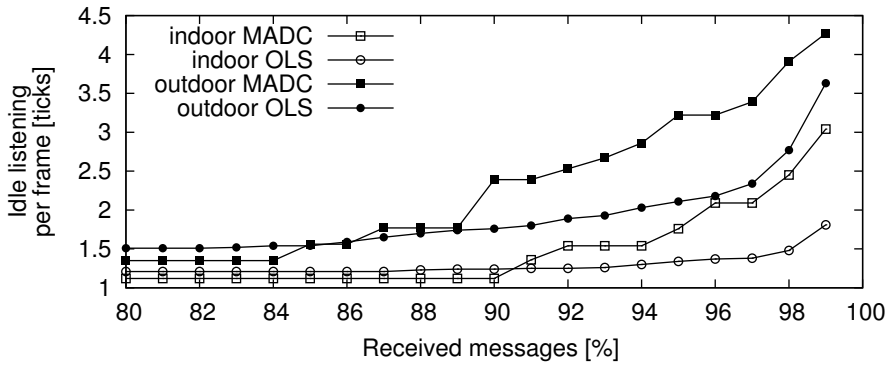


Figure 6.2.5: The longer guard times nodes apply (and cause idle listening), the less frames they miss because of not compensated drift; This figure shows the relation between the average idle-listening time per frame and the reception rate

However, such small differences in idle listening have almost no impact on the lifetime. The next paragraph presents the lifetime results in detail.

6.3 Evaluation

This paragraph evaluates idle listening caused by guard times (GTs) needed to compensate drift of 1-minute sleep period. The following approaches are evaluated (the number in brackets are theoretical drift values between senders and receivers based on the length of guard times):

- *Worst-case* (152.5 ppm); nodes estimate GTs based on very worst-case drift, i.e., including not only oscillator drift, but also other factors, like jitters in code execution. According to the drift experiment, this drift is at least 300 ticks on Tmote Sky nodes.
- *Oscillator Worst* (40 ppm); for the GT estimation nodes consider only the worst-case drift parameter of the oscillator, 40 ppm for a Tmote Sky node pair.
- *Static* (20.3 ppm); guard times compensate drift of approx. 98% frames, according to the drift experiment outdoors. The remaining 1% frames is lost because of not compensated drift.
- *MADC* (2.18 ppm); nodes compensate drift with MADC (Moving Average Drift Compensation), i.e., they predict future drift based on previous drift samples. In this case, MADC does not compensate drift of less than 1% frames.

- *OLS* (1.83 ppm); like MADC, nodes predict future drift and estimate GT. Here they use linear regression based on the ordinary least squares method. In this case, OLS misses 1% of frames because of not compensated drift.

Obviously, idle listening wastes energy, and therefore shortens the lifetime. Since guard times result in idle listening, their reduction affects the lifetime. That is, the shorter are guard times, the longer is the lifetime. To estimate the lifetime gain of various solutions to guard times, this evaluation uses the energy consumption model presented in Chapter 7. Besides, it considers the same scenario with the corresponding hardware and software parameters. In short, nodes form a multi-hop network and monitor the environment. When an event occurs, once an hour in this case, they have to notice the sink within 5 seconds. To support such an end-to-end delay, they wake up frequently according to the LETED approach. As they use guard times each time they wake up, the nodes consume a significant amount of energy to compensate drift. Therefore, shorter guard time can significantly prolong the lifetime.

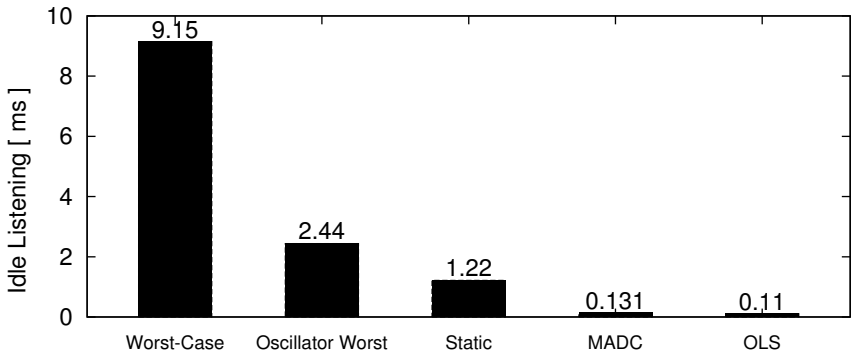


Figure 6.3.1: Idle Listening caused by guard times (various solutions to guard times compared) needed to compensate drift of 1-minute period

Should nodes compensate drift of all frames, i.e., *Worst-case* approach, they need GTs of more than 9 ms for 1-minute sleep period (see Figure 6.3.1). Such long GTs are more than 2x longer than the longest frame supported by IEEE 802.15.4 standard. Nodes with such an approach achieve a lifetime at least 2x shorter than other solutions (see Figure 6.3.2). It shows the GT estimation based on worst-case drift results mainly in enormous energy waste. Thus, it should not be applied to energy-efficient applications.

A common way to estimate GTs considers the worst-case drift parameter of the crystal oscillator. In this case, it results in 2.4 ms guard time. However, although this approach shortens GTs almost 4x against the very worst case, it still results in

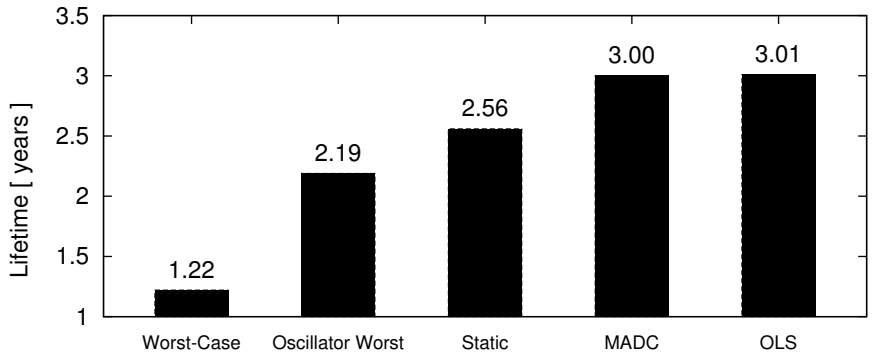


Figure 6.3.2: Lifetime of nodes with different drift-compensation approaches; based on LETED coupled with DLDC-MAC;
nodes support 5-second end-to-end delays in 5-hop networks

significant idle listening, i.e., 2.4 ms per frame.

The drift experiment revealed that nodes receive most frames in a short drift window. Thus, when nodes use shorter guard times, they prolong the lifetime, and still receive most frames. In this scenario, nodes used GTs of 1.4 ms and received 98% frames, depicted as *Static* in Figure 6.3.1. By doing so, they shortened GTs by a half against the approach based on worst-case oscillator drift. Since it decreased idle listening, the solution prolonged the lifetime from 2.19 to 2.56 years (see Figure 6.3.2).

The solutions to the drift problem based on drift prediction, linear regression or moving average in this case, outperforms other approaches. They shorten GTs almost 20x against the solution based on worst-case oscillator drift. It results in a lifetime gain of about 40%. It is because of short GTs provided by these solutions. On average, nodes need 0.1 ms long GTs to compensate drift, i.e., GTs are approx. 3% of the longest frame in IEEE 802.15.4 standard.

Although this evaluation use the model introduced in Chapter 7, MADC achieved a lifetime 3% longer than the results presented in that chapter. The evaluation of chapter 7 applied extra solutions to deal with multi-hop drift, and nodes consumed slightly more energy.

Clearly, the solutions based on drift prediction are not tailored to LETED only, but works with other MAC protocols as well. For example, the previous work [5] uses the MADC approach to nodes with the IEEE 802.15.4 MAC. Should nodes wake-up every minute to receive beacons, they prolong the lifetime 5% by using MADC instead of common solutions to the drift problem. Thus, any schedule-based MAC benefits from new approaches to drift compensation.

6.4 Summary

This chapter presented various solutions to drift compensation. It introduced MADC (Moving Average Drift Compensation), a drift prediction method that can be applied to sensor nodes owing to its simplicity. MADC shortens idle listening 20x against the usual solutions to drift compensation. Besides, it is not limited to DLDC-MAC and LETED. Any protocol that deals with the clock drift problem, like IEEE 802.15.4, can benefit from reduced idle listening provided by MADC.

7 Lifetime Evaluation

This chapter evaluates various solutions to short end-to-end delays. It introduces an analytical model for estimating the lifetime of nodes and compares the performance of LETED and DLDC-MAC with other duty-cycled protocols.

7.1 Overview

7.1.1 Scenario

Table 7.1: Parameters of the considered scenario

Parameter	Description	Value
λ_{frame}	data frame length	128 bytes
T_{event}	how often events occur	variable
n	hop count: source to sink	2, 5, and 10
d_{EtE}	maximum end-to-end delay	variable
$T_{mcuactive}$	how long the µC is active a day when radio is powered down	10 minutes

This evaluation considers sensor nodes that monitor some events, e.g., gas leakage. After source nodes detect such events, they send notice frames to the sink. Tables 7.1, 7.2 and 7.3 present the parameters of the scenario described below.

The lifetime of nodes depends on needed end-to-end delays, since it influences the duty cycle. Thus, the evaluation considers variable end-to-end delays, starting from less than a second. Since end-to-end delays depend on the hop distance between the source and the sink, three scenarios are differentiated: with 2, 5 and 10 hops to the sink.

In this scenario, nodes send small frames, only 128 bytes, after they detect an event. The frequency of events determines the number of transmissions. The more events sources detect, the more frames they send. Variable event periods are considered, ranging from 60 seconds to 12 hours.

Table 7.2: LETED and DLDC-MAC parameters

Parameter	Description	Value
t_{tx_offset}	the time a TX slot follows the corresponding RX slot in LETED	50 ms
t_{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
λ_{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%

After nodes with LETED received a frame, they wait for the time t_{tx_offset} before sending it to the next node. The smallest t_{tx_offset} must compensate drift arisen between the sender and the receiver over the sleep period. Therefore, t_{tx_offset} should be long enough to counter the drift problem. However, t_{tx_offset} affects end-to-end delay and duty cycle (see Eq. 5.2.2), and therefore it should be short enough. In this scenario nodes apply t_{tx_offset} of 50 ms.

To cope with errors occurring in multi-hop drift estimation, nodes send extra SYNC frames (details in Chapter 5). The SYNC period should be short enough to recover from multi-hop drift changes, caused mainly by temperature variation. In this scenario nodes send SYNC frames every 5 minutes. The value has been estimated during simulations with the LETED protocol introduced in Chapter 5, and it guaranteed slot synchronization with common clock drift rates.

Table 7.2 presents DLDC-MAC parameters too. Although the beacon period T_{beacon} equals 2 minutes, it does not affect end-to-end delays, since nodes use LETED slots for transmissions of event notices. As nodes with DLDC-MAC receive beacons from all neighbors, the number of neighbors affects duty cycle and the lifetime. In

Table 7.3: Hardware parameters of the energy consumption model together with values used for evaluation

Parameter	Description	Value
Q	available energy	1800 mAh
$I_{mcuactive}$	current consumption when the µC is active	2 mA
I_{tx}	current consumption when sending	20 mA
I_{rx}	and receiving	22 mA
I_{sleep}	current consumption when the node sleeps	0.01 mA
ϑ	transceiver data rate	250 kbps
$E_{selfdischarge}$	daily self-discharge rate of batteries	0.74 mAh
$E_{startup}$	energy needed to power the	7.2 nAh
$E_{shutdown}$	transceiver up and to power it down	4.2 nAh
$E_{txrxswitch}$	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
$\lambda_{preamble}$	preamble length	4 bytes
λ_{SFD}	the length of SFD (Start Frame Delimiter) field	1 byte

this scenario, nodes have four neighbors on average. When a beacon is missed, nodes apply a double-length guard time for receiving the following beacon. It results in longer idle listening, shortening the lifetime. The average missed beacon rate is as high as 1%, that is, the bit error rate (BER) equals about 10^{-6}.

Nodes can miss beacons and data frames in this scenario, as approaches for handling the frame loss problem are not adopted. These include ARQ and CSMA/CA, which are evaluated separately in Chapter 8.

Finally, Table 7.3 lists the hardware parameters of the *Tmote Sky* sensor platform. The parameters come from the datasheet [33] and from measurements. The values of current and energy consumption are given in mA and mAh units, and not in W and Ws, as hardware datasheets usually provide the former.

Tmote Sky nodes use two AA batteries as the energy source. Here they have 2x rechargeable batteries Sanyo eneloop [42], each with the total capacity of 2000 mAh. However, Tmote Sky uses about 80% of the available energy, as these batteries provide 80% of capacity with required 1.2V voltage. Besides, the scenario also considers the self-discharge of the batteries. Sanyo eneloop batteries lose about 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 use the drift prediction solution based on moving average (MADC), introduced in Chapter 6. This way relative drift among two nodes can decrease to approx. 2.18 ppm.

7.1.2 Evaluated MAC approaches

This analysis compares LETED and DLDC-MAC with the following approaches, which support low duty cycles:

1. Staggered schedule
2. Schedule based TDMA (Time Division Multiple Access)
3. Preamble Sampling

These approaches were previously discussed in Chapter 3, and therefore they are briefly introduced here.

Staggered Schedule

This chapter discusses the main idea of the staggered schedule independently of LETED, although the latter is based on the same method. By doing so, energy savings of LETED can be calculated and compared with the generic staggered schedule approach.

By applying the staggered schedule, nodes arrange their wake-up times to limit end-to-end delays. Shortly after receiving a frame, the node forwards it immediately to the next hop, as the next node is already in the listening state (see Chapter 3). Average end-to-end delays d_{EtE} equal to:

$$d_{EtE} = \frac{T_{sleep}}{2} + t_{frame} + (n-1) \cdot (t_{frame} + t_{offset}) \tag{7.1.1}$$

where T_{sleep} is the sleep period, n the number of hops on the path to the sink, t_{frame} the frame length, and t_{offset} is the time between the RX slot and the corresponding TX slot on each node.

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

To estimate energy consumption and the lifetime of nodes with the staggered schedule, LETED energy consumption was considered, but without the improvements of idle-listening avoidance (ILA).

Scheduled MAC

In schedule-based approaches, for instance, IEEE 802.15.4, Dozer, S-MAC or DLDC-MAC, nodes mostly sleep and agree on specific short wake-up times. Thus, nodes do not send data immediately after they detect an event, but wait till the next wake-up period (details in Chapter 3). It causes average end-to-end delays d_{e2e} equal to:

$$d_{e2e} = n \cdot (\frac{T_{sleep}}{2} + t_{frame})$$

where n is the number of hops to the sink, T_{sleep} is the sleep period, and t_{frame} is the frame length. To limit end-to-end delays, nodes adapt the sleep period:

$$T_{sleep} = \frac{d_{e2e}}{n} - t_{frame}$$

However, by doing so, the duty cycle of nodes increases, energy consumption rises, and the lifetime is shortened. Based on the DLDC-MAC, which is an example of schedule-based approach, this chapter examines the results of scheduled MAC protocols. Section 7.2.3 introduces DLDC-MAC energy consumption model. There are only minor differences in the energy model between DLDC-MAC and other scheduled-based MACs. Therefore, these protocols achieve similar lifetime results as DLDC-MAC.

Preamble Sampling

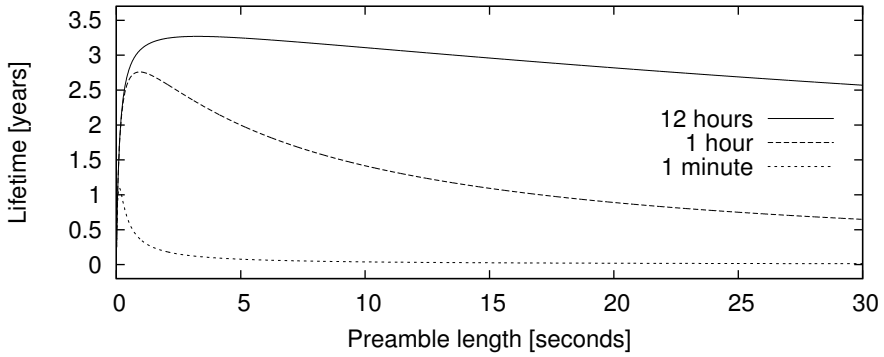
As already introduced in Chapter 3, protocols based on Preamble Sampling shorten delays by adapting the sleep period, although they were not designed to primarily support end-to-end delays. As the average forwarding delay equals the half of the sleep period T_{sleep}, which is the worst-case preamble length, the average end-to-end delay d_{EtE} of a n-hop path is estimated as:

$$d_{EtE} = n \cdot (\frac{T_{sleep}}{2} + t_{frame})$$

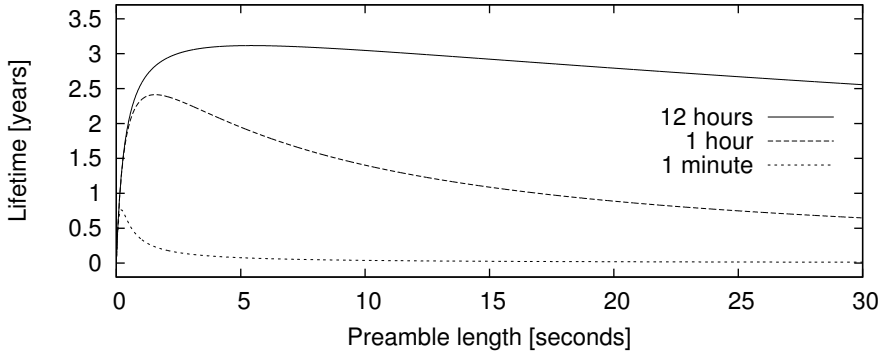
where t_{frame} is the frame length.

Preamble sampling causes idle listening on both senders because of long preambles, and on receivers because of periodic checks of the channel activity. Besides, if nodes detect a preamble on the channel, they remain in the RX state until the frame arrives, and it increases idle listening as well. Clearly, to prolong the lifetime nodes need to reduce idle listening by adapting the preamble length.

There is a tradeoff in preamble length. On the one hand, short preambles reduce idle listening of transmissions. On the other hand, the idle-listening time at receivers can be made longer with short preambles, as they need to often wake up. However, this might decrease idle listening too, as receivers do not wait long for data once a preamble is detected.



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



(b) TICER emulates long preambles by sending consecutive beacon frames; nodes require about a millisecond to receive a beacon and detect channel activity. Although TICER does not perform better than B-MAC, it works with all transceivers

Figure 7.1.1: There is the optimal preamble length in solutions based on Preamble Sampling; the results show the lifetime for various duty cycle, i.e., nodes send frames once a minute, once an hour and every 12 hours.

To find the optimal preamble length, the formulas introduced in the lifetime model were applied to the scenario mentioned previously. 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 7.1.1 presents the lifetime of B-MAC and TICER protocols. By adapting the preamble length, nodes balance idle listening in TX and RX states, so they find the optimal preamble. With an average TX period of a minute, B-MAC achieves the longest lifetime with a preamble of 120 ms (see Figure 7.1.1 and Table 7.4). Should nodes rarely send data, idle listening on senders becomes smaller. In this case, to balance idle listening in TX and RX states, nodes 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.

The main difference between TICER and B-MAC is the longer time that the former needs to discover channel activity. In this scenario, nodes with TICER expect "preamble beacons" of 30 bytes, and therefore they 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, the lifetime is reduced by 15%, if nodes apply TICER instead of B-MAC. However, the main advantage of TICER is that it works with every transceiver, as it sends ordinary frames. On the contrary, B-MAC needs a low-level access to individual bits of preambles. Besides, the transceiver supporting B-MAC must be able to transmit preambles of any length.

Table 7.4: The optimal preamble length and the lifetime (in brackets) for B-MAC and TICER

	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)

7.2 Energy Consumption Model

This section introduces the energy consumption model of nodes using LETED coupled with DLDC-MAC. Moreover, it provides a model for protocols based on Pream-

ble Sampling. Table 7.5 lists the symbols used in the model. The parameters related to the scenario were introduced previously (see Tables 7.1, 7.2 and 7.3).

7.2.1 Lifetime and Daily Energy Consumption

The lifetime of sensor nodes with LETED and DLDC-MAC is estimated as:

$$Lifetime = \frac{Q}{E_{day}} \tag{7.2.1}$$

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:

$$\begin{aligned} E_{day} &= E_{LETED} + E_{LDC} \\ & \quad E_{mcu} + E_{selfdischarge} \end{aligned} \tag{7.2.2}$$

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

7.2.2 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 only keeping the transceiver in the RX state. Therefore, LETED consumes energy while sending frames E_{tx_slots}, or when keeping the radio in the RX state E_{rx_slots}:

$$E_{LETED} = E_{tx\_slots} + E_{rx\_slots}$$

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

$$N_{active} = \frac{T_{day}}{T_{event}} \tag{7.2.3}$$

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

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

where t_{frame} is the expected frame length, and t_{tx_offset} is 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:

$$N_{passive} = \frac{T_{day}}{T_{slot}} - N_{active}$$

As nodes send during active slots only, the total TX energy is estimated as:

$$E_{tx\_slots} = N_{active} \cdot (t_{frame} \cdot I_{tx} + E_{startup} + E_{shutdown}) \tag{7.2.4}$$

where I_{tx} is the transceiver current consumption while sending, $E_{startup}$ 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:

$$t_{frame} = \frac{\lambda_{frame} + \lambda_{preamble} + \lambda_{SFD}}{\vartheta} \tag{7.2.5}$$

where λ_{frame} is the expected data length, and ϑ is the transceiver data rate. t_{frame} also includes the preamble $\lambda_{preamble}$ and the Start Frame Delimiter λ_{SFD}.

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

$$t_{rx\_active} = t_{guard} + t_{frame} + t_{rx\_post} \tag{7.2.6}$$

where t_{guard} is the guard time, t_{frame} is the average frame length, and t_{rx_post} is 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 Eq. 7.2.16).

By applying the Idle Listening Avoidance (ILA) solution, introduced in Chapter 5, nodes can shorten passive slots to:

$$t_{rx\_passive} = t_{guard} + t_{preamble} + t_{SFD} \tag{7.2.7}$$

where t_{SFD} is the time needed to detect the Start Frame Delimiter of incoming frames, and $t_{preamble}$ is the preamble reception time together with the SFD field. $t_{preamble}$ is estimated like t_{frame} (see Eq. 7.2.5).

As stated before, nodes with LETED consume energy E_{rx_slots} while receiving in both active and passive slots:

$$E_{rx\_slots} = N_{active} \cdot (t_{rx\_active} \cdot I_{rx}) + N_{passive} \cdot (t_{rx\_passive} \cdot I_{rx}) + (N_{active} + N_{passive}) \cdot (E_{startup} + E_{shutdown}) \quad (7.2.8)$$

$$E_{rx\_slots} = I_{rx} \cdot (N_{active} t_{rx\_active} + N_{passive} t_{passive}) + (N_{active} + N_{passive}) \cdot (E_{startup} + E_{shutdown}) \quad (7.2.9)$$

where I_{rx} is the current drawn in the RX state.

7.2.3 Energy Consumption of DLDC-MAC

DLDC-MAC protocol consumes energy when sending $E_{txbeacon}$ and receiving $E_{rxbeacon}$ beacons:

$$E_{LDC} = E_{txbeacon} + E_{rxbeacon} \quad (7.2.10)$$

Thus, to estimate the energy consumption of DLDC-MAC, the total number of beacons a day B is estimated:

$$B = \frac{T_{day}}{T_{beacon}} \quad (7.2.11)$$

where T_{beacon} is the beacon period, i.e., 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 $t_{rxbeaconafter}$ time in the RX state to receive potential network control frames, like network join requests. Clearly, as nodes switch from the TX to the RX state, they consume extra $E_{txrxswitch}$ energy. The total energy consumed for sending beacons equals:

$$E_{txbeacon} = B \cdot (t_{txbeacon} \cdot I_{tx} + t_{rxbeaconafter} \cdot I_{rx} + E_{txrxswitch} + E_{startup} + E_{shutdown}) \quad (7.2.12)$$

Both parameters beacon length λ_{beacon} and the listening time after beacons λ_{beacon_after} are expressed in bytes. The following formulas express them in time units, according

to the transceiver data rate ϑ:

$$t_{txbeacon} = \frac{\lambda_{beacon}}{\vartheta} \tag{7.2.13}$$

$$t_{rxbeaconafter} = \frac{\lambda_{beacon\_after}}{\vartheta} \tag{7.2.14}$$

Beacon Reception

Nodes with DLDC-MAC receive beacons from all neighbors and consume energy $E_{rxbeacon}$:

$$E_{rxbeacon} = B \cdot nbours \cdot (t_{rxbeacon} \cdot I_{rx} + E_{startup} + E_{shutdown})$$

where $nbours$ is the number of neighbors, and $t_{rxbeacon}$ is the time needed to receive a single beacon. As nodes compensate clock drift, they apply guard times t_{guard} to each beacon. Therefore, the time to receive a single beacon equals:

$$t_{rxbeacon} = t_{guard} + t_{txbeacon} \tag{7.2.15}$$

For the sake of simplicity, $t_{txbeacon}$ includes the preamble and the SFD field as well.

Obviously, guard times depend on the beacon period T_{beacon}, i.e., 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:

$$t_{guard} = \frac{\delta \cdot T_{beacon}}{1 - MBR} \tag{7.2.16}$$

where δ is relative drift among neighbors, and MBR is the average missed beacon rate. In this scenario, nodes use the MADC (Moving Average Drift Compensation) solution for guard times (details in Chapter 6).

7.2.4 Preamble Sampling

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

$$E_{preamble} = E_{tx} + E_{rx}$$

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}:

$$N_{tx} = \frac{T_{day}}{T_{event}}$$

where T_{day} is "the amount of time units a day", like in the 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, TX energy is estimated as:

$$E_{tx} = N_{tx} \cdot (T_{sleep} + t_{frame}) \cdot I_{tx} + E_{startup} + E_{shutdown}$$

where T_{sleep} is the sleep time of receivers and also the preamble length.

With the Preamble Sampling approach, nodes repeatedly check the channel activity. The number of such check operations a day N_{rx} is:

$$N_{rx} = \frac{T_{day}}{T_{sleep}}$$

On average, nodes receive a half of the preamble before receiving data frames, and the total reception energy equals:

$$E_{rx} = [N_{rx} \cdot t_{rx} + N_{tx} \cdot (\frac{T_{sleep}}{2} + t_{frame}] \cdot I_{rx}$$

where t_{rx} is the sampling channel duration, i.e., the time nodes need to discover whether other nodes send a long preamble. 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 [37] needs approx. 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 in TICER. In this case, t_{rx} is a few times longer than in B-MAC. For instance, if wake-up frames are 30 bytes 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 Chapter 5 revealed that nodes need a few ms to read data from the RX buffer.

7.2.5 Microcontroller Energy Consumption

Although microcontrollers (µCs) exhibit several power consumption states, this evaluation considers only two: active $E_{mcuactive}$ (executing code, reading sensors, sending, receiving, etc.) and sleep E_{sleep} (only a low rate clock is running). Thus, the

microcontroller consumes E_{mcu} energy a day:

$$E_{mcu} = E_{mcuactive} + E_{sleep} \tag{7.2.17}$$

$$E_{mcuactive} = T_{mcuactive} \cdot I_{mcuactive} \tag{7.2.18}$$

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

$$E_{sleep} = T_{sleep} \cdot I_{sleep} \tag{7.2.19}$$

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, i.e., they sleep to save energy. Clearly, T_{sleep} is calculated as a complement of other activities, i.e., sending and receiving data, listening for potential transmissions, code execution, etc.

7.3 Results

7.3.1 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 send 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.

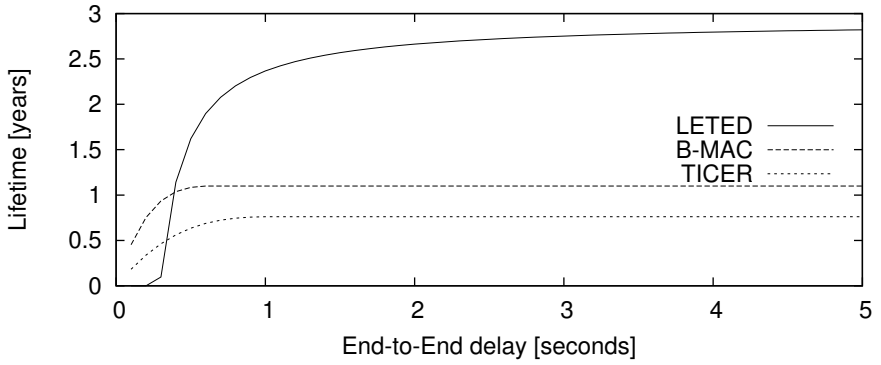
Section 7.1.2 introduced the tradeoff in the preamble length of B-MAC and TICER. In short, there is the optimal preamble length that offers the longest lifetime. Table 7.4 illustrates the best preambles of B-MAC and TICER, which are applied in all scenarios. By doing so, this evaluation presents the best cases of B-MAC and TICER.

Comparison

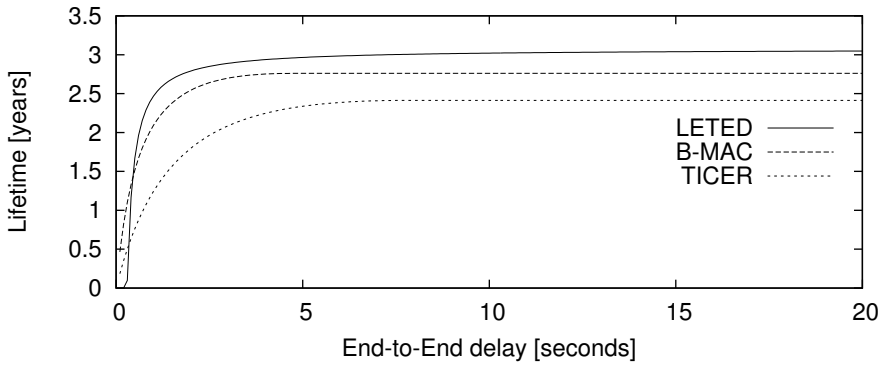
In the S-1 scenario, LETED outperforms Preamble Sampling and achieves significantly longer lifetimes. For example, nodes with LETED work 2x or 3x longer than with B-MAC or TICER (see Figure 7.3.1a). Such a huge difference originates from the energy consumed for transmissions. LETED needs only 0.033 mAh a day for transmissions and 0.078 mAh to receive (see Figure 7.3.2). Nodes with B-MAC, however, consume an order of magnitude more energy, i.e., 0.992 mAh and 0.564 mAh for sending and receiving. 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, resulting in 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 of B-MAC in S-2, i.e., 2.96 and 2.76 years respectively for end-to-end delays of 5 seconds (see Figure 7.3.1b). In this case, B-MAC uses preambles of 940 ms and consumes slightly more energy for communication than LETED (see Figure 7.3.2). 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 TICER work shorter than B-MAC, as they need a longer time to check the channel activity (see Figure 7.3.1b).

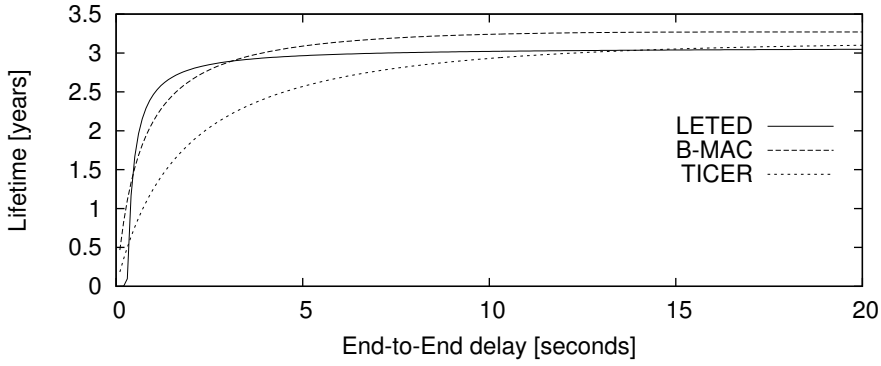
In the S-3 scenario, B-MAC achieves the best result in lifetime, among three examined approaches, for delays longer than 2.4 seconds (see Figure 7.3.1c). In this case, it consumes only 0.056 mAh a day in total for TX and RX (see Figure 7.3.2). Although LETED consumes even less energy for transmissions than B-MAC (see Figure 7.3.2), it needs extra energy for the underlying DLDC-MAC protocol. Therefore, nodes with LETED work slightly shorter than with B-MAC. However, there are only minor differences in lifetime between these two solutions. For example, nodes with LETED operate 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 because of longer periods needed to check the channel activity.



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



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

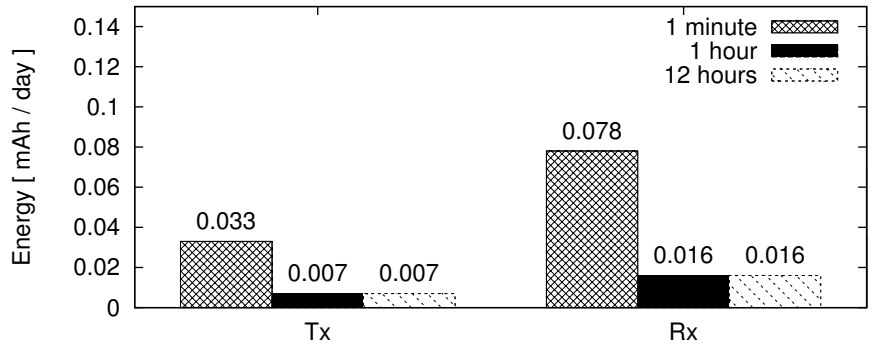


(c) S-3 Scenario: event notices sent every 12 hours

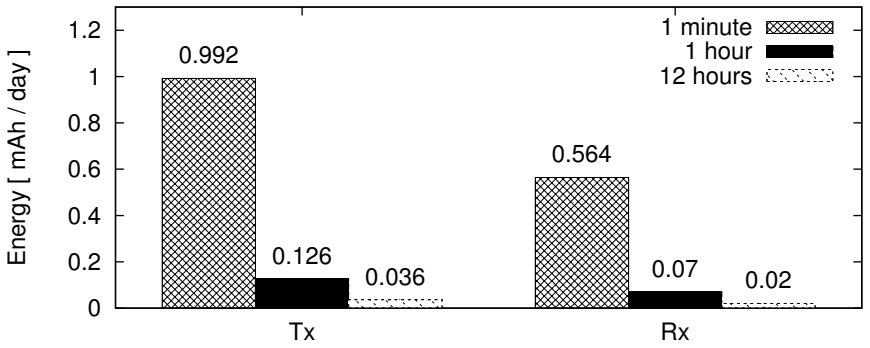
Figure 7.3.1: The lifetime of nodes with LETED and with Preamble Sampling (B-MAC and TICER).

Preamble Sampling

The lifetime of nodes with B-MAC and TICER depends mainly on the data rate. For example, nodes with B-MAC show a lifetime of 2.76 years when supporting 5-second delays and sending data rarely, i.e., once an hour (see Figure 7.3.1b). Should nodes



(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 7.3.2: 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

send frames once a minute, the lifetime is reduced to 1.1 years (see Figure 7.3.1a). 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 7.3.2). 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, energy consumption is increased at the receiver too, when transmitting high data rates (see Figure 7.3.2).

LETED

There are only minor differences in the lifetime of nodes based on LETED supporting various data rates. For example, with an event frequency of 1 minute, i.e., data rate is one frame a minute, nodes are operational 2.82 years and support 5-second delays (see Figure 7.3.1a). If events occur once an hour, the lifetime is longer by 4% only. Such minor differences for 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 7.3.2). Thus, energy consumption cannot be reduced significantly by transmitting at lower rates.

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 multi-hop drift (details in Chapter 5). Therefore, although events rarely occur, that is, every 1 and 12 hours respectively, nodes send SYNC frames every 5 minutes. As they transmit the same number of frames in both cases, the results do not differ.

Summary

The above observations show that LETED fits better than Preamble Sampling (PS) for scenarios with moderate data rates, i.e., about one frame a minute. In these cases, nodes with LETED can achieve significantly longer lifetimes. With lower duty cycles, PS achieves similar results as LETED.

The main advantage of PS over LETED is the small code size. For example, B-MAC needs 4 kB of ROM [37], 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 several sources detect the same event in normal cases, multiple notices are forwarded 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 reduces the collision risk. However, it causes extra end-to-end delays.

2. Receivers wake up periodically to check shortly for the channel activity. The best scenario was considered, i.e., nodes reliably detected channel activities. However, receivers can wrongly sense the channel. That is, they either miss an activity, or detect an idle channel as active (false positive):

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

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.

LETED should perform better in dense networks or/and with higher data rates. Owing to the TDMA approach, 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 development sensor nodes have more memory, they do not suffer from large LETED size.

Unfortunately, LETED poses the risk that nodes lose the synchronization and do not wake up at the same time. For example, should crystal oscillators start running imprecisely, nodes with LETED cannot handle it, and the protocol does not work. Although nodes did not encounter such oscillator problems with drift experiments discussed in Chapter 6, such risks cannot be excluded.

7.3.2 LETED and Schedule-Based MAC

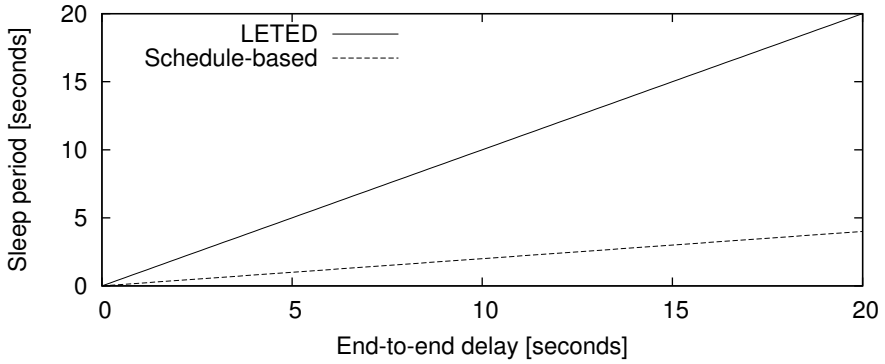


Figure 7.3.3: Sleep period of LETED and schedule-based MAC for various end-to-end delays in 5-hop networks: the longer the sleep period, the better

This section compares LETED with schedule-based MAC (S-B) protocols, that is, they use a wake-up schedule, and are represented by DLDC-MAC in this case. Since

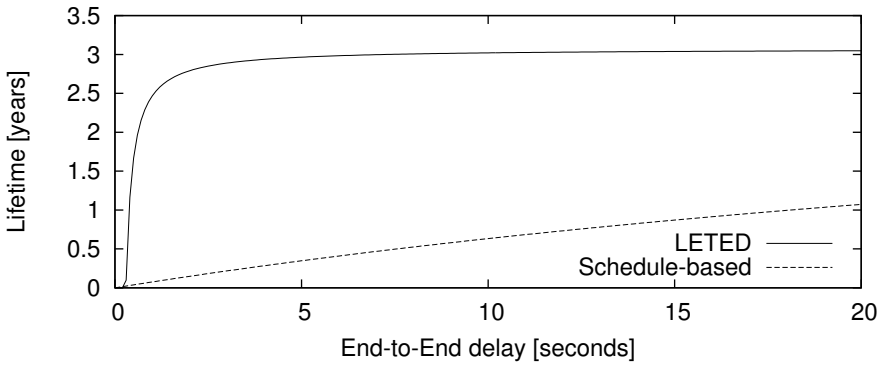


Figure 7.3.4: Lifetime of nodes with LETED and with schedule-based MAC protocols

schedules of S-B approaches are not aligned along the path, nodes wake up often to support short delays. As introduced in Chapter 3, the sleep period equals to the delay time divided by the number of hops to the sink. Figure 7.3.3 presents sleep periods of LETED and S-B. Owing to the staggered schedule, nodes with LETED sleep long, nearly 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 bring down the sleep period to a second. Thus, nodes with S-B wake up often, consequently consume more energy, shortening the lifetime.

As expected, LETED outperforms S-B protocols in scenarios with short end-to-end delays. Figure 7.3.4 shows that LETED achieves 8x longer lifetimes than S-B for delays of 5 seconds and shorter. For example, nodes with LETED can work almost 3 years and support 5-second delays. In this case, the S-B lifetime is only 0.35 years. For longer delays, i.e., from 5 to 10 seconds, LETED achieves lifetimes 6x longer than S-B.

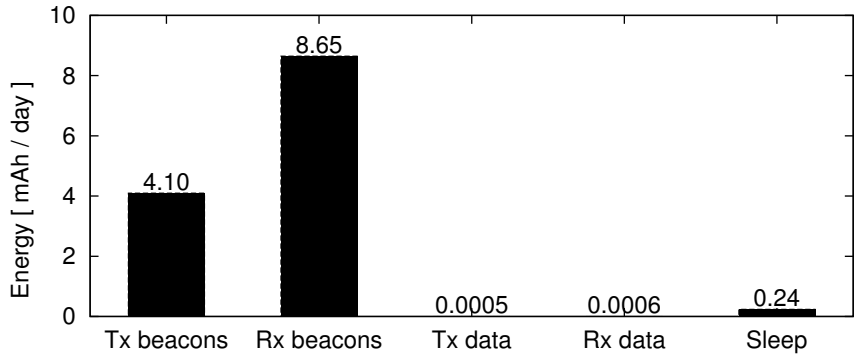


Figure 7.3.5: DLDC-MAC energy consumption; short sleep periods (1 second here) and a long time of beacons transmission and reception (about 12 ms altogether) result in an excessive energy waste

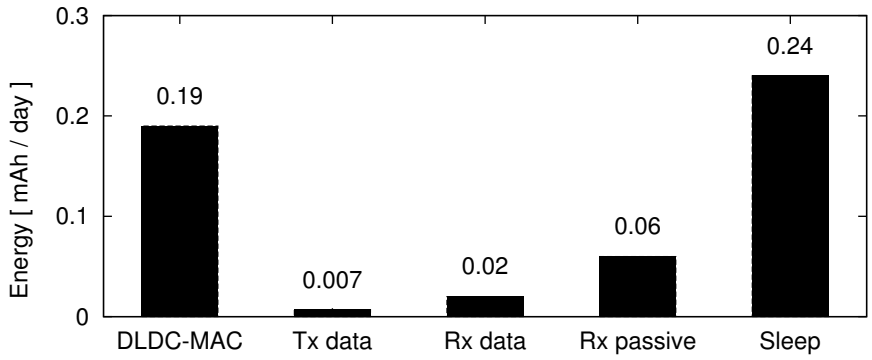


Figure 7.3.6: Energy consumed by LETED; owing to Idle Listening Avoidance, nodes consume a small amount of energy while waiting for potential transmissions (Rx passive)

Figure 7.3.5 presents energy consumption of DLDC-MAC. As above stated, nodes wake up often to send or receive beacons. A huge amount of energy is therefore consumed only for preparing to data transmissions. That is, nodes need about 13 mAh for beacons altogether (see Figure 7.3.5), which is larger by four orders of magnitude from the energy consumed on data transmission. Other S-B protocols may achieve even worse results. For example, Dozer [9] applies beacons too, but uses longer guard times than DLDC-MAC. Therefore, it needs more energy for beacons. Besides, S-MAC [53] requires longer times in the active state than DLDC-MAC, as it uses extra RTS and CTS frames apart from guard times. Thus, DLDC-MAC offers a similar performance to the best case of S-B protocols.

LETED keeps nodes ready for transmissions in passive slots, i.e., nodes listen periodically for incoming data. The energy of keeping nodes ready is depicted in Figure 7.3.6 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 Chapter 5) and the drift prediction approach (see Chapter 6), nodes wake up for about 0.5 ms only in passive slots. Therefore, LETED consumes significantly less energy than S-B protocols.

Nodes with LETED apply 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 standalone protocol (13 mAh / day) that supports short delays. In the first case, nodes do not need to send beacons often, since LETED takes care of fast transmissions along the path. They use beacons to send control frames only, e.g., to set up a new schedule. Therefore, they choose a beacon period to be 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 (see Figures 7.3.5 and 7.3.6).

7.3.3 Staggered Schedule

LETED applies new solutions, which reduce idle listening, to the staggered schedule, introduced in DMAC [29]. First, it minimizes passive slots and therefore powers down the transceiver early, referred to as ILA (see Chapter 5). Second, LETED shortens guard times by applying the MADC drift prediction approach (details in Chapter 6). Since DMAC does not suggest a way to detect and to shorten passive slots, this work assumes it uses the generic software solution (see Chapter 5). Besides, DMAC adopts an external time synchronization protocol to cope with clock drift. In this evaluation, however, DMAC applies the same solution to the drift problem as LETED.

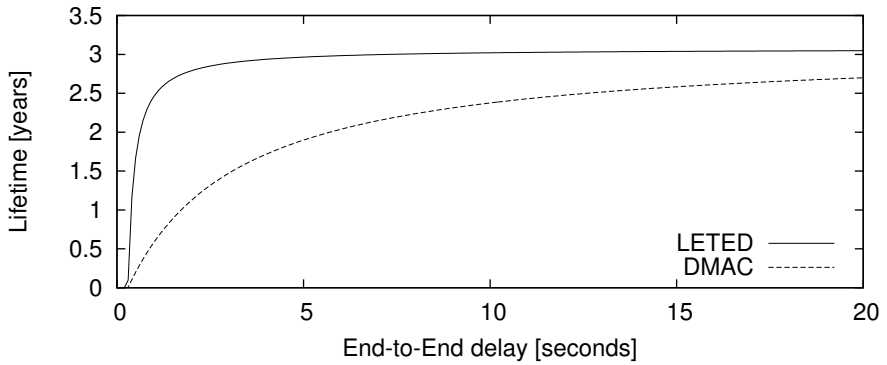


Figure 7.3.7: Lifetime of LETED and DMAC; owing to Idle Listening Avoidance (ILA) LETED shortens significantly passive slots, saves the energy and prolongs the lifetime

Figure 7.3.7 presents the lifetime of nodes working with LETED and DMAC. Owing to the energy-saving solutions, LETED can prolong 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 7.3.8). 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 7.3.9). 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 about 150 µs to receive

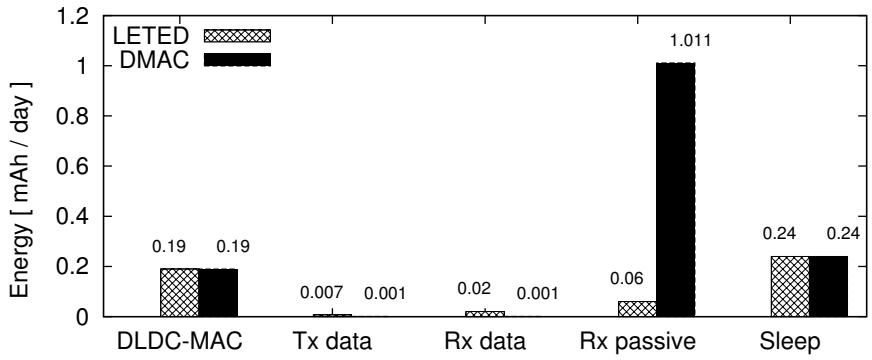


Figure 7.3.8: Energy consumed by LETED and DMAC, which represents MAC protocols with staggered schedule;

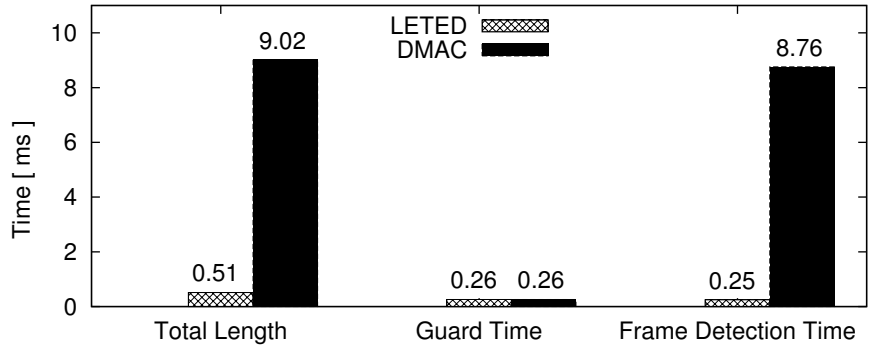


Figure 7.3.9: Passive slots consist of guard times for compensating clock drift and the time needed to detect that no frame arrives in the current slot;

a preamble, including the Start Frame Delimiter (SFD), and about 100 µs to handle the SFD interrupt. If the transceiver does not raise the SFD interrupt within this time, the node powers down the radio. DMAC, however, uses a generic software approach. That is, it usually needs long times to get frames from the RX buffer, almost 9 ms in this case. Clearly, such a huge difference in t_{detect}, that is, 0.25 ms vs. 8.76 ms, is the main reason for the improved performance of LETED.

In this consideration, LETED and DMAC applied the same method to compensate drift and thus do not differ in the length of guard times (see Figure 7.3.9). However, DMAC should use an external time synchronization protocol and not the energy-efficient solution to guard times based on the moving average. This can lead to a degraded performance of DMAC.

Table 7.5: 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_{tx_slots}, E_{rx_slots}	energy consumed a day for sending and receiving
E_{mcu}	daily energy consumption of µC
$E_{mcuactive}$	daily energy consumption of µC in active mode when radio is powered down
E_{sleep}	daily energy consumption in the sleep state
t_{rx_active}, $t_{rx\_passive}$	length of RX active/passive slot
$t_{txbeacon}$	transmission time of a single beacon
$t_{rxbeacon}$	average reception time of a single beacon
$t_{rxbeaconafter}$	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 Frame Delimiter field
T_{sleep}	total sleep time 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

8 Increasing Link-Layer Reliability in Sensor Networks: ARQ and CSMA/CA

This chapter examines the ARQ (Automatic Repeat reQuest) and CSMA/CA (Carrier Sense Multiple Access With Collision Avoidance) protocols in sensor networks with a low duty cycle (LDC). It presents empirical results and discusses potential gains of both solutions. Since ARQ and CSMA/CA cause extra energy consumption, this chapter evaluates their impact on the lifetime of sensor nodes.

8.1 Empirical Evaluation

8.1.1 Overview

To evaluate the performance of ARQ and CSMA/CA in sensor networks, indoor and outdoor experiments were carried out. The nodes were grouped into pairs: one node applied CSMA/CA with ARQ and another used ARQ only. In this way, nodes with and without CSMA/CA worked under similar conditions. It allowed a reasonable comparison.

Nodes with CSMA/CA applied the default TinyOS configuration, that is:

- They waited from 300 µs to 10 ms before starting to sense the channel.
- On idle channel, they started the transmission after a random time, up to 200 µs.
- If nodes detect a channel activity, they waited a random time, from 300 µs to 2.4 ms, and sensed the channel again.

To evaluate the impact of ARQ retries on the RX rate, nodes sent each frame 10 times. Since each frame included the current retry counter, the sink estimated the RX rate for various number of ARQ retries.

After powering up, nodes selected randomly the TX time of the first frame. Then, they sent frames periodically to the sink. Because of clock drift, however, TX times

of source nodes moved relatively to each other and posed an overlap risk. Since overlap cases affect the results, nodes added a random time (up to 250 ms) to each TX time and minimized such risks.

Indoor Experiments

Table 8.1: Three indoor experiments with various transmit frequencies f and additional traffic generators (TG); fpm (frames per minute):

Acronym	Data traffic parameters	Channel duty cycle
E1	$f = 1\,fpm$, no TG	0.01%
E2	$f = 2\,fpm$; 1x TG with $f = 300\,fpm$	0.52%
E3	$f = 7.5\,fpm$; 2x TG with $f = 1200\,fpm$ each	4.08%

These experiments evaluated three pairs of Tmote Sky nodes placed in an office environment. Each experiment lasted a week. The nodes periodically sent frames to the sink with a different frequency (see Table 8.1). In experiments E2 and E3, some nodes served as traffic generators, i.e., they sent extra frames to increase the duty cycle.

The experiment E1 resembles common scenarios with a low duty cycle. The E2 and E3 experiments consider higher data rates, for example, dense networks or frequent transmissions. Clearly, in the latter two scenarios, there is a higher collision risk, and nodes may benefit from CSMA/CA.

Since the office is only 15 meters long, the Signal-to-Noise Ratio (SNR) of the default TX power is high. In this case, nodes do not suffer from unreliable wireless communication and receive almost all frames. Thus, to consider common scenarios with a lower SNR, e.g., because of larger distances between sensor nodes, the TX output power was reduced to -25 dBm.

Nodes were placed in an occupied office, that is, employees were present in working hours and used wireless devices. Tmote Sky nodes use transceivers based on IEEE 802.15.4 standard on the frequency band of 2.4 GHz. Since there were wireless LAN devices working on the same frequency in the office, they affected communication between nodes.

Outdoor Experiment

Ten pairs of Tmote Sky nodes, with and without CSMA/CA, were divided into 5 groups, placed outdoors, and ran 2 months. The average distance from nodes to the sink was about 40 meters. To cover such a distance, nodes needed the full TX power (0 dBm). Only the sink worked indoors, as it delivered experimental data to a logging computer with USB connection. In such a scenario, however, indoor wireless devices could affect the sensor network. To evaluate an outdoor environment, i.e., without other wireless devices, the antenna of the sink was placed outdoors, behind the window, and connected to the sink with a cable. As the window had metal blinds pulled down during the experiment, it prevented that wireless indoor devices affected significantly the communication from nodes to the sink.

In this scenario, nodes sent frames periodically to the sink every 8 seconds. As each frame was about 4 ms long, it resulted in the channel duty cycle of about 1%.

Two out of twenty sensor nodes stopped working at the beginning, probably because of a loose contact to batteries. Nonetheless, the experiment was carried out with the remaining 18 nodes.

8.1.2 Results

CSMA/CA

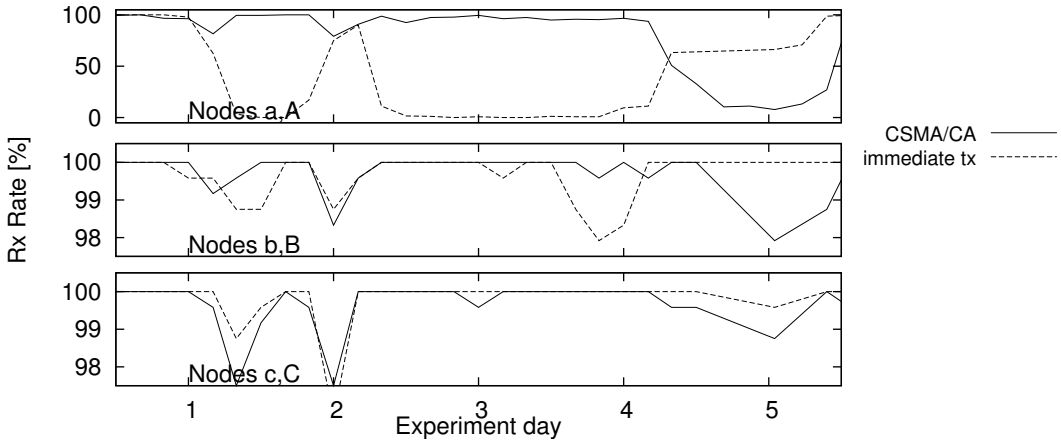


Figure 8.1.1: Indoor E1 Experiment (4.08% duty cycle): Reception rate without ARQ, either with CSMA/CA or without it

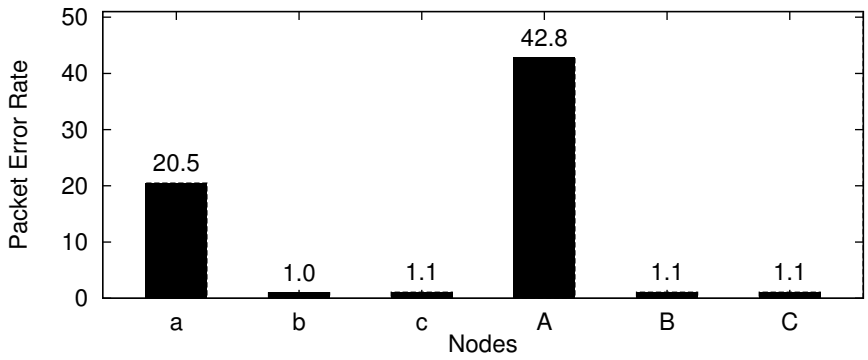
Because of a low traffic load, CSMA/CA did not improve significantly the RX rate in common LDC scenarios, i.e., in the E1 experiment. The sink rarely missed frames, apart from the node group A (see Figure 8.1.1). Therefore, the average PER

was almost the same for nodes with CSMA/CA and w/o it (see Figure 8.1.2a). For example, nodes with CSMA/CA of the group B had a PER lower by 0.1% than nodes without it. Although such a small number of nodes does not provide accurate results, it shows a general tendency of CSMA/CA: it hardly improves the RX rate in applications with a low duty cycle.

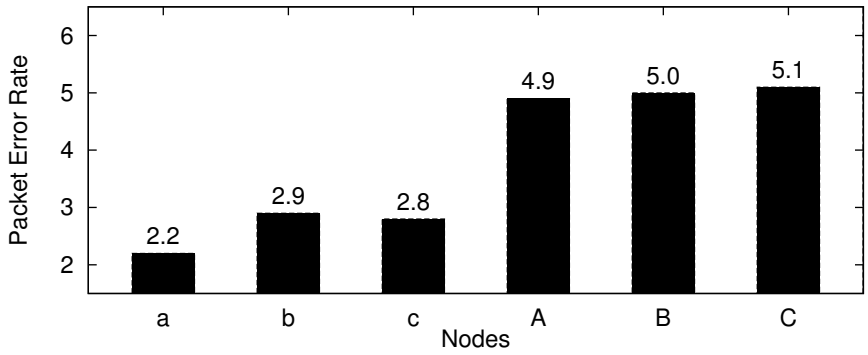
During the E1 experiment, the node group A suffered from communication problems. For example, during the third day, the sink missed almost all frames from the node without CSMA/CA (see the top plot in Figure 8.1.1). Later on, the node with CSMA/CA suffered from similar problems, and the sink missed about 90% frames. Therefore, the average PER of these nodes, 20.5% and 42.8% respectively (see Figure 8.1.2a), was much higher than the PER of other nodes. Since the node group A was far away from the sink, the SNR at the sink was low, and it caused such poor results. The sink received more frames from the node with CSMA/CA than from the node without it. However, the former node did not achieve better results owing to CSMA/CA, as the difference in the RX rate is too large. For example, during the third day, the sink received all frames from the node with CSMA/CA, but missed everything from the node w/o it (see Figure 8.1.1 top). Theoretically, CSMA/CA could recover from collisions and provide such good results. In this case, however, the collision risk was low, since nodes rarely send frames. Moreover, as other nodes without CSMA/CA achieved good results (see middle and bottom plots in Figure 8.1.1), other wireless devices did not cause collisions as well. Besides, during the fifth day, the sink received more frames from the node without CSMA/CA than from the node with collision avoidance. Therefore, such poor results of the node group A stem probably from a low SNR and not because of a high collision risk.

In the E2 experiment, CSMA/CA improved slightly the RX rate. The sink received 96% to 98% frames from nodes with CSMA/CA and 92%-96% from nodes without it (see Figure 8.1.3). Therefore, the average PER of nodes with CSMA/CA was higher by 2-3% from the PER of nodes without it (see Figure 8.1.2b). Clearly, in this experiment, collisions occurred more often than in E1, and therefore CSMA/CA improved the RX rate. However, if nodes already applied ARQ protocol, the extra CSMA/CA did not improve the RX rate significantly. For example, nodes using ARQ with 1-2 retries achieved an RX rate of almost 100% (details introduced in the next paragraph). Clearly, CSMA/CA cannot improve the results significantly in this case.

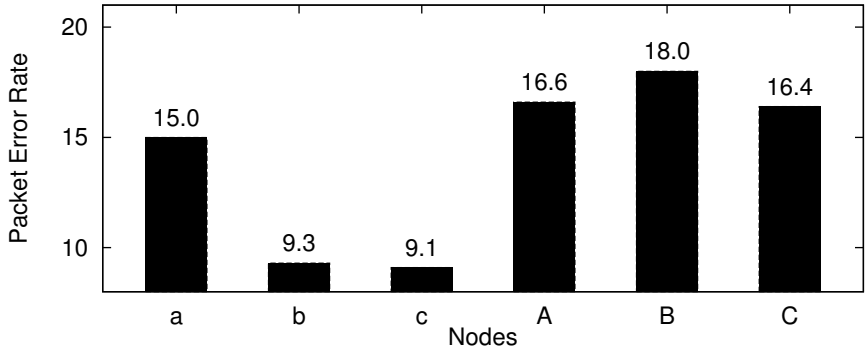
Because of a higher collision risk, the sink missed more frames in the experiment E3 than in E1 and E2 (see Figures 8.1.1, 8.1.3 and 8.1.4). Thus, the average PER in E3 was higher than in previous runs, and nodes with CSMA/CA achieved better



(a) E1 indoor experiment



(b) E2 indoor experiment



(c) E3 indoor experiment

Figure 8.1.2: Measured packet error rate among all nodes in 3 indoor experiments; nodes a/b/c used CSMA/CA; nodes A/B/C worked without CSMA/CA

results. For example, the sink missed more than 16% frames from nodes without CSMA/CA in E3 and about 5% in E2 (see Figure 8.1.2). On average, CSMA/CA improved the reception rate by about 7%, apart from the node group A. In this case,

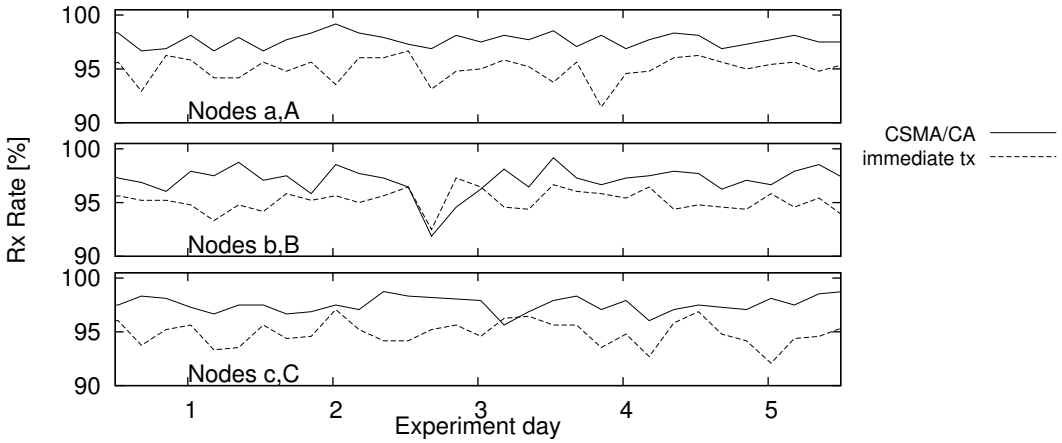


Figure 8.1.3: Indoor E2 Experiment (0.52% duty cycle): Reception rate without ARQ, either with CSMA/CA or without it

the difference between nodes with and without CSMA/CA was only 1.6%, probably because of a low SNR of the node with collision avoidance.

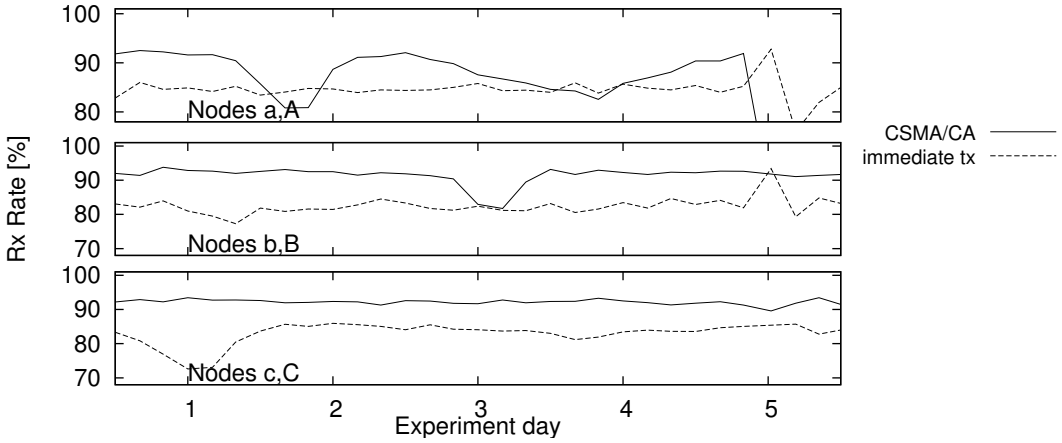


Figure 8.1.4: Indoor E3 Experiment (4.08% duty cycle): Reception rate without ARQ, either with CSMA/CA or without it

CSMA/CA improved the reception rate in the E3 experiment from 81.9% to 89.3%, when nodes did not apply the ARQ solution (see Figure 8.1.5). However, when nodes send retries on frame loss, the CSMA/CA approach improves only slightly the RX rate. For example, with 2 ARQ retries, the sink received 98.3% frames from nodes w/o CSMA/CA and 99% from nodes with collision avoidance (see Figure 8.1.5). With more ARQ retries, the sink received almost all frames, and CSMA/CA could not improve the performance.

In the outdoor experiment, nodes with CSMA/CA sometimes achieved slightly bet-

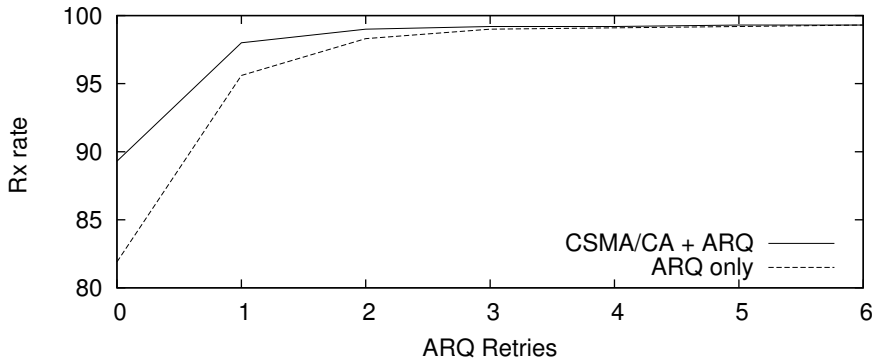


Figure 8.1.5: Indoor E3 experiment: Average reception rate with various ARQ retries number for nodes with and without CSMA/CA

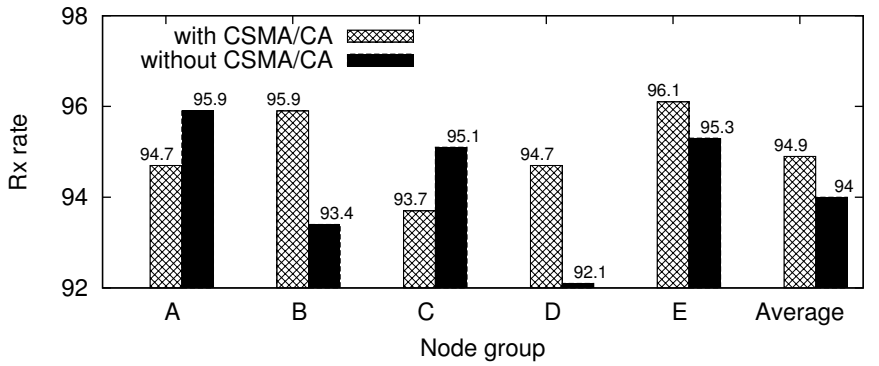
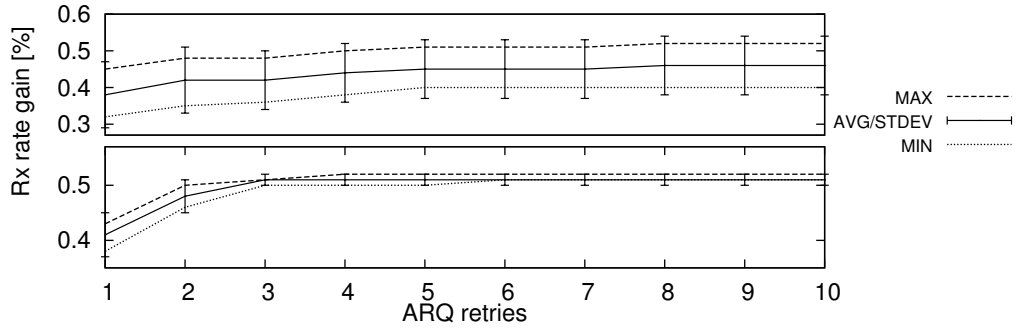


Figure 8.1.6: Outdoor experiment: Reception rates of nodes with and without CSMA/CA; the result for each node group separately and the average among all nodes

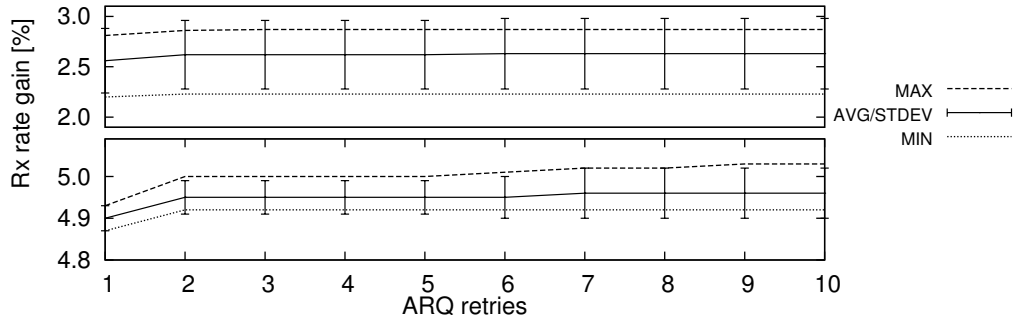
ter RX rates than without applying collision avoidance. For example, CSMA/CA improved the RX rate from 93.4% to 95.9% in the node group B (see Figure 8.1.6). However, in some cases, the sink received more frames from nodes without CSMA/CA. For example, nodes in the group C achieved 95.1% RX rate without CSMA/CA and 93.7% with the solution applied. Obviously, CSMA/CA did not decrease the reception rate, and such results stem from the estimation errors caused by a small number of nodes in each group. On average, CSMA/CA recovered from some collisions and slightly improved the RX rate, from 94% to 94.9% (see Figure 8.1.6). Clearly, these results are rather rough, as the evaluation considered a few nodes only. For instance, the sink received fewer frames from two nodes without CSMA/CA (nodes 17 and 20) than from others. If the results do not include both nodes, there is no difference in the reception rate between nodes with CSMA/CA and without it. Therefore, these observations show only that CSMA/CA might slightly improve the RX rate

in applications with a low duty cycle, but does not provide accurate estimations of CSMA/CA gain.

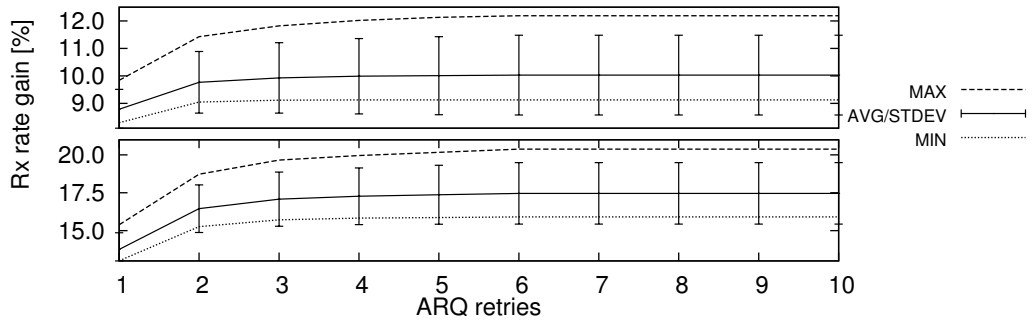
ARQ



(a) E1 indoor experiment; the results exclude the node group A, since it was placed too far from the sink



(b) E2 indoor experiment



(c) E3 indoor experiment

Figure 8.1.7: ARQ impact on the reception rate for various retry count; the plots shows average reception rate gain (and standard deviation, min/max values) for each indoor experiment; CSMA/CA with ARQ in top diagrams, ARQ w/o CSMA/CA in bottom plots

During the indoor E3 experiment, the sink received up to 92% frames without ARQ applied (see Figure 8.1.4). ARQ improved the RX rate to more than 95% with 1 retry and to 97% with 2 retries. Figures 8.1.7 show the average gain of ARQ in indoor experiments. For example, in the E3 experiment, ARQ improved the RX rate by 14% with just 1 retry for nodes w/o CSMA/CA and by 17% with 3 retries (see Figure 8.1.7c). Any further increase in the retry number did not improve the RX rate significantly, as the sink received almost all frames with 3 retries.

In the E2 experiment, ARQ with 1 retry improved the RX rate to 99%, i.e., by 5% for nodes w/o CSMA/CA (see Figure 8.1.7b). As the sink received almost all frames with 1 retry only, further increase in applying ARQ retries did not significantly improve the RX rate.

As nodes did not suffer from a high PER in E1 (see Figure 8.1.2a), apart from the node group A, the ARQ solution did not affect the RX rate significantly. On both node types, i.e., with and without CSMA/CA, ARQ increased the RX rate by 0.5% in the best case (see Figure 8.1.7a). As already mentioned, the sink missed many frames from the node group A, as the long distance and a low SNR caused a high packet error rate. In this case, ARQ did not improve the RX rate significantly, as the SNR of retries was also too low to receive them. It shows that ARQ should not be applied at all with a low SNR, as it does not improve the RX rate, but only wastes energy.

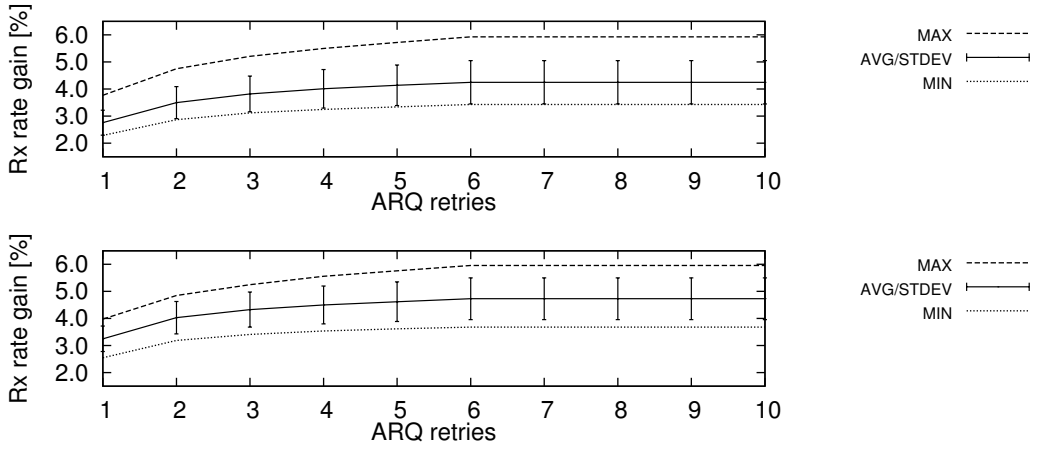


Figure 8.1.8: Average reception rate (together with standard deviation and min/max values) of all nodes evaluated outdoors, with collision avoidance (top) and w/o it (bottom), for various number of ARQ retries

In the outdoor experiment ARQ improved the RX rate only to a certain retry number as well (see Figure 8.1.8). With two retries nodes increased the RX rate by 3.5% (with CSMA/CA) and by 4% (ARQ only). Further increase in ARQ retries

improved the RX rate slightly, as the sink received 98% frames and more with 2 retries.

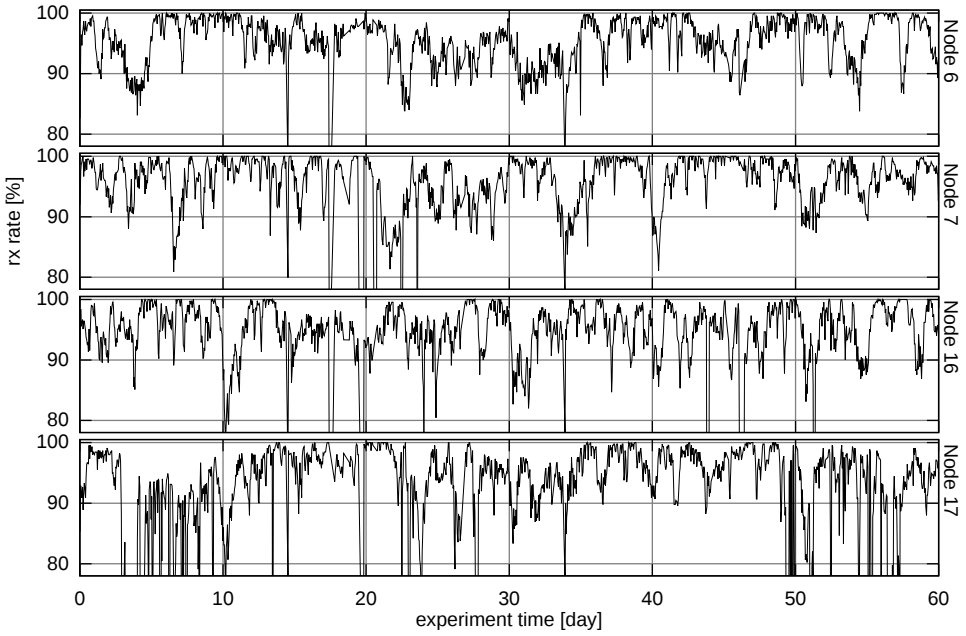


Figure 8.1.9: Reception rate of groups 2 w/o ARQ throughout the outdoor experiment; nodes 6,7 applied CSMA/CA approach; nodes 16 and 17 transmitted frames immediately without collision avoidance

Figures 8.1.9 and 8.1.10 present the reception rate of eight out of twenty nodes throughout the outdoor experiment. Even when the nodes did not apply the ARQ approach, the sink usually received most frames, i.e., the average reception rate was approx 94.6%. The evaluation shows that the reception rate varies over time but the RX rate dropped rarely below 90%. Nodes recover from unreliable communication in these short periods by applying the ARQ solution. As above said, they increase the RX rate to 98% and more with 2 retries only.

8.2 Lifetime Evaluation

Chapter 7 introduced the energy consumption model, which estimates the lifetime of sensor nodes. This section adds the ARQ and CSMA/CA solutions to the model and evaluates their impact on the lifetime. Table 8.2 presents the parameters of ARQ and of collision avoidance applied here. The evaluation considers the scenario presented in Chapter 7. That is, nodes send data to the sink and support 5-second end-to-end delays with LETED and DLDC-MAC. In this evaluation, they apply extra solutions to communication reliability: ARQ and CSMA/CA. However, both solutions are

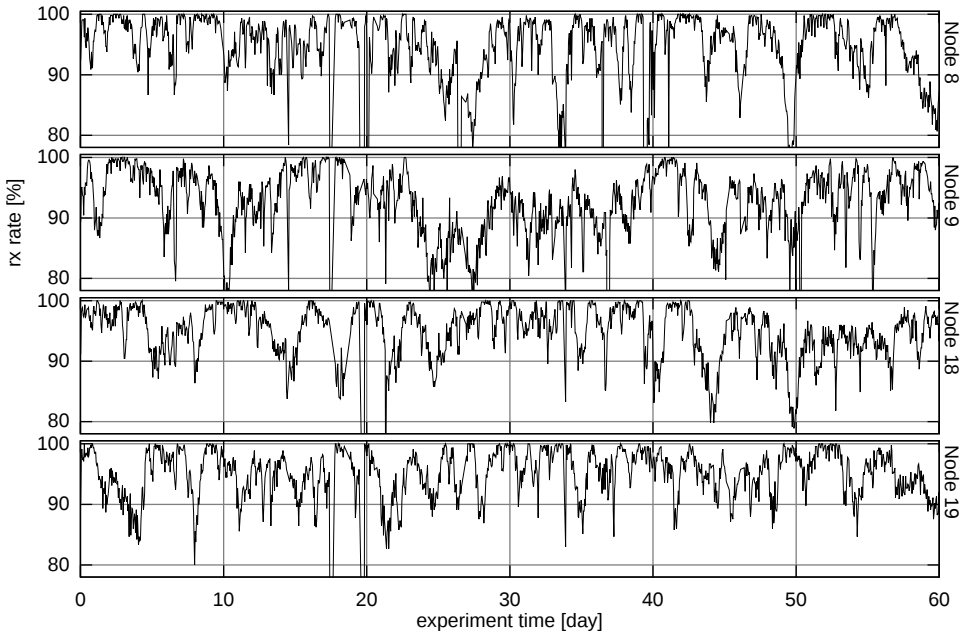


Figure 8.1.10: Reception rate of groups 3 w/o ARQ throughout the outdoor experiment; nodes 8,9 applied CSMA/CA approach; nodes 18 and 19 transmitted frames immediately without collision avoidance

used with LETED only, and not with DLDC-MAC, because of the following reasons:

- By applying CSMA/CA, node may postpone beacon transmissions to avoid collisions. However, as DLDC-MAC uses beacons to estimate clock drift, postponed beacons would influence the drift estimation.
- With ARQ, neighbors should send an acknowledgment (ACK) upon beacon reception. However, as there are usually several neighbors, they all send ACKs. Thus, they need to synchronize transmit times to avoid ACK collisions. Clearly, it results in an extra protocol overhead, but does not provide notable benefits.

8.2.1 Model Adaptation

TX Energy

The packet error rate (PER) depends on the frame length, i.e., the longer the frame is, the higher is the PER:

$$PER = 1 - (1 - BER)^{8*\lambda_{frame}}$$

where BER in the bit error rate, and λ_{frame} is the frame length expressed in bytes.

The number of retries $\eta_{retries\_tx}$ nodes send on average, i.e., for each frame, depends on the PER_{frame} (packet error rate of the frame). In addition, if nodes miss ACKs, they send frames again. Thus, the PER of acknowledgments PER_{ACK} affects $\eta_{retries\_tx}$ as well:

$$\eta_{retries\_tx} = \sum_{i=1}^{R_{max}} \left[PER_{frame} + (1 - PER_{frame}) \cdot PER_{ACK}\right]^n$$

where R_{max} is the highest retry count. Thus, the average number of frames η_{frames_tx} sent in a single active slot equals:

$$\eta_{frames\_tx} = 1 + \eta_{retries\_tx}$$

Since nodes may miss ACKs as well, the number of received ACKs η_{ACK_rx} in a single active slot equals:

$$\eta_{ACK\_rx} = \eta_{frames\_tx} \cdot (1 - PER_{frame}) \cdot (1 - PER_{ACK})$$

As above said, nodes expect an ACK for each sent frame. They do not receive ACKs in two cases. First, the frame was lost, and neighbors did not send ACK. Second, neighbors received the frame, but the ACK was lost. Thus, the number of expected but missed ACKs is estimated as:

$$\eta_{ACK\_missed} = (\eta_{frames\_tx}) \cdot \left[1 - (1 - PER_{frame})\,(1 - PER_{ACK})\right]$$

If nodes apply CSMA/CA, they check the channel activity for t_{CCA_listen} time before each transmission. On idle channel, they switch the transceiver from RX to TX state, and it consumes E_{rxtx_switch} energy.

When nodes do not receive an ACK, they wait for a few ms before sending frames again. Such a solution stems from the assumption the channel does not change significantly in a short time, i.e., a few milliseconds in this case. Thus, retries send immediately suffer from the same poor channel quality as the lost frames. To save energy, nodes keep the transceiver powered down between successive retries.

Nodes expect to receive ACKs with a length of t_{ACK} after the time specified offline before the deployment. Should nodes apply CSMA/CA, they wait for t_{CCA_avg} on each ACK, since neighbors may postpone ACK transmissions. However, nodes wait the worst-case back-off time t_{CCA_max}, if ACK is lost. The adapted equation to estimate TX energy, based on Eq. 7.2.4, is:

$$
\begin{aligned}
E_{txslots} \;=\; & N_{txslots} \cdot \{\eta_{frames\_tx} \cdot [E_{startup} + t_{CCA\_listen} \cdot I_{rx} + E_{rxtx\_switch} + \\
& t_{frame} \cdot I_{tx} + E_{shutdown} + E_{txrx\_switch}] + \\
& [(\eta_{ACK\_rx} \cdot t_{CCA\_avg} + \eta_{ACK\_missed} \cdot t_{CCA\_max} \\
& + \eta_{frames} \cdot t_{ACK}) \cdot I_{rx}]\}
\end{aligned}
$$

Rx Energy

In this evaluation, nodes miss frames with a certain packet error rate PER_{frame} and do not send ACKs. The average number of sent ACKs η_{ACK} in a single active slot equals:

$$\eta_{ACK} = \eta_{frame\_tx} \cdot (1 - PER_{frame}) \tag{8.2.1}$$

Nodes send ACKs almost immediately after frame reception. With CSMA/CA, they wait for only t_{CCA_avg} to counter the collision risk. The total energy consumed for ACK transmissions equals:

$$E_{tx\_ack} = N_{active} \cdot \eta_{ACK} \cdot \left(t_{CCA\_avg} \cdot I_{rx} + E_{rxtx\_switch} + t_{ACK} \cdot I_{tx}\right)$$

In general, nodes detect missing frame indirectly, i.e., they listen for incoming data, and after a timeout they assume the frame was lost. Thus, they listen the time needed to receive a frame. The average number of lost frames η_{lost} in a single active slot equals:

$$\eta_{lost} = \eta_{frame\_tx} \cdot PER_{frame}$$

Since nodes send ACKs for each frame received, the number of received packets $\eta_{received}$ equals the number of sent ACKs η_{ACK}, which was estimated previously (see Eq. 8.2.1):

$$\eta_{received} = \eta_{ACK}$$

In passive slots, nodes wait the maximal back-off time t_{CCA_max} before assuming the slot is passive. Thus, the passive slot length from Eq. 7.2.7 with applied CSMA/CA equals:

$$t_{rx\_passive} = t_{guard} + t_{CCA\_max} + t_{preamble} + t_{SFD}$$

There are two reasons for the idle channel in passive slots: neighbors did not send frames, or the frame was lost. However, nodes cannot determine it and wait the highest retries count R_{max} before giving up the reception try. The total energy consumed for reception in both active and passive slots equals:

$$\begin{aligned} E_{rx\_slots} &= I_{rx} \cdot [N_{active} \cdot \eta_{received} \cdot (t_{CCA\_avg} + t_{rx\_active}) + \\ &\quad N_{active} \cdot \eta_{lost} \cdot (t_{CCA\_max+} t_{rx\_active}) + \\ &\quad N_{passive} \cdot (1 + R_{max}) \cdot t_{rx\_passive}] + \\ &\quad [N_{active} \cdot (\eta_{received} + \eta_{lost}) + N_{passive} \cdot (1 + R_{max})] \\ &\quad \cdot (E_{startup} + E_{shutdown}) \end{aligned}$$

$$\begin{aligned} E_{rx\_slots} &= N_{active} \cdot (t_{rx\_active} \cdot I_{rx}) + N_{passive} \cdot (t_{rx\_passive} \cdot I_{rx}) + \\ &\quad (N_{active} + N_{passive}) \cdot (E_{startup} + E_{shutdown}) \end{aligned}$$

8.2.2 Results

Table 8.2: Model parameters used for the lifetime evaluation

Symbol	Description	Value
BER	Bit error rate (affects the average number of ARQ retries)	10^{-4}
R_{max}	ARQ maximal retries count	3
t_{CCA_listen}	Time to perform CCA before transmission	1 ms
t_{CCA_avg}	Average back-off time	5 ms
t_{CCA_max}	Worst-case back-off time	100 ms

The experiment results, previously introduced in this chapter, presented RX rates of single links. However, common sensor networks need a multi-hop communication. In this case, frames may be corrupted on each hop on the path to the sink. Hence, the probability P_{rx} the sink receive a frame, called the RX rate in this work, depends on the PER of each link:

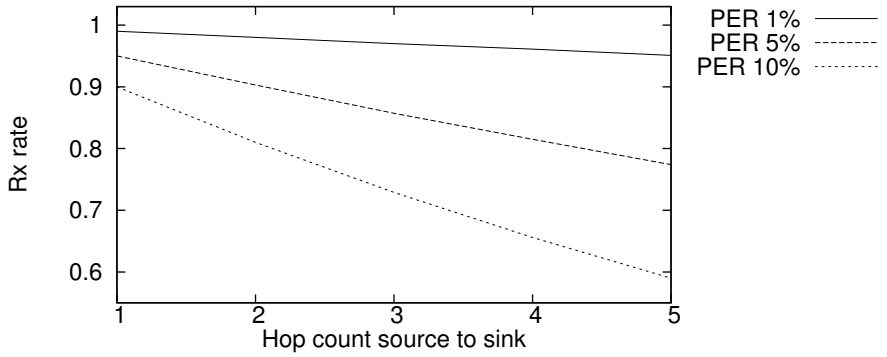


Figure 8.2.1: Packet reception rate (RX rate) in a multi-hop network for various average link packet error rate (PER)

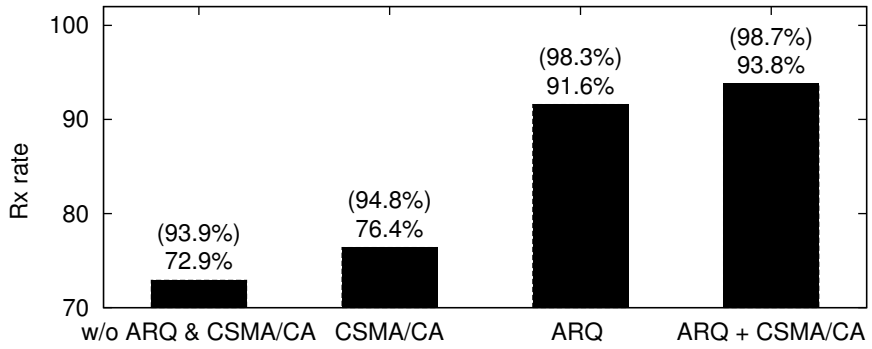
$$P_{rx} = \prod_{i=o}^{n}(1 - PER_i)$$

where n is the number of hops to the sink, and PER_i is the packet error rate of i-th link (hop) on the path to the sink. For the sake of simplicity, P_{rx} is estimated from the average packet error rate PER_{avg} of a single link:

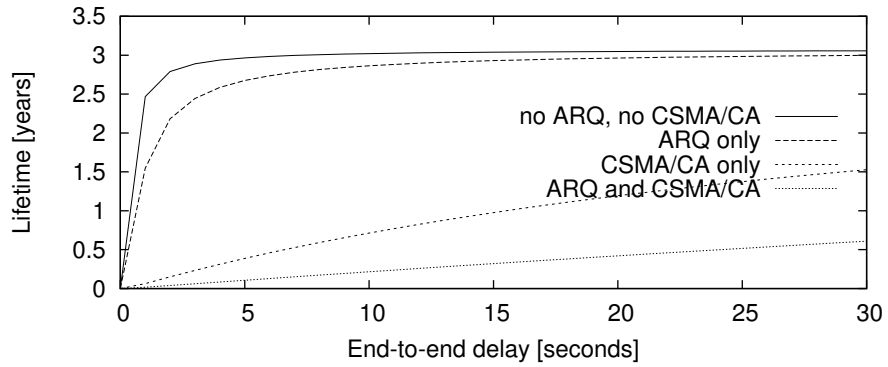
$$P_{rx} = (1 - PER_{avg})^n \tag{8.2.2}$$

Figure 8.2.1 depicts the RX rate in multi-hop networks for various PER of a single link, according to Eq. 8.2.2. As stated above, the average PER of the outdoor experiment w/o any reliable mechanisms was approx. 5%. In other words, the sink misses 5% frames in single-hop communication. However, in 5-hop networks the sink misses more than 20% frames with such a PER (see Figure 8.2.1). Besides, during some periods in the outdoor experiment, the nodes missed 10% frames and more in a single-hop network. Should all links of 5-hop paths have a similar PER, the sink receives less than 60% frames. The above examples demonstrate the need of solutions to reliable communication, like ARQ or CSMA/CA, in multi-hop networks.

Figure 8.2.2a shows the potential performance of ARQ and CSMA/CA in 5-hop networks, discussed in the following paragraphs. It was estimated by applying empirical single-hop PER values to Eq. 8.2.2. The lifetime results of ARQ and CSMA/CA, presented in Figure 8.2.2b, were estimated with the model introduced previously.



(a) Estimated reception rate at the sink in a 5-hop network for various solutions to reliable communication (ARQ applies up to 3 retries); the values in brackets are empirical RX rates of a single hop



(b) Impact of ARQ and CSMA/CA on the lifetime of nodes with LETED

Figure 8.2.2: Performance of solutions to reliable communication - ARQ and CSMA/CA - in a multi-hop network together with their impact on the lifetime

ARQ Performance

As expected, since ARQ increases the communication overhead, it shortens the lifetime of sensor nodes. Therefore, nodes w/o reliability solutions achieve the best lifetime, for instance, almost 3 years for 5-second end-to-end delays (see Figure 8.2.2b). However, they suffer from high packet loss rates in multi-hop networks. For example, with the average single-hop RX rate of 93.9%, the sink receives only 72.9% frames in 5-hop networks (see Figure 8.2.2a).

If nodes apply the ARQ protocol, they improve the RX rate by approx. 20% in the same scenario. By doing so, they shorten the lifetime by 10% only, that is, from 2.96 years to 2.67. However, such good lifetime results do not stem from the ARQ protocol itself, but from the way LETED deals with passive slots. ARQ with 3 retries increases a few times the energy consumed in passive slots, as nodes try to

receive each potential retry (see Chapter 3). However, owing to the Idle Listening Avoidance ILA (see Chapter 5), LETED consumes only a tiny amount of energy in passive slots. Therefore, although ARQ increases 4x the energy consumption of passive slots, it is still relatively small (see Figure 8.2.5). For instance, the energy consumed in passive slots with ARQ (0.25 mAh a day) is almost the same as the sleep energy, about 0.24 mAh a day.

Apart from passive slots, ARQ also affects energy consumed for sending and receiving data, as it involves transmissions of retries and acknowledgments. However, with BER of 10^{-4}, which results in approx. 10% PER, the average number of retries per frame is small. Thus, it only slightly affects the energy consumed for sending or receiving data (see Figures 8.2.3 and 8.2.4). That is, nodes with ARQ need extra 0.002 mAh energy a day for ACK transmissions,and 0.003 mAh to receive ACKs.

The example above reveals that nodes with LETED and ILA solutions benefit from ARQ protocol in low duty cycle networks. It improves the RX rate significantly, but impacts only slightly the lifetime.

CSMA/CA Results

Nodes should not apply CSMA/CA with LETED, as the latter reduces the collision risk owing to the transmission schedule (TDMA). In addition, CSMA/CA shortens the lifetime significantly, but does not improve the communication quality with low duty cycles. According to the experiments, CSMA/CA improved the single-hop RX rate by 1% on average in low duty cycle scenarios. Thus, nodes improve the RX rate about 5% in 5-hop networks, that is, from 72.9% to 76.4% (see Figure 8.2.2a). In this case, CSMA/CA shortens the lifetime 8x, i.e., from 3 years to 4.5 months.

Such poor lifetime results of CSMA/CA stem mainly from excessive energy consumption of passive slots. Nodes listen the worst-case back-off time in a single passive slot. Thus, although a single frame reception needs a few ms, nodes wait 100 ms in this scenario. By doing so, they consume almost 200x more energy in passive slots than LETED w/o CSMA/CA, that is, 11.1 mAh a day vs. 0.06 mAh (see Figure 8.2.5).

Since nodes with CSMA/CA check the channel activity before sending data, the approach affects the TX energy too. In this scenario, nodes check the channel for 1 ms before transmissions. In this case, nodes with CSMA/CA consume 0.001 mAh more energy for transmissions in a day (see Figure 8.2.3).

Nodes need about 9 ms to receive a single frame without CSMA/CA. When they apply CSMA/CA, nodes increase the RX time 1.5x (to 14 ms) to compensate transmissions postponed by CSMA/CA. It results in a similar energy penalty. Nodes with

CSMA/CA need about 1.5x more RX energy, that is, 0.024 mAh a day instead of 0.016 mAh (see Figure 8.2.4).

The results confirm the assumption that CSMA/CA should not be used with low duty-cycled applications. It does not significantly improve the RX rate, but mainly shortens the lifetime.

CSMA/CA with ARQ

Nodes with ARQ and CSMA/CA achieved the best performance during the experiments presented previously. On average, they improved the single-hop RX rate from 93.9% to 98.7% (see Figure 8.2.2a). However, it results in a significant lifetime penalty, as nodes work almost 30x shorter: 1.5 months instead of 3 years (see Figure 8.2.2b). Clearly, such disastrous results stem mainly from the poor CSMA/CA performance. In this case, nodes apply CSMA/CA to each retry and increase idle listening even more than CSMA/CA without ARQ. Thus, it results in excessive energy consumption. For example, nodes with CSMA/CA and ARQ consume 700x energy more in passive slots than nodes without any solution to reliability (see Figure 8.2.5).

Obviously, nodes must not use ARQ with CSMA/CA in low duty-cycled networks, although it achieves the best result in the RX rate. Since it shortens the lifetime significantly, nodes should apply other solutions to deal with unreliable links. For example, if nodes use ARQ only, they increase the performance significantly and only slightly affect the lifetime.

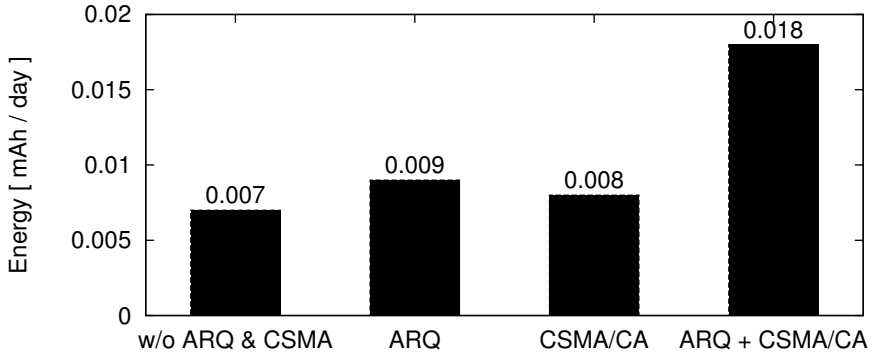


Figure 8.2.3: Energy consumed for sending data with LETED

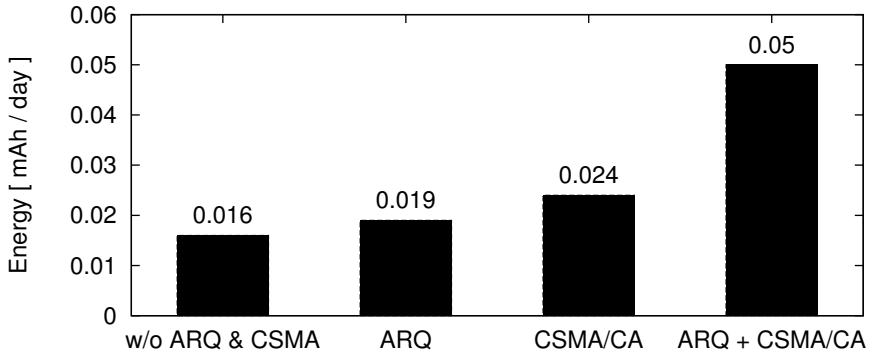


Figure 8.2.4: Energy consumed for data reception with LETED

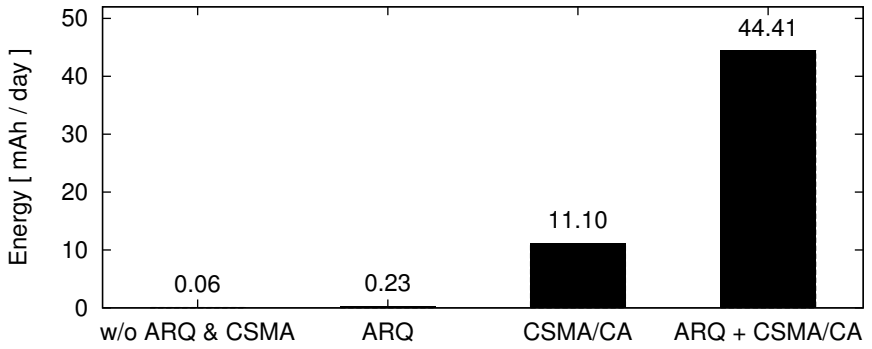


Figure 8.2.5: Energy wasted in passive slots of LETED

9 Conclusions

This work addressed tough challenges of particular sensor network applications with Quality of Service needs. That is, nodes must work for a long time, support short end-to-end delays and send data reliably in multi-hop networks.

Firstly, Distributed Low Duty Cycle MAC (DLDC-MAC) was presented, which serves as a basis for the remaining solutions to the challenges mentioned previously. In short, DLDC-MAC keeps nodes mostly in the sleep state and synchronizes wake-up times between neighbors. Secondly, based on the staggered schedule, introduced in DMAC and Q-MAC protocols, this work presented the LETED (Limiting End-to-End Delays) approach that achieves good results in lifetime and delays. For example, sensor nodes support end-to-end delays of 5 seconds and work almost 3 years with off-the-shelf hardware platforms. Other approaches cannot support such long lifetimes in similar scenarios.

Such good results of LETED stem from the novel solutions that reduce idle listening:

1. ILA (Idle Listening Avoidance) exploits features of commercially available transceivers, quickly detects an idle channel, and powers down the radio with almost no delay.

2. Owing to prediction of future drift based on the moving average filter, nodes use short guard times to compensate drift between neighbors.

Apart from solutions to the above-said challenges, this work introduced a variety of empirical and simulative results. These results may serve as a basis for further research of protocols for wireless sensor networks:

- Indoor and outdoor drift experiments confirmed that not only the drift parameter of oscillators, but also other factors can influence relative drift among nodes. Nonetheless, nodes receive most frames within a small drift window. For example, about 98% frames were affected by drift 8x smaller than the worst case, which is based on the oscillator parameter.

- This work presented the results of experiments with ARQ and CSMA/CA. They confirm that applying CSMA/CA in sensor networks is not advantageous,

as it mainly results in energy waste and does not provide any significant gain in the connection quality. On the contrary, ARQ improves the reception rate and slightly affects energy consumption.

- A two-week test of DLDC-MAC in an office experiment provides results relevant for other protocols too, mainly at the Data Link and Network layers. For example, the experiment confirmed that communication over long distance may suffer from high packet losses. In this case, multi-hop paths should be applied. Therefore, MAC protocols should deliberately abandon such poor links, if there are multi-hop paths available.

DLDC-MAC and LETED were compared with other state-of-the-art protocols by using the lifetime estimation model, which considers 27 hardware and software parameters. The model can be applied to examine end-to-end delays and lifetime of various MAC protocols, which are based either on a schedule or on Preamble Sampling. By doing so, researchers can quickly obtain lifetime results for different scenario parameters and hardware platforms.

Although the solutions presented here achieve better results than other protocols, they possess drawbacks as follows:

1. Since LETED and DLDC-MAC are based on TDMA, they must synchronize their wake-up times and therefore rely on the crystal oscillator. Should oscillators not work properly, these protocols cannot provide wake-up synchronization, and the communication is impossible. On the contrary, protocols based on Preamble Sampling work well even with non-functioning oscillators.

2. As LETED and DLDC-MAC handles various problems, which stem mainly from unreliable communication and clock drift, they need extra memory to implement remedies to them. On the Tmote Sky hardware platform, they occupy together with the TinyOS operating system more than 40 kB of flash memory, leaving less than 8 kB to other software. Simpler protocols, like Preamble Sampling, need about 10x less memory. However, with current development in embedded hardware, sensor nodes are equipped with more memory than previous generations. It solves partly the problem of the large code size.

9.1 Future Work

Since this work focused mainly on ensuring long lifetimes of nodes and short end-to-end delays, it did not consider thoroughly the aspects of reliable communication. Besides, it evaluated only major solutions of the Data Link Layer: ARQ and

CSMA/CA. Thus, it neglected solutions of other layers and potential gains of cross-layer cooperation. For instance, an obvious way to solve the problem of packet losses is to transmit data over different paths simultaneously. In the simplest case, source nodes send the same frames over various paths, resulting in redundancy. Obviously, it involves extra transmissions and increase energy consumption but can improve the communication quality. Although such means were addressed in other research works, they did not consider applications with a low duty cycle. Therefore, future work will focus on reliable communication, provided by several communication layers, and their impact on lifetime in sensor network applications with a low duty cycle.

The idea of load balancing was not considered, that is, alternating routes from sources to the sink. Thus, the same routes are used 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 prevent such risks. However, if nodes maintain a wake-up schedule along paths, like LETED does, they need to set up a new schedule on route change. It results in an extra communication overhead, and nodes may fail to support short end-to-end delays after the source changed the route. 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, e.g., 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.

The solutions presented in this work achieved good results in the lifetime of nodes and end-to-end delays. However, these solutions can be improved and many open issues should still be addressed in future research efforts:

- 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.

- DLDC-MAC neglects the problem of missing beacons when discovering neighbors. As a result, nodes may not learn about some neighbors, leading to various problems. For example, since routing protocols rely on the neighbor list provided by the MAC layer, they will not find the best path to the sink. Besides, if nodes with DLDC-MAC are not aware of some neighbors, they cannot prevent collisions with them. Therefore, future work will investigate energy-efficient solutions that tackle the problem of neighbor discovery.

- Crystal oscillators considered in this work have precision of ± 20 ppm. There are, however, cheap temperature-compensated crystal oscillators (TCXOs) that provide frequency stability of ± 2 ppm and are designed for low power. By applying drift prediction approaches to TCXOs, nodes can use guard times shorter than presented in this work and achieve better results in lifetime. Therefore, the solutions to guard times based on TCXO need empirical evaluations.

Bibliography

[1] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: a media access protocol for wireless LAN's. In *ACM SIGCOMM Computer Communication Review*, 1994.

[2] M. Brzozowski, R. Karnapke, and J. Nolte. IMPACT - A Family of Cross-Layer Transmission Protocols for Wireless Sensor Setworks. In *Proceedings IPCCC*, 2007.

[3] 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 ISADS*, 2009.

[4] M. Brzozowski, H. Salomon, and P. Langendoerfer. ILA: Idle Listening Avoidance in Scheduled Wireless Sensor Networks. In *Proceedings WWIC 2010.*

[5] M. Brzozowski, H. Salomon, and P. Langendoerfer. On Efficient Clock Drift Prediction Means and their Applicability to IEEE 802.15. 4. In *Proceedings EUC 2010.*

[6] M. Brzozowski, H. Salomon, and P. Langendoerfer. Completely distributed low duty cycle communication for long-living sensor networks. In *Proceedings EUC*, 2009.

[7] M. Brzozowski, H. Salomon, and P. Langendoerfer. Limiting End-to-End Delays in Long-Lasting Sensor Networks. In *Proceedings MobiWac*, 2010.

[8] M. Brzozowski, H. Salomon, K. Piotrowski, and P. Langendoerfer. Cross-Platform Protocol Development for Sensor Networks: Lessons Learned. In *Proceedings SESENA*, 2011.

[9] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: Ultra-Low Power Data Gathering in Sensor Networks. In *Proceedings IPSN 2007.*

[10] Q. Cao, T. Abdelzaher, T. He, and J.Stankovic. Towards Optimal Sleep Scheduling in Sensor Networks for Rare-Event Detection. In *Proceedings IPSN 2005.*

[11] connectBlue. Product Brief OWLAN211g http://www.connectblue.com.

[12] A. El-Hoiydi. Aloha with Preamble Sampling for Sporadic Traffic in Ad Hoc Wireless Sensor Networks. In *Proceedings ICC 2002*.

[13] A. El-Hoiydi. Spatial TDMA and CSMA with Preamble Sampling for Low Power Ad Hoc Wireless Sensor Networks. In *Proceedings ISCC 2002*.

[14] A. El-Hoiydi and J.-D. Decotignie. WiseMAC: An Ultra Low Power MAC Protocol for Multi-hop Wireless Sensor Networks. In *Proceedings ALGOSENSORS 2004*.

[15] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proceedings MobiCom 1999*.

[16] S. Ganeriwal, D. Ganesan, H. Shim, V. Tsiatsis, and M. B. Srivastava. Estimating Clock Uncertainty for Efficient Duty-Cycling in Sensor Networks. In *Proceedings SenSys 2005*.

[17] S. Hoeckner, A. Lagemann, and J. Nolte. Integration of Event-Driven Embedded Operating Systems Into OMNet++ - A Case Study with Reflex. In *Proceedings SIMUTools*, 2009.

[18] B. Hohlt and E. Brewer. Network Power Scheduling for TinyOS Applications. In *Proceedings DCOSS 2006*.

[19] B. Hohlt, L. Doherty, and E. Brewer. Flexible Power Scheduling for Sensor Networks. *Proceedings IPSN 2004*.

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

[21] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[22] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2005.

[23] B. Karp and H.T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings MobiCom 2000*.

[24] S. Kim, R. Fonseca, and D. Culler. Reliable Transfer on Wireless Sensor Networks. *Proceedings IEEE SECON 2004*.

[25] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. TinyOS: An Operating System for Sensor Networks. In *Ambient Intelligence*. 2005.

[26] E.-Y.A. Lin, J.M. Rabaey, and A. Wolisz. Power-Efficient Rendez-vous Schemes for Dense Wireless Sensor Networks. In *Proceedings ICC 2004*.

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

[28] H. Liu, H. Ma, M. El Zarki, and S. Gupta. Error control schemes for networks: An overview. *Mobile Networks and Applications*, Volume 2:167–182, 1997.

[29] 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*, 7:863–875, 2007.

[30] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The Flooding Time Synchronization Protocol. In *Proceedings SenSys 2004*.

[31] Maxim Integrated Products. DS32kHz: 32.768kHz Temperature-Compensated Crystal Oscillator http://datasheets.maxim-ic.com/en/ds/DS32kHz.pdf, 2007.

[32] R. Min and A. Chandrakasan. Top Five Myths about the Energy Consumption of Wiireless Communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7(1):65–67, 2003.

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

[34] R. Musaloiu-E., C.J.M. Mike Liang, and A. Terzis. Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks. In *Proceedings IPSN 2008*.

[35] Ch. E. Perkins and E. M. Royer. Ad-hoc On-Demand Distance Vector Routing. In *Proceedings WMCSA*, 1999.

[36] K. Piotrowski, P. Langendoerfer, and S. Peter. tinyDSM: A Highly Reliable Cooperative Data Storage For Wireless Sensor Networks. In *Proceedings CTS 2009*.

[37] J. Polastre, J. Hill, and D. Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proceedings SenSys '04*.

[38] Proxim wireless. ORiNOCO 11b Client PC Card datasheet: http://www.proxim.com/learn/library/datasheets/11bpccard.pdf.

[39] J.M. Rabaey, M.J. Ammer, J.L. da Silva Jr, D. Patel, and S. Roundy. PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking. *Computer*, 33:42–48, 2000.

[40] T.S. Rappaport et al. *Wireless Communications: Principles and Practice*. Prentice Hall PTR New Jersey, 2002.

[41] Y. Sankarasubramaniam, Ö.B. Akan, and I.F. Akyildiz. ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks. In *Proceedings MobiHoc 2003*.

[42] Sanyo. Twicell HR-3UTG datasheet. http://www.eneloop.info.

[43] 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 ISLPED 2008*.

[44] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava. Optimizing Sensor Networks in the Energy-Latency-Density Design Space. *IEEE Transactions on Mobile Computing*, 2002.

[45] P. Sommer and R. Wattenhofer. Gradient Clock Synchronization in Wireless Sensor Networks. In *Proceedings IPSN 2009*.

[46] P. Sommer and R. Wattenhofer. Symmetric Clock Synchronization in Sensor Networks. In *Proceedings REALWSN 2008*.

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

[48] T. Van Dam and K. Langendoen. An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings SenSys 2003*.

[49] A. Varga. The OMNeT++ Discrete Event Simulation System. In *Proceedings ESM*, 2001.

[50] N. A. Vasanthi and S. Annadurai. Energy Efficient Sleep Schedule for Achieving Minimum Latency in Query based Sensor Networks. In *Proceedings SUTC '06*.

[51] K. Walther and J. Nolte. A Flexible Scheduling Framework for Deeply Embedded Systems. In *Proceedings AINAW*, 2007.

[52] C.Y. Wan, S.B. Eisenman, and A.T. Campbell. CODA: Congestion Detection and Avoidance in Sensor Networks. In *Proceedings SenSys 2003.*

[53] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings INFOCOMM 2002.*

[54] J. Zhao and R. Govindan. Understanding Packet Delivery Performance In Dense Wireless Sensor Setworks. In *Proceedings SenSys 2003.*

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

Nomenclature

µC Microcontroller

ACK ACKnowledgment

ADC Analog-to-Digital Converter

ARQ Automatic Repeat reQuest

ASIC Application-Specific Integrated Circuit

B-MAC Berkeley MAC

BER Bit Error Rate

BOT Back-Off Time

BPSK Binary Phase-Shift Keying

CCA Clear Channel Assessment

CIP Critical Infrastructure Protection

CODA COngestion Detection and Avoidance

CPU Central Processing Unit

CSMA/CA Carrier Sense Multiple Access With Collision Avoidance

CTS Clear-To-Send

DLDC-MAC Distributed Low Duty Cycle MAC

EDT Event Detection Time

ESRT Event-to-Sink Reliable Transport

FEC Forward Error Correction

FFD Full-Function Device

FPS Flexible Power Scheduling

FTSP Flooding Time Synchronization Protocol

GPSR Greedy Perimeter Stateless Routing

GT Guard Time

ILA Idle Listening Avoidance

ISR Interrupt Service Routine

LAN Local Area Network

LDC Low Duty Cycle

LPL Low Power Listening

LPP Low Power Probing

LR Linear Regression

MAC Medium Access Control

MADC Moving Average Drift Compensation

MCU Microcontroller

MF Mobility Framework

NACK Negative ACKnowledgment

OLS Ordinary Least Squares

OQPSK Offset Quadrature Phase-Shift Keying

OS Operating System

OSI Open Systems Interconnection

PER Packet Error Rate

PHY Physical Layer

PS Preamble Sampling

QoS Quality of Service

RAM Random-Access Memory

RATS Rate Adaptive Time Synchronization

RFD Reduced-Function Device

RICER Receiver Initiated CyclEd Receiver

ROM Read Only Memory

RTS Request-To-Send

RTS/CTS Request-To-Send / Clear-To-Send

RX Receive

RxINT Receive Interrupt

S-B Schedule-Based

S-MAC Sensor-MAC

SFD Start Frame Delimiter

SINR Signal to Interference and Noise Ratio

SNR Signal to Noise Ratio

STEM Sparse Topology and Energy Management

SYNC Synchronization

T-MAC Timeout-MAC

TCXO Temperature-Compensated Crystal Oscillator

TDMA Time Division Multiple Access

TICER Transmitter Initiated CyclEd Receiver

TSP Time Synchronization Protocol

TX Transmit

WiseMAC Wireless Sensor MAC

WLAN Wireless LAN