BAKKALAUREATSARBEIT

Establishing Wireless Time-Triggered Communication using a Firefly Clock Synchronization Approach

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Bakkalaureus der Technischen Informatik

unter der Leitung von

Univ.Ass. Dipl.-Ing. Dr.techn. Wilfried Elmenreich
Institut für Technische Informatik 182

durchgeführt von

Robert Leidenfrost
Matr.-Nr. 0426381
Wichtelgasse 39/3/19, A-1160 Wien, Austria

Wien, im September 2007 .....

Abstract

The application area of wireless sensor networks often include the deployment in harsh environments. Therefore, the devices are mainly battery-powered but nevertheless must provide a lifetime in the order of months to years. This requires the use of energy efficient protocols. Such protocols need to support duty-cycling, where a node only enables the transceiver module for a short time if it expects a message or wants to transmit to other nodes. However, whereas most protocols are aimed at reducing the duty-cycle, our approach also establishes a time-triggered approach, which takes advantage of the *a priori* known transmission events. Such a technique requires a global notion of time and therefore is based on algorithms for maintaining synchronization. This ensures that every node has the same local view of the global time. The controllers running this algorithm often have to be very inexpensive and make use of the cheap on-chip oscillators which generally entail big clock drifts and thus requires a frequently resynchronization and/or a precise clock drift calibration. Additionally, the network topology and unidirectional communication connections also have a deep impact on the precision of the synchronization algorithm. For this reason we introduce an alternative biologically-inspired synchronization algorithm together with a clock rate calibration scheme to build up synchronicity. Synchronicity is the ability to organize simultaneous collective actions across a sensor network. Hence our approach is based on distributed synchronous clocking and is a type of internal synchronization. In contrast to centralized clock synchronization schemes, the distributed synchronization approach has the inherent advantage for complete scalability and graceful degradation. This thereby achieved common notion of time is also used to establish a time-triggered protocol. In order to perform a rapid development, this approach was first simulated with a modified probabilistic wireless sensor network simulator for different network topologies and parameter choices. The results are then compared with the outcomes of several testbed experiments based on ZigBee nodes from Atmel.

Acknowledgements

I would like to thank my advisor Wilfried Elmenreich who enabled me to write this thesis. His comments and feedback were very helpful in order to improve the quality of the work.

Moreover, I gratefully acknowledge Iris for her patience and in particular my parents who made this study possible.

Contents

1 Introduction

Since technology improvements in the last decade have made smaller and more inexpensive sensor nodes possible, sensor networks have become a big research field. In the beginning such a network was composed of small numbers of sensor nodes whereas these nodes were wired to a central processing unit. Nowadays sensor networks are mostly built-on wireless technology with a big number of sensing nodes with local processing. Hence, that progress enabled the use of sensor networks in a variety of applications. For example, the monitoring of a phenomenon could be a difficult challenge if the exact location where to place a sensing element is not known. Alternatively, distributed sensing allows a closer placement to the phenomenon and therefore a better Signal-to-noise Ratio (SNR) [EGPS01]. However, in most cases the environment where such sensing nodes are in use is harsh and usually does not provide an infrastructure. Such a malicious environment challenges some design constraints like robustness, low power consumption, physical size, network discovery, lifetime and many others that vary from sensor to sensor [CK03]. Thus, the sensors must rely on local, finite, and relatively small energy sources. Especially, communication is a key energy consumer. In order to be economically feasible, the devices generally must have a lifetime on the order of months to years [RSPS02] without battery replacement. According to [YHE04], the major sources of energy waste are packet collisions, overhearing, control packet overhead, and idle listening whereas in many MAC-protocols such as IEEE 802.11 more than 50 percent are spent on idle listening [YHE04]. Several approaches have been proposed to improve energy efficiency focusing mostly on clustering mechanisms, routing algorithms, energy dissipation schemes, sleeping schedules, and so on. Still a maximized network lifetime requires the use of a well-structured design methodology and must consider the tradeoffs between energy consumption, system performance, and operational fidelity.

1.1 Motivation and Objectives

Typical energy efficient implementations keep the nodes largely inactive for most of the time and become active only for a short time if something is detected. This results in a periodic sleep/listen approach which reduces the idle

listening and is called sleep scheduling. A simple solution for this concept is based on coordination which means that an ensemble of nodes must agree on the same schedule and therefore will sleep and listen at the same time. Different distributed algorithms for choosing and maintaining such a schedule have already been proposed. A good survey on clock synchronization algorithms can be found in [SBK05]. In other words clock synchronization in sensor networks, especially in multi-hop wireless ad-hoc networks, is an important necessity to share a common view of the local clock time. Without a precise clock synchronization, the mobile devices do not wake up at the same time and thus the power management operation will not work well.

Another requirement is a reliable and collision-free communication among the participants, especially when the devices are communicating over wireless media. This requisite can be met through establishing a TDMA medium access scheduling which also eliminates the problem if the nodes are several hops away and a single broadcast cannot reach them all. To achieve this goal, every node must have the same notion of time which demands the implementation of a synchronization approach.

In our work we have used the biologically-inspired Firefly algorithm for synchronicity to achieve coordinated sleeping. According to [WATP+05] synchronicity is the ability to organize simultaneous collective actions across a sensor network. Hence our approach is based on distributed synchronous clocking and is a type of internal synchronization. In contrast to centralized clock synchronization schemes, the distributed synchronization approach has the inherent advantage for complete scalability and graceful degradation.

A further important problem is that the underlying controllers of the individual nodes often have to be very inexpensive and the applications makes use of the imprecise on-chip oscillators which have big clock drifts of up to $\pm 100000 ppm$. This requires a frequently resynchronization. On this account, along with the clock-state synchronization approach based on the Firefly algorithm, we introduced a clock-rate calibration. This enables an infrequent resynchronization and therefore improves energy consumption.

In this work we present a low duty-cycle protocol which works without beacons using the RFA [WATP+05] for network synchronicity. This approach is appropriate for ad-hoc sensor networks where the topology is not known or might change. Based on this synchronization service, a TDMA scheme was introduced to divide a period into several slots with different activities. More precisely, such activities for an individual node can be the broadcasting of a message, listening to a message, executing of different tasks, or simple a sleep interval. In general this structure is similar to a TTP. However, in contrast to TTPs our TDMA scheme is not based on time synchronization but on synchronicity. Hence we have no reference clock or global coordination for defining

the absolute time duration of an interval.

1.2 Structure of the Thesis

The thesis is structured as follows: Chapter 2 gives an introduction into the basic terms and concepts used throughout the work. Further, Chapter 3 reflects the results of related work using the Firefly algorithm. The concept of this biologically-inspired algorithm for clock state synchronization and the clock rate calibration approach are described in Chapter 4. Next, Chapter 5 presents JProwler, a probabilistic wireless sensor network simulator and the modifications made to it so that the simulation model comes closer to reality. Chapter 6 represents the simulation results of our algorithm based on several experiments including different network topologies and parameter choices. Moreover, Chapter 7 explains the used testbed environment and further the implementation of the algorithm whereas the results of these experiments are discussed in Chapter 8. Finally, the thesis ends with a conclusion in Chapter 9 summarizing the key results of the presented work and giving an outlook on what can be expected from future research in this area.

2 Concepts

This chapter gives an overview on the *terms* and *concepts* used for clock synchronization in distributed systems that are required to understand this work. It should be noted that the semantics of the introduced terms differs due to different applications and purposes of distributed systems and clock synchronization.

This chapter first introduces the terms of distributed systems and their composition. Afterwards the focus lies on clocks, time standard and formats, and clock synchronization. The third part gives an overview of the Time-Triggered-Architecture.

2.1 Distributed Systems

The term "Distributed Systems" has various definitions depending on the application and the research field. In [Pau02], Paulitsch cited two definition which also best fits to this work. Generally, a distributed system comprises of different nodes which are able to interact among each other. In this work the communication is performed wireless and each node is represented by a sensor node. Further, the main task in our work is to gather information from all nodes which therefore requires special protocols concerning energy and throughput. However, such a decentralized structure consists of different basic components which are described in the next paragraph.

Nodes, Communication Systems, Software, and States. A distributed system consists of multiple, autonomous computers, which are called *nodes*. A node consists its own hardware (e.g. oscillator, processor, memory, interfaces) and software (e.g. application programs, operating systems) to perform a well-defined distributed communication pattern. The *software* is an algorithmic description that determines the behaviour of a node's system. In our context the software also determines the coordination of the activities for each individual sensor node to maintain a *shared state* which defines the relevant parts of a *local state* of the distributed system. Additionally, the node software can be divided into two data structures [Kop97, p. 76]: The *initialization state (i-state)* and

the *history-state (h-state)*. The *i-state* is a static data structure that contains the re-entrant program code and the initialization data and is usually stored in a Read-Only Memory (ROM). On the other hand, the h-state reflects the dynamic data structure of the node which can change its content over computational progress and must be stored in a Random Access Memory (RAM). The nodes are interconnected by a network called *communication system* which allows them to communicate among each other respectively the exchange of data. *The state enables the determination of a future output solely on the basis of the future input and the state the system is in.*[MT89]. Further the *global state* of a system is defined as the union of the local states of its components [Sch93].

Components. In our work we use the term *component* to describe a part of the distributed system which cannot be decomposed for a given level of abstraction. So a component can be a node, the software running on it, or the node's state. In contrast to a component, a system can be decomposed into subsystems.

2.2 Clocks, Time, and Clock Synchronization

This section introduces concepts and terms used for clock synchronization in distributed systems and mainly refers to [Kop97, p. 45].

2.2.1 Concepts of Clocks

In distributed systems, the participants often measure events which occur at a specific point in time. This representation of abstract point in time is based on the node's own local clock which is independent and individual. On this account, the same event can result in different absolute time representations. If the global tasks and algorithms are only based on the trivial local environmental measurements of a single node, then this concept is adequate. However, most applications and algorithms used in sensor networks are based on the comparison of events measured by many different nodes. In other words every node must have a local view of the global time so that the measured events on different nodes can be reordered in a distinct way. For this concept some new terms must be introduced.

Physical Clock. A physical clock contains a *counter* and an *oscillator mechanism* which periodically generates events to increase the counter. This periodic event is called *microtick* whereas the duration between any two microticks is

called *granularity*. The granularity is usually conditioned by the parameters of the physical oscillator (e.g. oscillator type, frequency, ambient temperature, etc.) and therefore can lead to a *digitalization error* in time measurement. Hereinafter the time of the physical clock k is called $microtick^k$. Further, microtick i of clock k is denoted by $microtick_i^k$. The granularity g of clock k is defined as the nominal number n^k of microticks of a reference clock between two microticks of clock k.

Reference Clock. A reference clock z usually has a very small granularity g^z and can be seen as a granular representation of real-time. Because the granularity is usually about some femtoseconds ($10^{-15}sec$), the digitalization error is negligible. Whenever an event is timestamped by the reference clock $Clock(event)$, then if z is the single reference clock in the system, $z(e)$ is called $absolutetimestamp$ of event e.
The concept of reference clock allows the use of simple models, because in contrast to the dense real-time, the reference clock represents the real-time as a natural number. This simplifies the ordering of timestamped events to simple integer arithmetics.

Drift Rate of a Clock. In reality, every clock has a clock drift even if it is very small. That is, after some time the clock drifts apart from another clock and is therefore usually denoted with respect to the nominal number n^k of microticks of a reference clock. So the drift of a clock k is determined by the ratio between the measured duration of a granule of this clock with the reference clock z and the nominal number of microticks of the reference clock for that granule. Because the clock drift can change over time, it is calculated for a distinct granule of clock k between microtick i and microtick $i+1$:

$$drift_i^k = \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k}$$

Usually, many clocks are relatively accurate and thus have a clock drift very close to 1. For that reason, another term called *drift rate* is introduced. The drift rate of clock k at the instant of microtick i is indicated by ρ_i^k and declares the absolute deviation of the clock drift to the perfect clock:

$$\rho_i^k = \left| \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k} - 1 \right|$$

Because typical drift rates are in the range of 10^{-2} to $10^{-10}s/s$, the drift rate is often quoted in *parts per million* (ppm). A typical watch crystal has about 20ppm. That is $\frac{20}{10^6}sec/sec$ and results in an error over a day of about $1.73sec/day$.

Virtual Clocks In our work a clock must have the possibility for adjusting the frequency and further for changing the state, i.e.,the content of the counting register. The second condition is usually no problem whereas the change of the frequency of an oscillator seems to be a problem for so, because many controllers have generally implemented commercial off-the-shelf (COTS) oscillators so far which cannot adjust their frequency. A more sophisticated solution could be the use of Voltage Controlled Crystal Oscillator (VCXO)s. However, this is too expensive and therefore unthinkable for sensor networks. For this reason, the problem must be solved at the software level and thus abstracts from hardware dependent parameters.

A realization of such a virtual clock providing the two adjustment criteria is based on the concept of macroticks. Therefore, a macrotick is represented by a tick of the virtual clock and comprises a number of microticks which are generated by a hardware clock due to an oscillator. The granularity between two microticks is based on the frequency f_{osc} of this oscillator and the nominal granularity g of the virtual clock is based on the number of comprised microticks. Thus, a counter increments a hardware register due to this oscillator and continuously compares this value with g. If the comparison matches, then the register will be reset to 0 and the hardware raises an interrupt which corresponds to a macrotick and a software counter is incremented. So the clock rate-adjustment for a clock k can be simply performed by changing the threshold g of the hardware counter based on an adjustment value, denoted by $H^k(t)$. The frequency of the virtual clock VC with the hardware clock k based on an oscillator that has a frequency f_{osc} can be calculated as follows:

$$f_{VC}(t) = \frac{f_{osc}}{g + H^k(t)}$$

Offset. The *offset* is defined as the time difference of two clocks j and k having the same granularity at the microtick i measured with respect to the reference clock z.

$$offset_i^{jk} = \left| z(microtick_i^j) - z(microtick_i^k) \right|$$

The same applies for virtual clocks j and k, but is denoted by

$$O_i^{jk} = \left| z(VT_i^j) - z(VT_i^k) \right|$$

Precision. In an ensemble of clocks, the *precision* defines the maximum offset between any two clocks during a period of interest. So the precision of an ensemble with n clocks $[1 \dots n]$ over all interesting microticks i is defined as

$$\Pi = \max_{\forall 1 \leq j,k \leq n} \{offset_i^{jk} | \forall i\}$$

or with respect to virtual clocks as

$$\Pi = \max_{\forall 1 \leq j,k \leq n} \{O_i^{jk} | \forall i\}.$$

Accuracy. The *accuracy* denotes the maximum offset of a given clock with respect to a reference clock over an interval of interest. So for a clock k with the reference clock z, the accuracy is defined by

$$accuracy^k = \max_{\forall i} \{offset_i^k\}$$

where i can take in all microticks that are of interest. This equally applies to virtual clocks and is specified by:

$$accuracy^k = \max_{\forall i} \{O_i^k\}$$

2.2.2 Clock Synchronization.

Clock synchronization is an important mechanism in every distributed system. The need for distributed synchronous clocking can be energy constraints as well as the global timestamping of events. Particularly sensor networks are often multi-hop topologies and therefore usually need a decentralized solution for maintaining synchronization. However, the fact that the clocks have different clock drifts make it difficult to bring the time of the clocks in close relation with respect to each together. A general approach is a frequently resynchronization, but this is not applicable for sensore networks as this will increase the communication and therefore also strongly increases the energy consumption. The *precision* is a typical measure for the quality of clock synchronization.

Internal Clock Synchronization. *Internal clock synchronization* is required in an ensemble of clocks where the *precision* must be kept to a minimum. This is done by mutual resynchronization of the clocks. It should be noted that this type of synchronization does not necessarily mean synchronization to real-time, because all clocks can have a similar clock drift and therefore may be synchronized within a very good precision, but the accuracy with respect to real-time might be unacceptable. The duration for a period of resynchronization is called *resynchronization interval* R_{int}. So after a resynchronization event, the clocks still run free and may drift apart with a defined maximum drift rate ρ. Note that ρ does not declare if a clock ticks faster or slower than the reference clock. At the end of each resynchronization interval, the offset of the clock with respect to a reference clock can be $\pm\rho \cdot R_{int}$. Therefore, a new term called *drift offset* Γ is introduced with $\Gamma = 2 \cdot \rho \cdot R_{int}$. Assumed that ρ

dedicates the maximum possible drift rate in an ensemble of clocks, then the drift offset indicates the maximum absolute divergence of any two clocks in this ensemble during the resynchronization interval R_{int}.

The Synchronization Condition. Due to unpredictable communication jitter and clock drift, the synchronization is usually bounded within a small interval and affects the precision. Further, the different synchronization algorithms are not perfect and thus are limited again to a small synchronization inaccuracy called *convergence function* Θ. All these facts degrades the precision and can be brought into an equation named *synchronization condition*.

$$\Theta + \Gamma \leq \Pi$$

Lower Bounds for Internal Clock Synchronization. Any *convergence function based synchronization algorithm* has a lower bound for the optimal maximum deviation.

Srikanth and Toueg showed that in an ensemble of clocks, the maximum drift rate achievable by a fault tolerant internal clock synchronization algorithm is lower bounded to the maximum drift rate of all clocks in that ensemble [ST87].

Another important lower bound was introduced in [LL84] and depends on the number of clocks N and the communication jitter ε. Assuming the clocks have no clock drift, the results from Lundelius and Lynch state that an ensemble of clocks cannot be synchronized to a better precision than

$$\Pi = \varepsilon \cdot \left(1 - \frac{1}{N}\right).$$

Last but not least, according to [DHS84], any clock synchronization algorithm applied on an ensemble of perfect clocks with no clock drift and for any communication network graph G, the precision has a lower bound of:

$$\Pi \geq \frac{U_G}{2}$$

U_G declares the *uncertainty* in transmission time of the network graph G and is defined as the maximum of any minimal *delay jitter* $\varepsilon(\sigma)$ whereas σ can be any communication sequence starting with node p and ending with node q. In more detail

$$U_G = \max\{\min\{\varepsilon_G(\sigma) \mid \forall \sigma \in S(p,q)\} \mid \forall p, q \in G; p \neq q\}$$

whereas $S(p, q)$ contains all possible sequences of nodes starting with p and ending with q. Further, $\varepsilon_G(\sigma)$ is referred to as the delay jitter of the communication sequence $\sigma =< p_0, \ldots, p_n >$ consisting of nodes p_i, for i from 0 to n, and is calculated as followed:

$$\varepsilon_G(\sigma) = \sum_{i=0}^{n-1} (\varepsilon_G(p_i, p_{i+1}))$$

Moreover, $\varepsilon_G(p_i, p_{i+1}) = U_G(p_i, p_{i+1}) - L_G(p_i, p_{i+1})$ and defines the variation in transmission and processing time for messages between p_i and p_{i+1}. Consequently, $U_G(p_i, p_{i+1})$ defines the upper bound and $L_G(p_i, p_{i+1})$ the lower bound on transmission and processing time for messages between p_i and p_{i+1}. In addition to that, the authors proof that there exist algorithms such that for all communication networks the precision is not greater than the *uncertainty* U_G.

External Clock Synchronization. In an ensemble of clocks, *external clock synchronization* always requires one or more reference clocks that are not part of this ensemble. This kind of synchronization is performed with a periodically resynchronization of the clocks with respect to the reference clock and thus keeps the clock within a bounded interval of the reference clock. The quality of external clock synchronization is measured by the accuracy. Additionally, external synchronization of an ensemble of clocks with an accuracy A results in internal synchronization with a precision of at most $2 \cdot A$. The converse is not true.

Clock State Correction vs. Clock Rate Correction. Clock synchronization usually uses two different approaches to achieve an accurate precision: State correction and rate correction. For *state correction* the calculated correction term is immediately applied to the local clock whereas *rate correction* modifies the rate of a node's clock. Clock rate correction can be implemented either by changing the number of microticks for some macroticks or in the case of a VCXO by adjusting the supplied voltage. Moreover, assumed that the rate correction algorithm is usually not perfect, it might occur that the drift rates of all clocks continuously increase or decrease in the same way. To avoid this common drift, the rate correction algorithm should incorporate a common drift compensation. This is done by an evaluation of the average rate correction terms among all clocks which should be close to zero.

Principle of Operation of Distributed Clock Synchronization. Every distributed clock synchronization algorithm usually proceeds in the same way and can be distinguished in three different phases:

1. Phase: Collection of clock time values. In order to achieve a good precision, the algorithm must satisfy that in any synchronization period, every node obtains the local clock state of the global time counter of all other participating nodes. Otherwise, the precision degrades or the synchronization may completely fail.

2. Phase: Calculation of correction values. Depending on the convergence function of the synchronization algorithm and on all or some of the collected clock states, every node calculates a *correction value* for the local clock representing the global time counter. In the case of a correction value greater than a predefined precision, the synchronization algorithm must ensure that either the node deactivates itself or the other nodes ignore it until it is again synchronized.

3. Phase: Clock correction. Lastly, every node has to apply the correction term from Phase 2 to the local clock.

2.3 Drift of Oscillators

The drift of an oscillator mainly depends on the drift rate of the underlying technology, e.g., crystal or RC-oscillator. For this reason the next paragraphs describe the drift of crystals and RC-oscillators in more detail.

Drift of crystal oscillators. According to [Sch88], the drift rate of a crystal oscillator consists of a *systematic error* and a *stochastic error* whereas the systematic error, that determines the nominal drift rate, is constant and the stochastic error, that is a random drift within a specified interval, changes over time. This is true for short observation periods in the range of seconds and minutes. On the other hand, for longer observations the systematic error changes similarly to the stochastic error. The stochastic drift rate is usually two orders of magnitudes smaller than the nominal drift rate and is therefore negligible [Sch96]. It should be noted that this generally applies to oscillators based on crystals, but the clock drift also changes due to environmental influences and mainly depends on the oscillator type and the cut of the crystal. [Vig00] categorizes the environmental impacts into time-, temperature-, acceleration-, ionizing radiation-, and other influences. The time can influence the frequency in a short term (noise) or in a long term (aging). Further, the temperature has the biggest influence on the crystal's frequency and is distinguished between *static frequency-temperature effects* and *dynamic frequency-temperature effects*

(e.g. warm-up, thermal shock). The other categorizations hardly affect the crystal and are not explained in detail in this work. The short-term stability of crystal oscillators are discussed in detail in [Sch95].

As mentioned above, the characteristics of a quartz crystal mainly depends on the angle of cut from a reference. Many different cuts have been developed for different applications, e.g, AT-, BT-, CT-, DT-, ET-, GT-, MT-, NT- and SC-cut. AT-cut is the most commonly used of the "high-frequency" cuts. [Bec56] states, that the crystal-frequency mainly depends on the temperature and can be visualized in a frequency-temperature-angle curve which is defined by some temperature coefficients of higher order whereas for the temperature range from -60 to $+100°C$, temperature coefficients of higher order than three can be neglected. Additionally, the frequency-temperature behaviour also depends on the ratio of dimension, order of overtone, shape of plate, and the type of mounting. Thereafter, under the assumption that the change of the three temperature coefficients with the angle of orientation is linear, the frequency-temperature-angle characteristics is dedicated by the expression

$$\frac{f - f_0}{f_0} = \frac{\Delta f}{f_0} = a_0(\theta_0) \cdot [T - T_0] + b_0(\theta_0) \cdot [T - T_0]^2 + c_0(\theta_0) \cdot [T - T_0]^3 + \left(\frac{\partial a_0(\theta)}{\partial \theta} \cdot [T - T_0] + \frac{\partial b_0(\theta)}{\partial \theta} \cdot [T - T_0]^2 + \frac{\partial c_0(\theta)}{\partial \theta} \cdot [T - T_0]^3 \right) \cdot (\theta - \theta_0)$$

where f denotes the measured frequency as a function of the temperature T and f_0 defines the frequency at the arbitrary temperature T_0. Further, the variables a_0, b_0, and c_0 are the corresponding first, second, and third order temperature coefficients whereas $\frac{\partial a_0(\theta)}{\partial \theta}$, $\frac{\partial b_0(\theta)}{\partial \theta}$, and $\frac{\partial c_0(\theta)}{\partial \theta}$ incorporate the derivatives with respect to the angle of the three temperature coefficients. Usually, for an angle of about $1°$, the derivatives can be neglected.

In the vicinity of a zero angle of orientation, the first order temperature coefficient a_0 is usually very small or zero. In that case two main types of frequency-temperature behaviour are distinguished: The first type has a dominant second order temperature coefficient c_0 whereas b_0 is very small. If so the frequency-temperature curve has a cubic form. Examples are AT-cut and GT-cut. Most other cuts can be assigned to the second type where the second order coefficient is predominant. This results in a parabolic frequency-temperature curve. Typical values for temperature coefficients of different cuts can be found in [BBL62, Bec56].

It should be noted that several oscillators with integrated temperature compensation have already been developed. For instance a Temperature Compensated Crystal Oscillator (TCXO) is based on a VCXO and therefore can maintain a frequency by adjusting the corresponding voltage. Another technique for

temperature compensation is based on digital processing (e.g. Microcomputer Compensated Crystal Oscillator (MCXO) and Digital Compensated Crystal Oscillator (DCXO)). Finally the Oven Controlled Crystal Oscillator (OCXO) uses an oven for frequency stabilization whereas the oven temperature must be higher than the maximum ambient temperature expected in the respective application. Last but not least, the frequency of these special crystals are relatively stable to temperature-variations, but on the other hand require more implementation size and can increase production costs.

Drift of RC-oscillators. RC-Resonators are generally used for internal-oscillators in controllers. Because it is cheaper to use the implemented oscillator than to extend the controller with an external crystal-oscillator, many applications for embedded systems make use of the internal RC-oscillator for generating the system clock. This can result in big problems, especially if the nodes in a distributed system have to accomplish a distributed clock synchronization based on such imprecise oscillators. However, a clock rate correction could slightly compensate this problem and thus improve the precision.

Various experiments on integrated uncompensated RC-oscillators [Atm06b, AP68, HSD+06, Sch95] show, that RC-oscillators are much more sensitive to temperature and supply-voltage variations than crystal oscillators. A typical frequency-temperature curve has a maximum at $0°C$ and looks like an inverted parable. From this it follows, that only the second temperature coefficient is dominant and is in the range of $b_0 \sim 2 \ldots 3 \frac{ppm}{°C^2}$. In contrast to crystals with a dominant second order coefficient (e.g BT-cut), the frequency of RC-oscillators may increase with a decreasing temperature up to a maximum at about $0°C$ which can be higher than the nominal frequency.

2.4 Firefly Synchronization

The synchronously flashing of thousands of fireflies is a very interesting phenomenon and was observed only by very few people. Because not all fireflies are able to synchronize their flashes, explorers and naturalists have reported such an occurrence only in the region stretching from India east and southeast to the Philippines and New Guinea. According to [BB76], the rhythmically flashing is reserved to the males whereas the females only respond to these flashes to begin a courtship. Further it is interesting to know, that different species have different flashing intervals reaching from $200ms$ to three seconds and can synchronize within a precision of about $20ms$. It should be noted that synchronization demands from the participants that they have an oscillating mechanism to control rhythmic activity which can be described as a resettable

pacemaker model. In detail, this mechanism is based on neural control systems which all depend on the charging and discharging of the capacitive membranes of the nerve cells. So in the case the charge reaches a specific level named *triggering level*, the firefly flashes and the membranes will be quickly discharged to a *basal level*. Additionally, the charge is hereinafter called phase. Different models inspired by such a biological synchronization were proposed in the last decades. The most important one are the phase-advance respectively the phase-delay model from Buck, the Pulse-coupled Biological Oscillators (PCO) model from Mirollo and Strogatz (MaS) (also called MaS model), and the RFA model. Other models are similar, but make different assumptions on the coupling strength and the propagation delay in wireless communication (e.g., [Abb90], [Ger96]).

J. and E. Buck also discussed different hypotheses for the reason of synchronously flashing. For instance, they proposed the *beacon hypothesis* stating that the combined light makes the beacon larger and stronger for attracting males and females from all directions which allows the males to find more females for a mating than they could by searching individually. However, this differs from Darwin's concept of evolution by natural selection of the fittest.

In [Buc88], J. Buck presents two different types of mechanisms used by such insects to synchronize with respect to each other. The first strategy is called *phase-advance synchronization* and the second one refers to as *phase-delay synchronization*. A simple simulation of both strategies implemented as a Java-Applet can be found at [Wil98].

Phase-Advance Synchronization Model This type of synchronization is based on the fact that simultaneously flashing is due to an alternate discharge and recovery of a battery-like mechanism. That means that the excitation of the insect linearly increases towards a threshold and then the firefly fires. Further, if an insect is stimulated near the time it would normally flash, then the it resets its phase and therefore flashes immediately. Thus, the flash periods are destined from the insect with the shortest period. A firefly species using this strategy is the common known American *Photinus pyralis* and is usually not observed to synchronize. The reason for this could be the irregular flashing rhythm. Compared to other species, an individual male of the Photinus pyralis has a cycle-to-cycle variability (measured in standard deviation / mean period) of about 1/20 whereas other species like *Pteroptyx effulgens* have a smaller variability of about 1/200.

Phase-Delay Synchronization Model In contrast to the phase-advance strategy, the *phase-delay synchronization* scheme does not advance the charge to the

triggering level but resets it to the *basal level* and, therefore, the firefly restarts and runs for its full normal duration before reaching the triggering level. This enables the insects to duplicate a wider range of flashing rhythms. In addition to that, this type of synchrony tends to be more regular and also achieves a better flash coherence due to the unneeded resetting in particular periods whereas the phase-advance strategy is bounded to the latency in triggering. An example of a species using this scheme is the *Pteroptyx effulgens*.

The fireflies are not the only miracle of nature which are able to mutually synchronize. Other similar oscillatory phenomena can be observed on a few kinds of crickets that chirp in synchrony or in some instance of human behaviour such as the spontaneously synchronized clapping in concert crowds. Moreover, it synchronization does not necessarily mean that the fireflies or other biological oscillators fire at the same time. Therefore, three different levels of synchronization were introduced and are described below.

Synchrony. Synchrony means that the oscillators fire in unison. In real populations, synchrony usually never occurs, because the oscillators often deviate from a nominal frequency. However, if the precision of the firing times are small compared to the synchronization interval, then a population of pulse-coupled oscillators can be seen to be in synchrony. For instance, synchrony appears in the firing of peacemaker cells, synchronous flashing of fireflies, or chorusing of crickets. According to [Pes75], synchrony is a cooperative effect between dissipation and coupling.

Phase locking. Phase locking occurs, if the phase difference between several oscillators is non-zero and constant. Therefore, they may not fire in unison. This weaker definition of synchronization can be found in some oscillator populations based on randomly distributed natural frequencies.

Frequency locking. Frequency locking is defined as a weak synchronization where the oscillators have the same average frequency but a varying phase difference to each other.

2.4.1 Mirollo and Strogatz (MaS) Model

In [MS90], Mirollo and Strogatz have described the synchronization behaviour of fireflies as a model of PCO. The model is a generalization of the *leaky integrate-and-fire* model of the cardiac pacemaker model from Peskin stated in [Pes75] and is based on the following assumptions:

1. The oscillators have identical dynamics (e.g. same period).
2. Each oscillator is coupled to all the others.

The MaS model is similar to the phase-advance synchronization model from Buck described in 2.4 but with the difference that MaS assume a concave-down excitation towards the threshold in contrast to Bucks model of a linear excitation. In detail, the model assumes that every oscillator is characterized by a state variable $x = f(\phi(t))$ which increases according to a phase variable $\phi(t)$ toward a threshold at $x_{th} = 1$. In the case x equals x_{th}, the oscillator fires and resets the state to $x = 0$. The phase ϕ is a variable which determines the course of time within an oscillator interval. Therefore, ϕ is a proportional function of the real-time t, but persists in the range from 0 to 1. For example, in the case of a cycle period with a duration of $T = 10s$, then $\phi(t)$ maps to $\phi(t) = \frac{(t \mod T)}{T}$ and has no dimension. Consequently, the function $f : [0, 1] -> [0, 1]$ is a one-to-one mapping from the phase to the state of the oscillator with the same value domain, but with the important characteristics that f is smooth, monotonic increasing, and concave down. Based on this conditions, MaS have calculated a general function $f(\phi)$ whereas the form of the curve depends on a parameter named dissipation factor, denoted by b:

$$f(\phi) = \frac{1}{b} \cdot ln(1 + [e^b - 1] \cdot \phi) \text{ with } b > 0$$

Figure 2.1 shows the function f for different dissipation parameters b. Note that $b = 0$ corresponds to the identity map for which a synchronization will not be achieved.

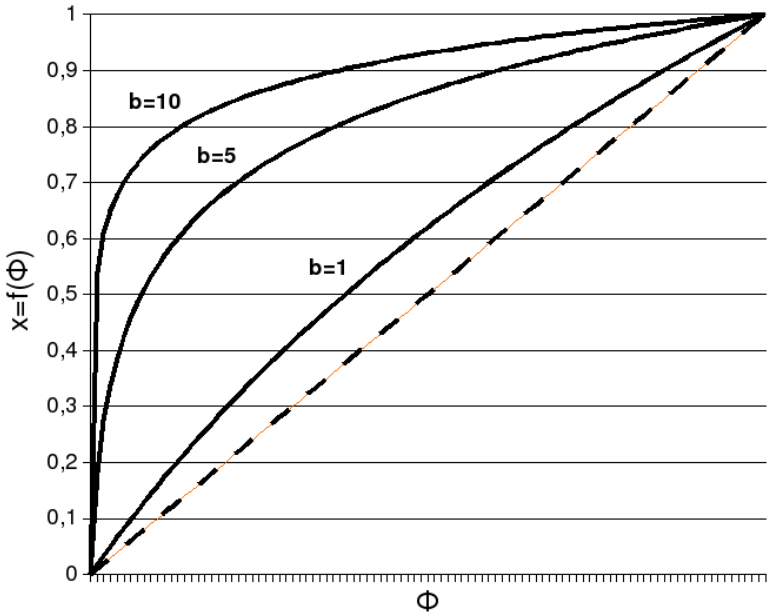


Figure 2.1: The state function dependent on different dissipation factors.

To satisfy the second assumption of coupling, every oscillator receiving a firing event has to advance its state by a constant value. This constant is named pulse strength and denoted by ε. Because of the concave down form of the function, the constant addition in the state domain results in a variable phase increment. Therefore, the coupling strength between the oscillators is only determined by the dissipation factor b and the pulse strength ε. The new phase is calculated through the less of 1 and the result of the firing function $g(\varepsilon + f(\phi))$ with $g = f^{-1}$. That is

$$\phi_{new}(\phi) = \min(1, g(\varepsilon + f(\phi)))$$

$$\text{with } 0 < \varepsilon < 1 \text{ and } g(u) = \frac{e^{bu} - 1}{e^b - 1}.$$

This ensures that an oscillator reacts more strongly to an event that occurs later in its phase. It should be noted that the state function f is not linear. Nevertheless, the Phase Response Curve (PRC) denoted by $\phi_{new}(\phi)$ is against one's expectations linear. Mirollo and Strogatz have also proved that the time until the network achieves synchronicity is proportional to the product $b \cdot \varepsilon$ and also depends on the initial phase of the oscillators. An example of such a phase shift is demonstrated in Figure 2.2. This graph shows the state of two oscillators ϕ_A and ϕ_B with a typical state function just before oscillator B has fired. $\phi_{A_{new}}$ denotes the state after the phase shift caused by the firing of B.

On a final note, the major result from MaS is that under the assumptions mentioned above, a set of oscillators with any initial phase will always achieve synchronicity. Additionally, Lucarelli and Wang proved in [LW04], that this model still achieves synchronicity when the communication topology is time varying. Consequently, assumption two can be neglected. This is of particular importance for sensor networks which are often based on multi-hop topologies. However, many prior reports have stated that the synchronization error increases with every hop. For instance, the authors from [EGE02] show that the synchronization error in a multi-hop network based on a Reference-Broadcast Synchronization (RBS) linearly increases due to the number of hops. The same applies on such a network topology based on the MaS synchronization model as presented in this work. Though, the results from [HS06] show that there still exists a solution which reduces the synchronization error over several hops. This improvement is based on spatial averaging. Therefore, they assume that each node has many neighbors and hence receives more information for synchronization. Based on the law of large numbers, the authors prove that the superposition of N received pulses ($N \to \infty$) eliminates the random errors in transmission and thus improves scalability.

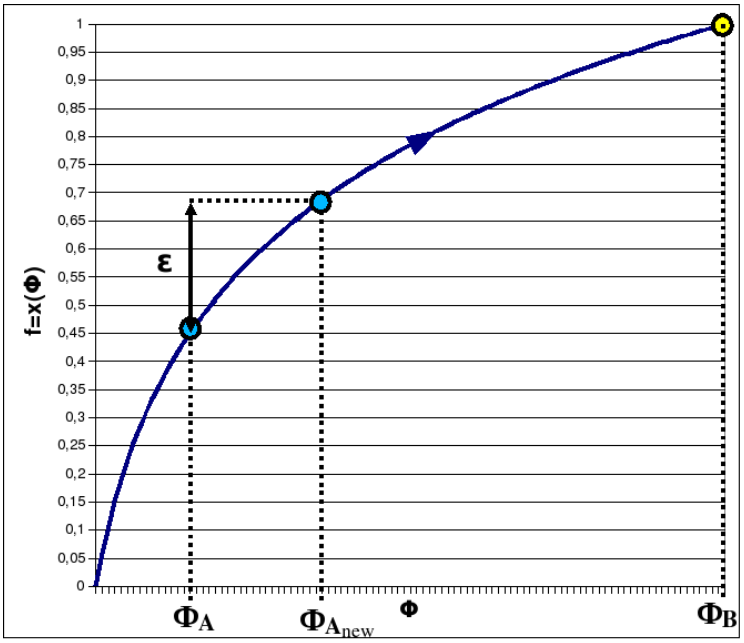


Figure 2.2: The graph shows a phase shift caused by a firing event.

2.4.2 The RFA Model

The RFA was introduced in [WATP+05] and is based on the MaS model, but with the difference that it is more appropriate for the implementation in wireless networks. Some assumptions resulting from the theoretical MaS model are listed below and makes it difficult for a practical application.

1. The oscillators have identical dynamics (e.g. same period).
2. Nodes can instantaneously fire.
3. Every firing event must be observed immediately (no loss).
4. All computations are performed perfectly and instantaneously.

The most problematic one is assumption three, because in reality the firing is implemented as a broadcast message which usually has an unpredictable delay mainly caused by the channel contention prior to the transmission. The other assumptions are not so problematic. For example, the nodes are based on oscillators which generally have a small but measureable drift. Next, the limits in floating point arithmetics result in inexact computations and finally the loss of messages in wireless networks due to influence problems or the varying link quality can not be avoided.

For this reason, the RFA model controls the unpredictable message delay via *MAC-layer timestamping*, so that the receiver has knowledge about the MAC

delay and consequently is able to determine the correct firing time. In addition to that, the RFA model is based on an explicitly added random transmission delay between 0 and a constant D at the application level, called *message staggering delay*. The message staggering delay should avoid message collisions, in the case if many synchronized nodes want to send simultaneously.

In contrast to the MaS model, the RFA model does not immediately react on an observed firing event. Instead it stores the corrected timestamp of all received firing events in a queue. Then, if the phase of a node reaches the end, it computes the new start phase based on the content of the queue. The computation is the same as in the MaS model. As a result, a node seems to react instantaneously, but to the data from the last period.

Computation of the start phase. The computation of the new phase is performed at the end of each period and is specified in Algorithm 1. The algorithm uses the function described in 2.4.1 to get the overall jump which corresponds to the start phase for the next period. Therefore, the instantaneous jump $\Delta(\phi)$ for each timestamp stored in the queue is calculated whereas $\Delta(\phi)$ declares the difference between the old and the new phase:

$$\Delta(\phi) = \min(1, g(\varepsilon + f(\phi))) - \phi$$

Further, all calculated phase jumps must cumulatively be added to all the other timestamps stored in the queue. This ensures that the overall phase jump corresponds to an instantaneously phase shift.

Algorithm 1 Calculation of the start phase.

1: $overall\_phasejump = 0$
2: **while** queue not empty **do**
3: $tStamp$ = next element from queue {get next timestamp}
4: $phasejump = \Delta(tStamp + overall\_phasejump)$
5: $overall\_phasejump = overall\_phasejump + phasejump$
6: **end while**
7: set current phase to $overall\_phasejump$

2.5 The Time-Triggered Protocol (TTP)

Many communication systems are usually based on an event-triggered approach. However, this concept is not appropriate for real-time systems, especially if fault tolerance is inevitable. For instance, event-triggered systems are not aimed at coping with the demands for hard real-time systems like

composability, predictability, guaranteed timeliness, and determinism. For this reason, the TTP [Kop97] concept has been designed at the Technische Universit"at Wien and is explained because of its use as an evaluation protocol in this work. The TTP is a protocol used for distributed real-time systems in safety critical applications whereas two variants for different requirements were developed. First the low-cost TTP/A [EHK+05] version for field-bus applications and secondly the more sophisticated TTP/C derivative for fault tolerant intra-cluster communication systems. The following sections mainly focus on the TTP/A architecture, but the mechanisms described here are common for both derivatives of TTP.

2.5.1 Overview of TTP/A

As described above, TTP/A is a time-triggered protocol for field-bus applications, whereas the purpose is to interconnect low-cost *smart transducer* nodes providing guaranteed timeliness for hard real-time communication. This type of TTP is based on the Master/Slave principle. Therefore, a TTP cluster must be made up of at least one master node and several slave nodes, whereas the other masters are shadow masters which take-over if the active master fails. Additionally, every master can also serve as a gateway node.

A smart transducer is a notion for a single electronic unit, which is composed of a sensor/actuator and a low-cost microcontroller, that contains the hardware (e.g. processor, memory, network controller) and software for the communication interface. Because such transducers are widespread and often have different vendors and capabilities, the communication interface must be generic for a better composability in networks. However, in conjunction with TTP/A, a transducer must support some standard functionalities to transmit data in a temporally deterministic manner and in a standard data format. The basic services of the TTP/A protocol are standardized by the OMG Smart Transducer Standard.

2.5.2 Principles of Operation

TTP/A relies on the TDMA scheme to achieve a predictable time behaviour in a cluster comprised of several nodes which are controlled by an active master. Therefore, communication is organized in rounds which are independent from each other. Each round is divided into a number of slots which are statically assigned to communication activities of the different nodes. Such an activity can be the sending of a message, the receiving of a message or task invocation. The slot associations are different for each node and are further described by

an entry in the Round Description List (RODL) in the file-system of the sender and the receiver(s).

2.5.3 The Interface File System (IFS)

The IFS provides a basis for storing and accessing local data of a node. Further, it serves as a source respectively sink for communication data and as an interface to the application. Every TTP/A node has some special files listed below, but can have further files with different access rights for the application.

The Documentation File The documentation file is the only file which must implemented in every TTP/A node and is set read only. It contains the unique physical name of the node (e.g. MAC address) and therefore serves as an identifier.

The Configuration File The configuration file contains the cluster name and the logical name of a node which is assigned by the TTP/A protocol. Further, it stores the current status which can be: Not synchronized (startup), passive (no send operation), not baptized, passive lock (no send operation), active, and error. This is also the file location that holds the current round number, the epoch counter, and the slot counter.

The Membership File This file is only needed in master or gateway nodes. It contains two membership vectors, whereas the index of each bit of the vectors represents the logical name of a node. The first vector contains all slaves which have sent a live-sign and the second one contains all slaves which have responded to the most recent master-slave operation.

The RODL file The RODL file is the most important file in the TTP/A protocol. It defines the actions which have to be performed in the particular round. There exist three different types of TTP/A rounds described below. Such a file contains several entries which define the type of operation for a corresponding slot. Such operation types can be read, write, and execute. For read and write operation, the entry also contains the location to the communication data in the IFS. In case of an execute operation, the entry must contain the parameters to the task descriptor.

Multi-Partner Rounds This is the standard round type in TTP/A and is used to exchange data between several nodes. The multi-partner rounds are statically defined by the user and depend on the application. They are periodically scheduled by the master.

Master-Slave Rounds The master-slave round may be used for debugging to read data from an IFS file, write data to an IFS file, or execute a task stored in an IFS file on a slave. Thus, a master-slave round is distinguished in a master-slave address (MSA) and master-slave data (MSD) round. The MSA round is therefore initiated by the master and specifies the destination node, the operation and the location of the communication data within the addressed node. Next, the MSD round can be regarded as a slave response to a MSA request. It contains the communication data which is transmitted between the master and the slave.

Broadcast Rounds A broadcast round is a master-slave round but with the difference that the name of the addressed node is a special broadcast address. Consequently, this round is received by all nodes and therefore can only contain a write or an execute operation. A typical execute operation is the *sleep* command which puts the complete TTP/A cluster into a *sleep mode*.

The Round Sequence (ROSE) file Because a TTP/A node can have different rounds, there must be a mechanism which defines the execution order for them. For this reason, the ROSE file is reserved to the master and specifies the sequence in which the rounds are scheduled.

2.6 The ZigBee Protocol

ZigBee [All06] is a standard for Low-rate Wireless Personal Area Network (LR-WPAN)s and has been ratified in late 2004 under IEEE 802.15.4 Wireless Networking Standard. The reason for this development was, that other wireless standards were not suitable for building automation and industrial control, because they needed lower cost, better latency, and lower power consumption [EP03]. The ZigBee protocol is similar to Bluetooth butm simpler, has a lower data rate and a very low power consumption. Thus, these characteristics make it possible that a node on a ZigBee network should be able to run several months to two years on just two AA batteries and therefore is best suited for embedded applications in consumer electronics, home and building automation, industrial control, sensor networks, and many others.

2.6.1 ZigBee Architecture

The ZigBee architecture is based on the standard Open System Interconnection (OSI) seven-layer model, but defines only the relevant layers. Therefore, ZigBee specifies the application and Network (NWK) layer and builds on the Physical (PHY) layer containing the radio frequency (RF) transceiver and the MAC layer from the IEEE 802.15.4-2003 [IEE03] standard. The purpose is, that devices from different vendors based on this standard will enable interoperable, low-cost, and highly usable products. The application layer is comprised of the Application Support (APS) sub-layer, the Application Framework (AF) profile, the ZigBee Device Object (ZDO)s, and the manufacturer-defined application objects. A simplified relation between these layers is depicted in the Figure 2.3.

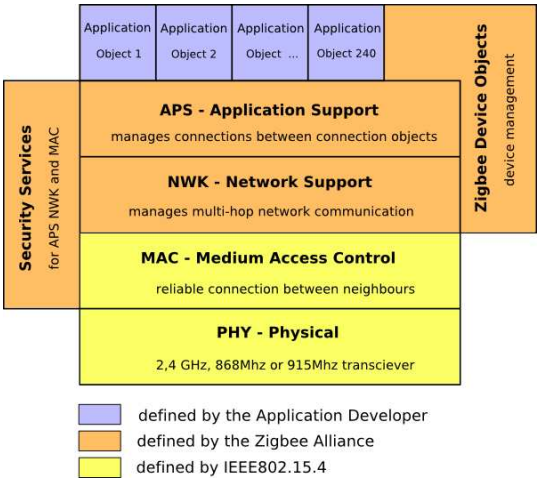


Figure 2.3: ZigBee architecture.

2.6.2 ZigBee Components

A ZigBee System is composed of a number of components which have different features. Because of different constraints regarding implementation costs, memory requirements, and other hardware resources, the ZigBee *Physical Device* type based on IEEE 802.15.4 is distinguished in three *Logical Device* types.

Generally, IEEE 802.15.4 provides the implementation of two different *Physical Device* types. The first one is the Full-function Device (FFD) and the second type is the Reduced-function Device (RFD). Every IEEE 802.15.4 network requires at least one FFD, which acts as a network coordinator. As listed in the Table 2.1 below, an RFD is implemented with minimum resources and therefore improves power consumption and reduces implementation size and costs [Kin03]. In contrast to FFDs which can also serve as a link coordinator

and is generally line powered, an RFD component is usually battery-powered and can only talk to an FFD.

Reduced Function Device	Full Function Device
Limited to star topology	Can function in any topology
Can only talk to an FFD	Can talk to RFDs and FFDs
Can not become a network coordinator	Can become a network coordinator
Very simple implementation	Can become a coordinator.

Table 2.1: ZigBee physical device types.

As mentioned above, ZigBee distinguishes the *Physical Device* types (RFD or FFD) into three different *Logical Device* types, which are described in the following paragraphs:

ZigBee Coordinator (ZC). Every network must contain exactly one ZC which is responsible for starting the network and therefore for initiating and maintaining the devices. The coordinator selects the frequency to be used by the network and associates the devices with a logical address. It also provides message routing, security management, and other services. Only an FFD can be a Personal Area Network (PAN) coordinator.

ZigBee Router (ZR). ZigBee Router (ZR)s are used for moving data and control messages through the network using a hierarchical routing strategy and thus can act as an intermediate router. They are used to extend a network or to combine several ZigBee clusters. Only an FFD can be a ZR.

ZigBee End Device (ZED). The end devices directly communicate with a ZigBee coordinator and are used for applications which do not need to send large amounts of data. They have a reduced functionality and therefore minimizes memory requirements and simplifies the implementation. Both FFDs and RFDs can act as a ZED.

2.6.3 Network Topologies

ZigBees's network layer supports three network topologies which are pictured in the Figure 2.4.

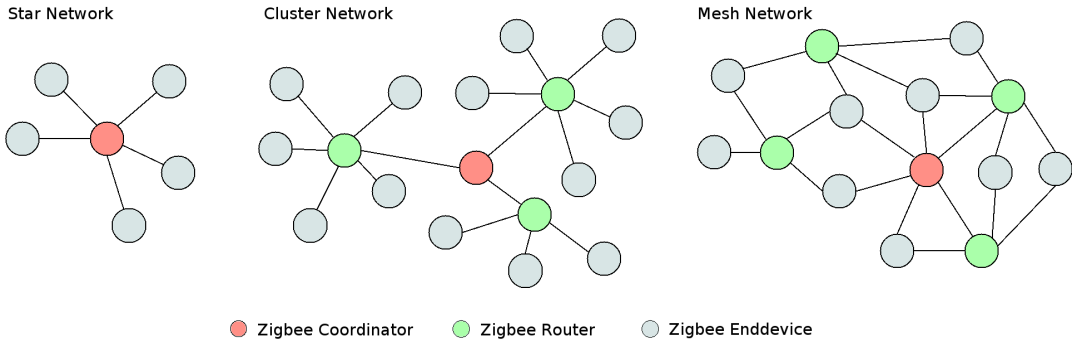


Figure 2.4: Topology models.

Star Topology. This network topology contains exactly one ZigBee coordinator, which establishes the connection between several ZEDs. In contrast to the coordinator, the end devices are usually battery-powered. Hence, this topology may be used in applications with constraints on power consumption such as home automation, toys and games, or personal computer peripherals. During each start of the coordinator and therefore of a network, the coordinator scans for other networks and then chooses a PAN identifier which is not currently used by others. Thus, although several star topologies may be in the radio range of each other, they are able to operate independently.

Mesh Topology. The mesh topology, also known as peer-to-peer topology, improves reliability and scalability by multipath routing and therefore can be ad-hoc, self-organizing, and self-healing. It contains again a single ZigBee coordinator and typically allows full peer-to-peer communication. Moreover, messages can also be routed through multiple hops which makes this topology appropriate for wireless sensor networks. Other applications that benefit from this may be industrial control and monitoring, or asset and inventory tracking. In such a topology, the ZigBee coordinator is responsible for starting the network. The network extension is therefore done through the use of ZRs and ZEDs.

Cluster Tree Topology A cluster tree topology usually consists a number of FFDs which build up a tree topology wheres a single ZigBee coordinator forms the root of this tree. Moreover, every FFD may be connected together with several ZRs or ZEDs and thus forms a cluster. In other words, the end devices correspond to the leaves of the cluster tree. In addition to that, every router can provide synchronization service to the neighboring devices.

Because the ZigBee coordinator forms the first cluster, it is also named Cluster Head (CLH) and provides a Cluster Identifier (CID), an unused PAN iden-

tifier, and the broadcasting of beacon frames. A CLH may add neighboring devices and can instruct a ZigBee router to become the CLH of a new cluster. This enables an increased coverage area at the cost of communication latency.

2.6.4 IEEE 802.15.4 PHY

According to [GNC+01, CGH+02], the physical layer of IEEE 802.15.4 offers two PHY options: The *2.4 GHz PHY* and the *868/915 MHz PHY*. Both are based on direct sequence spread spectrum (DSSS) methods and share the same packet structure. This allows a simpler and cheaper IC implementation. The 2.4 GHz PHY operates in the 2.4 GHz ISM band and therefore offers nearly worldwide availability. However, many other standards have also chosen this band and thus may be affected by the interaction of incompatible services operating in the same band. Because many LR-WPAN applications are not based on mobility, the IEEE 802.15.4 task group has also added a second PHY option which is a combination of the 868 MHz band in Europe, the 902 MHz band in Australia, and the 915 MHz band in the United States. The close frequency relation of these bands makes the use of similar hardware possible and thus lowers the production costs. Other advantages of the second PHY option are lower power consumption and the avoidance of the interference problem in the 2.4 GHz ISM band.

Compared to the 868/915 MHz PHY, the 2.4 GHz PHY is based on a higher-order modulation scheme, in which each data symbol represents several bits. For this reason, the 2.4 GHz PHY has a higher transmission rate of 250 kbps, while 868/915 MHz PHY offers 20 kbps for the 868 MHz band and 40 kbps the 915 MHz band. In comparison, every PHY option has its strengths and are stated in the Table 2.2.

868/915 MHz PHY	**2.4 GHz PHY**
better sensitivity	higher throughput
larger coverage area	lower latency
fewer interference problems	lower duty cycle

Table 2.2: Strengths of both PHY options.

The previous described three bands are subdivided into twenty-seven frequency channels. Thus the 868/915 MHz PHY offers a single channel between 868.0 and 868.6 MHz, and 10 channels between 902.0 and 928.0 MHz whereas the 2.4 GHz PHY offers 16 channels between 2.4 and 2.4835 GHz. Because of the interference problems with other applications based on the 2.4 GHz band, the standard also supports dynamic channel selection. However, the specific

channel selection is reserved to the network layer. The characteristics of the two PHY options are summarized in Table 2.3.

PHY	Channel	Frequency band	Data parameters		
			Bit rate (kbps)	Symbol rate (kbaud)	Modulation
868 MHz	0	868-868.6 MHz	20	20	BPSK
915 MHz	1-11	902-928 MHz	40	40	BPSK
2.4 GHz	12-27	2.4-2.4835 GHz	250	62.5	O-QPSK

Table 2.3: Frequency bands and data rates.

Packet structure. As mentioned above, both PHYs share the same packet structure as depicted in Figure 2.5. A packet at the physical layer is named Physical Protocol Data Unit (PPDU) and contains a preamble, a start of packet delimiter, a PHY header and the PHY Service Data Init (PSDU). The preamble together with the delimiter forms a synchronization header and the PHY header indicates the packet length. Because only seven bits are used to specify the payload length in bytes, a PSDU can have a maximum length of 127 bytes. Therefore, the maximum possible packet duration are 53.2 ms for the 868 MHz band, 26.6 ms for the 915 MHz band, and 4.25 ms for the 2.4 GHz band.

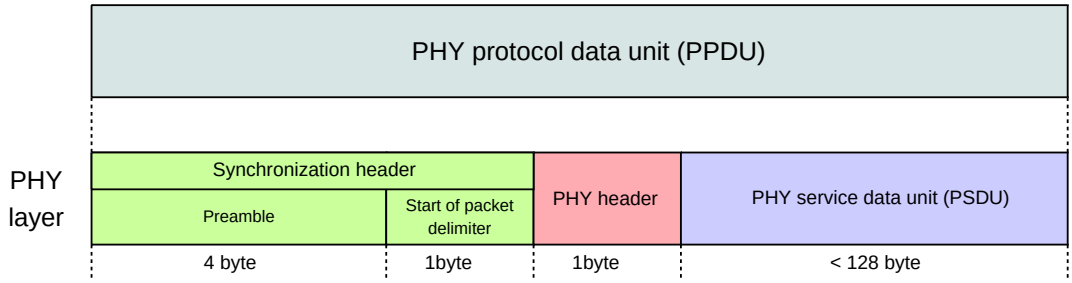


Figure 2.5: IEEE 802.15.4 PHY packet structure.

Sensitivity and range. In order to support low-cost implementation, the IEEE 802.15.4 standard specifies a receiver sensitivity of −85 dBm for 2.4 GHz and −92 dBm for 868/915 MHz PHY. However, the achievable range does not only depend on the receiver's sensitivity, but also on the sender's transmitting power. Therefore, the standard also specifies to transmit at least 1 mW. Another important aspect is the control of the transmitting power, because this could improve reliability and power consumption. For instance, the quality of the transmission is continuously observed and the transmitter power is reduced

as far as possible, so that the receiver can still hear the sender. Thus, this reduces the problem with interfering networks operating in the same frequency channel. Furthermore, a transmitting power of 1 mW allows a radio range of about 10 to 20 metres but can be up to 100 metres.

2.6.5 IEEE 802.15.4 MAC

According to [IEE90], the IEEE 802 group has split the data link layer into two sublayers, the MAC layer and the Logical Link Control (LLC) layer. The data link layer usually provides functionalities to transfer data between several network entities. Furthermore, it is responsible for error detection and correction that may occur in the physical layer. The LLC is standardized in [IEE98] and usually provides flow control, acknowledgement, and error recovery. Compared to LLC, the MAC layer is closer to the hardware and determines who is allowed to access the medium. As shown in [IEE03], the IEEE 802.15.4 MAC sublayer is responsible for the following tasks:

- Association and disassociation
- Acknowledged frame delivery
- Channel access mechanisms
- Frame validation
- Guaranteed time slot management
- Beacon management
- Security management

The MAC sublayer provides two services to the upper layers, which can be accessed through two Service Access Point (SAP)s: The *MAC data service* and the *MAC management service*. The MAC data service is responsible for the reception and transmission of MAC Protocol Data Units (MPDU)s and the MAC management service provides several service primitives to handle the above mentioned tasks.

The general MAC frame format. The MAC frame is named MPDU and is enclosed in the PSDU. An MPDU frame has a maximum packet size of 127 bytes and is composed of the MAC Header (MHR), the MAC Service Data Unit (MSDU), and the MAC Footer (MFR). Regarding to the Figure 2.6, the MHR contains a frame control field, a sequence number field, and an address information field. The control field indicates the type of MAC frame (e.g. beacon frame, data frame, acknowledgment frame, MAC command frame) and further specifies the format of the address field, i.e., 16 bit short address or 64

bit extended address. Therefore, the size of the address field may vary. The sequence number is used for the acknowledgment to provide a reliable link and the Frame Check Sequence (FCS) field helps verify the integrity of the MAC frame via 16-bit ITU-T Cyclic Redundancy Check (CRC). Consequently, 102 bytes are left for the MSDU and can be used for user data in the upper layers.

The 64 bit extended address is long enough to assign every device in the world a unique number. But if the communication is only based on this address type, then this will result in much overhead and is not beneficial. Because a PAN comprises by far fewer devices than can be addressed via 64 bit, the short address type was introduced and enables a maximum amount of 65536 participants. The short address will be assigned from the PAN coordinator through the association process.

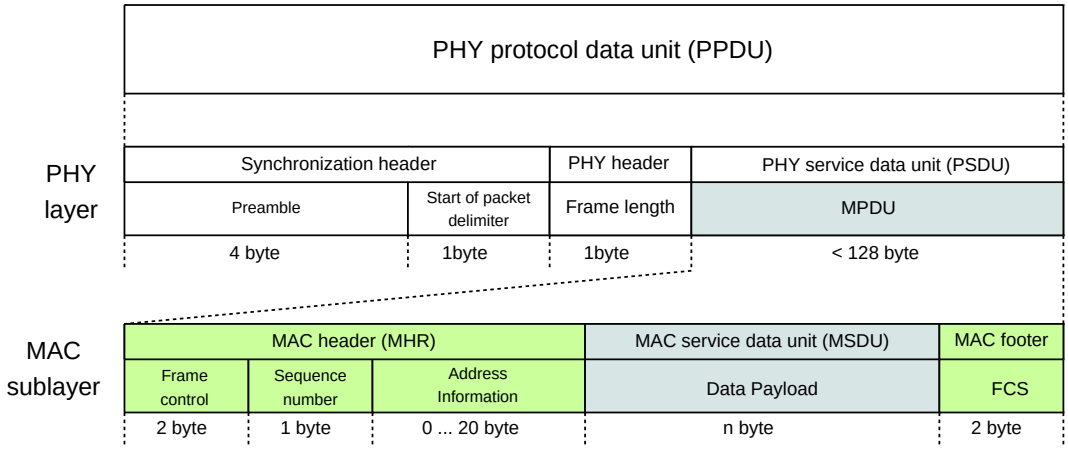


Figure 2.6: The IEEE 802.15.4 MAC frame format.

Channel access mechanisms. The MAC standard uses two types of channel access mechanisms which depend on the use of beacons. Both are based on the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) medium access mechanism and therefore are contention based. The only difference is that the first mechanism uses an unslotted variant and the second one a slotted variant of CSMA/CA. Networks with beacons use the slotted CSMA/CA mechanism and are usually deployed in low-latency devices, but a device can also choose not to use beacons for normal transfer and thus use the unslotted variant.

Superframe structure. To take advantage of the beacon scheme, IEEE 802.15.4 has the option to use a superframe structure which allows the PAN coordinator to allocate time-slots to devices with time critical data. The superframe structure is bounded by superframe beacons which are transmitted

in predetermined intervals by the coordinator. These intervals are divided into 16 equally sized slots, while the first slot of each superframe contains a special beacon frame. This beacon frame describes the structure of the superframe. The interval between two superframe beacons can be 15 ms up to 245 s. A superframe can also have an active and an inactive portion which can be used to save energy. In the active period each device can transmit at any time during a slot, but must finish before the next superframe beacon. The channel access mechanism in each slot is based on the slotted CSMA/CA strategy. For this reason the active period is also called Contention Access Period (CAP).

However, some applications may require guaranteed bandwidth for dedicated devices. For this reason, the standard also supports the use of Guaranteed Time Slot (GTS)s at the end of each active superframe. This results in a contention-free channel access mechanism. The portions used for GTSs together form a Contention Free Period (CFP). This mechanism is visualized in Figure 2.7. It should be noted that a contention based transmission should end before the start of a CFP.

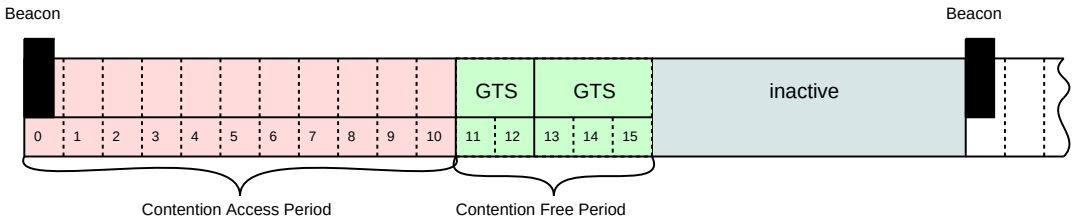


Figure 2.7: Superframe structure with GTSs.

CSMA/CA. In networks which does not use beacons, IEEE 802.15.4 determines the use of a contention-based channel access mechanism named CSMA/CA. The behaviour of the CSMA/CA algorithm is generally described with three parameters: The Number of Backoff Periods (NB), the Contention Window (CW) length, and the Backoff Exponent (BE).

NB is initialized to 0 before a new transmission attempts and is incremented each time a backoff was required while attempting this transmission.

The CW length is defined in number of backoff periods. A transmission will only be started if the channel is clear during the contention window. The CW is only used in the slotted variant of the CSMA/CA algorithm.

BE is the backoff exponent and defines the number of backoff periods a device has to wait before a channel assessment can be started. The MAC layer features different variables for the implemented CSMA/CA mechanism, i.e., the Minimum Backoff Exponent (macMinBE) and the Maximum Backoff

Exponent (macMaxCSMABackoffs). macMinBE defines the minimum number of backoff exponents and *macMaxCSMABackoffs* declares the maximum value of the backoff exponent before the algorithm will terminate with a channel access failure. Furthermore if macMinBE is set to 0, no collision avoidance will be performed during the first iteration of the algorithm. Although the receiver is enabled during the channel assessment, the device discards any received frame.

If a frame should be sent, the algorithm first waits for a random time determined by a random number of backoff periods from 0 to $2^{BE} - 1$ and then performs a Clear Channel Assessment (CCA). In the case the channel is free, the transmission will be started. Otherwise if the channel is assessed to be busy, both NB and BE are incremented by one. If the value of NB is less than or equal macMaxCSMABackoffs, then the algorithm waits again for a random time. Otherwise, if NB is greater than macMaxCSMABackoffs, then the algorithm terminates with a channel access failure.

Data transfer model. The MAC sublayer from IEEE 802.15.4 distinguishes three types of data transfer: The data transfer to a coordinator, the data transfer from a coordinator, and the peer-to-peer data transfer. In contrast to a star topology, a peer-to-peer topology may use all three transactions whereas a star topology will not make use of the peer-to-per data transfer model. It should be noted that the transfer mechanisms also depend on the use of beacons.

Data transfer to a coordinator. This scheme is used to transfer data from a device to the coordinator. In case of a network with the use of beacons, the device must first wait for a network beacon. If so, the device transmits its data frame using the slotted version of CSMA/CA. Afterwards, if the coordinator successfully received the frame, it sends an optional acknowledgment. The mechanism for a network without the use of beacons is similar, but with the addition that the device uses slotted CSMA/CA and it does not have to listen for a beacon before transmitting the data. Both principles are shown in Figure 2.8.

Data transfer from a coordinator. If a device requests data from the coordinator, then this mechanism will be applied. In a network using beacons, the coordinator must first indicate in a beacon that it is ready to send data to a device. Therefore, the device listens to the medium and if it receives such a beacon, then the device responds with a data request packet. Afterwards, the coordinator is allowed to send the pending message. In addition to that, all transactions are acknowledged. By contrast, a network without beacons uses

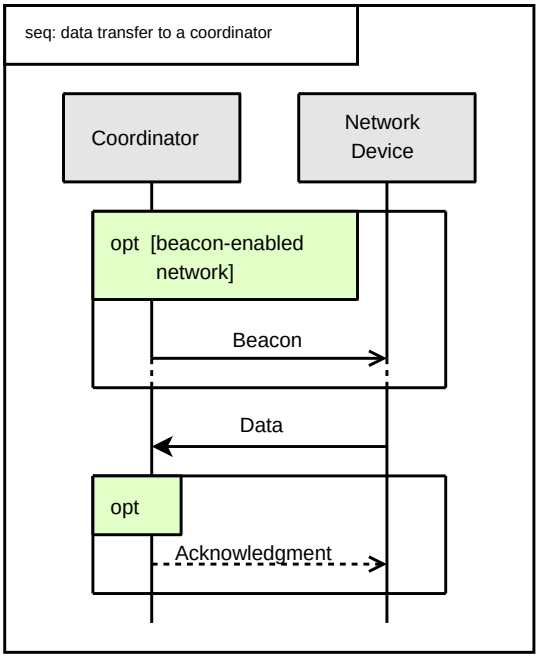


Figure 2.8: Communication to a coordinator.

unslotted CSMA/CA and the coordinator stores all messages and continuously waits for a data request from a device without any beacon messages. The transfer model for both variants is depicted in Figure 2.9.

Peer-to-peer data transfer. This model is used if a device in a network wants to communicate with each other and can be either performed by synchronized or an unsynchronized scheme. If no synchronization is performed, then the devices continously receive messages based on unslotted CSMA/CA. The synchronized scheme is not provided in the standard and may be implemented by the upper layers.

Robustness. The LR-WPAN standard supports some mechanisms to ensure robustness in communication. Theses mechanisms are the CSMA/CA scheme, frame acknowledgment, and data verification via a CRC strategy.

Security. The security mechanism in this standard includes the ability to maintain an Access Control List (ACL) and the use of symmetric cryptography to protect transmitted frames. The ability to use these functions are determined at the upper layers.

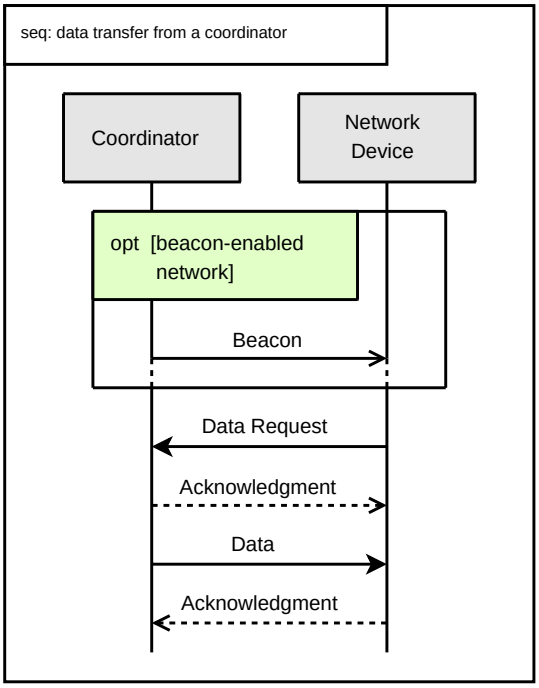


Figure 2.9: Communication from a coordinator.

3 Related Work

3.1 Spontaneous Synchronization in Multi-hop Embedded Sensor Networks

The work from A. Bletsas and A. Lippman [BL05] demonstrates a Firefly inspired distributed time synchronization technique for multi-hop sensor networks using nearest neighbor communication. The goal was to synchronize speakers, so that they play the same music at the same time. The nodes were programmed to have a period of about 13 seconds and a clock resolution of 5.9 ms. They measured the synchronization error as a function of network diameter and communication overhead. The synchronization error $err(t) = C_i(t) - C_j(t)$ was defined to be the difference between the local time of the edge nodes at the same instant. Some of the results are denoted in Table 3.1. It is interesting to know that the measurements show an average absolute synchronization error $|err(t)|$ in the order of a few milliseconds.

Network diameter (maximum number of hops)	**Average absolute synchronization error**
1	$\backsim 4\ ms$
2	$\backsim 10\ ms$
3	$\backsim 4\ ms$
4	$\backsim 2\ ms$
5	$\backsim 6\ ms$

Table 3.1: Synchronization error vs. network diameter.

The seminal result of this work is that in a real-world implementation, the absolute synchronization error does not necessarily depend on the network diameter, as it was quoted in many prior reports which show that in a multi-hop network the synchronization error linearly increases with the number of nodes. But all these reports are based on simulations whereas the work of Bletsas and Lippman presents the first implementation in a real-world embedded network. They have studdied this phenomenon and came to the conclusion that the synchronization error does not increase lineary with the network diameter.

Instead, the error mainly depends on the frequency skew[1] distribution of the node oscillators. Consequently, if the clock drift with respect to real-time at the receiver node is lesser than the drift of the sender, then the error due to the sender's clock drift, the dominant transmission time, and the processing time might decrease. Otherwise, if the frequency skew between the sender and receiver is positive, the error may increase.

3.2 A Scalable Synchronization Protocol

In [HS05], the authors present the emulation of the PCO model from Mirollo and Strogatz based on the narrow pulse characteristics of Ultra Wideband (UWB) systems. The experiments in the work of Scaglione show that in an all-to-all topology the number of necessary periods to achieve synchronicity decreases with an increasing number of nodes and/or a stronger coupling ε. Further, they observed that in an ensemble of N nodes, there exist a phase transition at the point $\varepsilon \cdot N = 1$, so that synchrony is achieved immediately. However, this is not true for a multi-hop scenario where the number of necessary periods for synchronization is approximately $O(log(N))$. Further, they described the relationship between energy efficiency and the time to synchrony. Additionally, they assumed that if a node fires, it cannot receive any other firing pulses for a short time. They stated that this avoids an infinite excitation between close-by nodes and that this phenomenon can also be observed in nature. For instance, this short duration also exists in spiking neurons and is called *refractory period*. In detail, the effect of the infinite excitation is caused by the propagation delay between the oscillators. For example, consider a network with an all-to-all coupling comprises a number of N nodes which are initially synchronized. If the coupling strength ε is chosen to be relatively large, then because of the propagation delay, a transmitted pulse will be received from the other nodes a short time after the end of the cycle period. Because of the large coupling strength and a big N, the sum of all these phase increments causes the receiving nodes to reach the threshold immediately. This results in a transmission and thereby causes the other nodes again to fire immediately. The authors have called this never-ending mutual excitation *avalanche effect*. The refractory period eliminates this problem, but on the other hand bounds the accuracy. Additionally, they found out that the precision of the MaS model is independent from the network topology, i.e., the synchronization error does not increase according to a larger multi-hop network. This comes from the fact that the synchronization error is dominated by the propagation delay and not by the network diameter.

[1] The frequency skew of a clock is defined to be the drift with respect to real-time minus 1.

3.3 Firefly Synchronization in Ad-hoc Networks

According to [TAB06], the authors introduce a time advance strategy based on the PCO model, which takes the delays in wireless systems into account. Furthermore, they incorporate the fact that a node cannot transmit and receive at the same time. The authors declare four important delays: The propagation delay T_0, the transmitting delay T_{tx}, the decoding delay T_{dec}, and the refractory delay T_{refr}. The propagation delay is the time which is needed for a message to propagate from the sender to the receiver and is proportional to the distance between both. Usually, this delay is in the range of several μs and therefore negligible. The transmitting delay is determined by the length of the message, whereas the decoding delay indicates the duration between the reception and the reaction to a synchronization message at the receiver. Further, the refractory delay defines a short period after a transmission. During the refractory delay a node ignores all received synchronization messages and ensures stability. Therefore, the total delay T_{del} depends on the dominant transmission and decoding delay. It should be noted that if the synchronization strategy is based on the PCO model, then the accuracy would be lower bounded to this total delay.

On the other hand, the time advance strategy presented in this paper compensates this total delay by delaying the transmission of the synchronization message. That is

$$T_{wait} = L \cdot T - T_{del} = L \cdot T - (T_{tx} + T_{dec})$$

where T denotes the reference synchronization period and $L \geq 1$ declares the number of delayed periods for the phase advance. Consequently, if an already synchronized node receives a synchronization message, it will be able to increment the phase after L periods, i.e., exactly $L \cdot T$ seconds after receiving and decoding the message. Therefore, this eliminates the unavoidable transmission and decoding delay and results in a natural oscillatory period with a duration of $L \cdot T$ seconds. In addition to that, phase increments due to received synchronization bursts are only possible during the receive interval which corresponds to

$$T_{rx} = T + (T_{refr} - T_{dec}).$$

The strategy presented here assumes that the transmission delay and the decoding delay are well known and constant. If so, the accuracy will be lower bounded to the propagation delay. A typical phase diagram with $L = 1$ and two oscillators (ϕ_1,ϕ_2) is shown in Figure 3.1.

The authors also describe a simulation experiment based on this model with 30 nodes and $L = 1$ in a fully meshed network. The results show that after

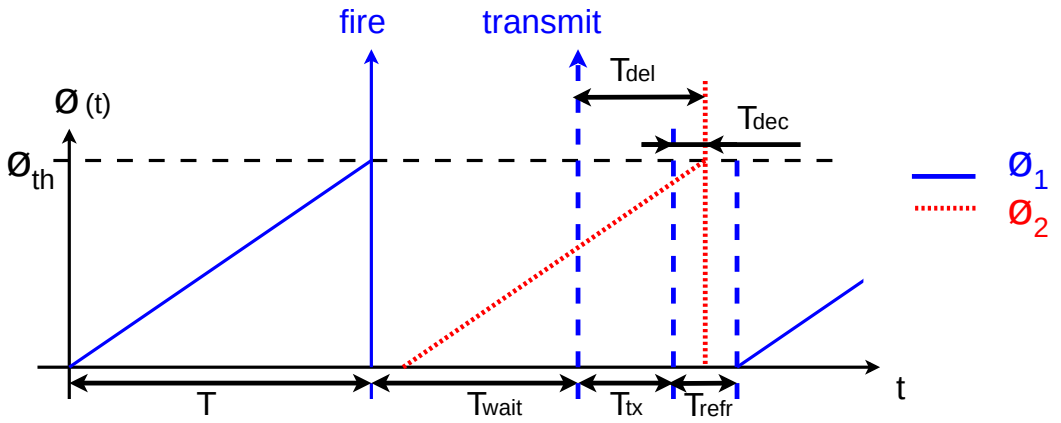


Figure 3.1: Phase diagram with $L = 1$.

an average time of about 15 periods the network converges and the nodes split into two groups each firing T seconds apart. Further, they mention that this strategy may be problematic in a multi-hop network.

It should be noted that this synchronization scheme may be inadequate for a real-world implementation, because if several synchronized nodes in the same group want to simultaneously transmit a synchronization burst, then the transmission delay will be extended due to the contention-based channel access mechanism and therefore degrades the accuracy.

3.4 Firefly-Inspired Sensor Network Synchronicity

This is the most important related work for this thesis and refers to [WATP+05]. Therein, the authors present an extended version of the PCO model, named Reachback Firefly Algorithm (RFA) which is well-suited for the implementation in sensor networks[2]. In order to reduce computing time for the calculation of the PRC, they simplified the function f described in Section 2.4.1 to $f(\phi) = ln(\phi)$. Consequently, the calculation of the phase jump equals $\Delta(\phi) = f^{-1}(f(\phi) + \epsilon)$ whereas ϵ defines the coupling strength. Next, after inserting function f in this term, we get a very simple linear phase jump:

$$\Delta(\phi) = \epsilon \cdot \phi$$

The next paragraphs in this section covers the simulation and testbed results of this work. To get a better understanding of the results, the authors introduced some new terms which are stated below.

[2]Note that the detailed functionality of the RFA model is described in Section 2.4.2

Group Spread: In order to measure the quality of synchronization in a network with different clusters, the authors introduced the notion of *group spread* which corresponds to the term precision, introduced in Section 2.2.1. The *group spread* is defined as the maximum time difference between any firings in a group, whereas the group is determined by a window size w and a clustering algorithm. Based on this window size, the algorithm classifies several groups out of a number of node firings with two constraints:

1. Every node firing must be classified into a group.
2. Every group contains as many firing events as possible.

Time To Sync: This defines the time until all nodes stay in a group without group change for 9 out of the last 10 periods.

50th and 90th Percentile Group Spread: The calculation of the percentiles is based on the distribution of group spread for all groups after the network has achieved synchronicity. More precisely, all group spreads are measured in the interval $[t_s + \frac{t_e - t_s}{2}, t_e]$, while t_s declares the first sync time and t_e determines the experiment end. This time containment avoids outliers caused by settling effects during the startup phase of the network.

Firing Function Constant (FFC): The FFC is defined to be the inverse of the coupling strength ϵ. This value defines the coupling between the oscillators and limits the maximum phase increment a node could make to be $\epsilon \cdot \phi_{th}$ respectively $\epsilon \cdot T$ in the time domain. Therefore, a small FFC results in a strong coupling, and conversely. Further, the authors state that if the FFC is too small, then the oscillators may overshoot and therefore prevents the network from achieving synchronicity. On the other hand, if the FFC is too large, then the time to sync will increase or the system will never get synchronized.

3.4.1 Simulation Results

The algorithm was simulated with TOSSIM in contrast to several parameter choices (e.g. different node topologies, FFC values, number of nodes).

The simulation results based on an *all-to-all* topology show that very small FFC values does not result in network synchronicity. On the other hand, larger FFC values increase the time to sync. Further, the authors observed that the time to sync does not exceed an upper limit of 400 time periods and that a large number of nodes increase the probability of overshooting, even if the FFC is very small.

Another simulation was based on a *regular grid* topology. In this topology a node has at most four neighboring nodes and therefor takes multiple hops

into account. They simulated different grid topologies of 4x4, 8x8 and 10x10 nodes. The results from this simulation are that FFC values in the range of $20 - 500$ usually result in synchronicity and larger values increase the time to sync, especially in larger grids. Further, they state that the time to sync is more dependent on the FFC value than on the network diameter. Nevertheless, the network diameter has a slightly impact on the group spread.

3.4.2 Testbed Results

The paper describes a testbed environment that comprises 24 MicaZ motes with a 7.3MHz clock, which are distributed over a large area in a building. This ensures a typical sensor network scenario with variable link quality. Based on this environment, the authors state that they observed a limit on the group spread due to the clock skew which is about $100\mu s$. To get a good overview of the achieved group spreads connected to the FFC, Table 3.2 shows the summarized results from [WATP+05].

FFC	**Time to sync**	**50th percentile group spread**	**90th percentile group spread**	**Mean Group std dev**
100	284.3 s	131.0 μs	4664.0 μs	410.4 μs
250	343.6 s	128.0 μs	3605.0 μs	572.2 μs
500	678.1 s	154.0 μs	30236.0 μs	1327.8 μs
1000	1164.4 s	132.0 μs	193.0 μs	63.6 μs

Table 3.2: Summarized testbed results [WATP+05].

4 Design Approach

In order to perform a distributed clock synchronization in an ensemble of nodes based on cheap RC-oscillators, we have to combat the synchronization of the clock state as well as the calibration of different clock rates due to temperature variations and initial drift. Therefore, two mechanisms are introduced in this chapter, where the first one performs the clock state correction and the second concept is responsible for clock rate calibration. It is important that both must be independent from each other.

4.1 Clock State Correction

The clock state correction in our design approach is based on the RFA and on the PCO model from Mirollo and Strogatz. Additionally to the algorithm described in Section 2.4.2, we used the definition of the smooth, monotonic increasing, and concave down function $f : [0, 1] \rightarrow [0, 1]$ from MaS to calculate the phase jump $\Delta(\phi)$. Note that the phase is a linear function of time and can take a value from the interval $[0, 1]$. Consider the dissipation factor b is greater than 0 and the pulse strength within $0 < \varepsilon < 1$, then the phase response function is therefore

$$\phi_{new}(\phi) = \min(1, f^{-1}(\varepsilon + f(\phi))) \tag{4.1}$$

and consequently the phase jump equals

$$\Delta(\phi) = \min(1, g(\varepsilon + f(\phi))) - \phi \tag{4.2}$$

where the smooth, monotonic increasing, and concave down function f is denoted by

$$f(\phi) = \frac{1}{b} \cdot ln(1 + [e^b - 1] \cdot \phi). \tag{4.3}$$

Because the direct implementation of all these functions would result in a time-consuming calculation process, we tried to minimize the equation by inserting

the inverse function f^{-1} in Equation 4.2. Let $f^{-1}(x) = \frac{e^{bx}-1}{e^b-1}$, then the simplified phase jump equals

$$\Delta(\phi) = \min(1, \alpha \cdot \phi + \beta) - \phi \tag{4.4}$$

where

$$\alpha = e^{\varepsilon b} \tag{4.5}$$

and

$$\beta = \frac{\alpha - 1}{e^b - 1}. \tag{4.6}$$

Assuming a strong dissipation factor $b \gg 1$, then β is negligible and the approximated phase jump function is reduced to

$$\Delta(\phi) = \min(1, \alpha \cdot \phi) - \phi. \tag{4.7}$$

In addition to that, due to a very small pulse strength ε we can assume $0 < \varepsilon \cdot b \ll 1$. Consequently, the dominant first order of the Taylor expansion allows us to define $e^{\varepsilon b} = (1 + \varepsilon b)$. Inserted into Equation 4.7, we finally get the most simple phase jump function. That is

$$\Delta(\phi) = \begin{cases} \varepsilon b \cdot \phi & \text{if } \phi \cdot (1 + \varepsilon b) < 1 \\ 1 - \phi & \text{else} \end{cases} \tag{4.8}$$

Therefore, we have a linear PRC where the constant α specifies the coupling strength for the oscillators and depends on the product of the dissipation factor b and the pulse strength ε. This result is similar to the simplified firing function described in [WATP+05].

4.1.1 Firefly Communication

In order to apply the simplified PCO model in a wireless system, we encounter some difficulties due to the assumptions which result from the mathematical model. As already described in Section 2.4.2, these assumptions are again listed below followed by a discussion of possible problems.

1. The oscillators have identical dynamics (e.g. same period).
2. Nodes can instantaneously fire.
3. Every firing event must be observed immediately (no loss).
4. All computations are performed perfectly and instantaneously.

Natural imprecision. The PCO model assumes that the oscillators and therefore the clocks on the nodes have exactly the same frequency and no variations over time. In reality this is not possible due to environmental influences and limits of fabrication and measurements. Additionally, computational imprecision due to floating point operations or truncating are a problem. However, the nature is also not perfect and therefore the approach presented here should also be able to deal with small frequency fluctuations. For this reason, we say that a node is synchronized if the maximum absolute deviation to all other nodes is smaller than a predefined synchronization window, denoted by w.

Communication delay. The next two assumptions are again usually impossible to accomplish, especially if the exchange of fire events is based on the communication characteristics of a wireless system. For instance, the message transfer usually suffers from unpredictable delay. If the delay is predictable and constant, then this will not be a problem so far. However, the unpredictable jitter demands a more complex solution. Further the link between two nodes in a wireless network is often asymmetric and may vary due to local changes. Therefore, in order better to be able to describe the behaviour of our approach, the important characteristics regarding the communication delay are analyzed in the next paragraphs. Further these delays are subdivided according to the different layers of the ISO OSI seven-layer model. It should be noted that the underlying protocols used in wireless sensor networks are often kept relatively small and simple in order to reduce memory and energy consumption and therefore are only declared at the physical layer respectively at the MAC layer which is comprised in the data link layer. For this reason, the next paragraphs describes only delays according to these layers and, of course, delays concerning the application layer.

Delays at the physical layer. The most significant delay according to the physical layer is generally caused by the transmission of a message, denoted by T_{tx}. The transmission delay mainly depends on the data rate and on the message size. Another important delay concerning the physical layer is the propagation delay T_{prop}. However, in wireless systems, the propagation delay is in the order of μs and therefore usually negligible.

Delays due to MAC layer. Delays caused by the MAC layer are much more greater than those caused by the physical layer and is denoted by T_{MAC}. The MAC-delay heavily depends on the medium and the type of medium access strategy. Furthermore, because the communication in wireless systems is generally contention-based, the protocols often use the CSMA/CA technique to destine the nodes which are allowed to send first. As described in Section 2.6.5, this mechanism makes use of backoff periods and

if a node has lost its contention, it will choose a new random contention window based on a greater backoff value. This results in unpredictable delays in the order of several milliseconds. Moreover, consider all nodes want to transmit simultaneously. This would additionally extend the backoff periods and make things worse. Last but not least, other additional reasons which increases T_{MAC} may depend on the protocol implementation and can be caused by the processing of MAC events stored in a First In First Out (FIFO) buffer, or simple interrupts which can be nasty if they occur very frequently.

Application-specific delays. Delays at the application layer are usually caused by program execution. Typical examples are message preparation, message encoding/decoding, and many other computational functions according the message delivery. The delay itself is not a big problem, if it is constant and predictable. But this is not true if the delay varies over time and thus produces a jitter which is defined to be the deviation between the delays. Because the jitter is not predictable, it is only possible to define an upper bound for it. However, this bound may be great if the program is written in an inefficient style, so that the delay heavily depends on the execution flow after a branche. This effect can be kept to a minimum by applying programming constraints and models, and performing path analysis (e.g., Worst-Case Execution Time (WCET) - oriented programming [Pus05]).

Prerequisites for precision improvements. The main goal of our synchronization model is to improve the precision in an ensemble of nodes by eliminating the measurable delay, especially those resulting from the MAC layer. The best way to do this is MAC-timestamping. Therefore, a received message contains a variable which indicates the duration required for passing down the send request from the application layer to the MAC layer. This delay is denoted by $T_{MAC,tx}$ and also includes the time spent on waiting for the transmission admission due to the CSMA/CA scheme. Additionally, the duration required for passing the data up to the application layer at the receiver is denoted by $T_{MAC,rx}$. Thus, the complete communication delay concerning the MAC layer and the physical layer is

$$T_{MAC} = T_{MAC,tx} + T_{tx} + T_{MAC,rx}.$$

It should be noted that MAC-timestamping must be supported by the underlying protocol or otherwise the precision may be lower bounded to the variations in T_{MAC} denoted by ε_{MAC}. Additionally, the clock used at the MAC layer for message transmission or MAC-timestamping should be more accurate than those used at the application layer.

However, this requirement does not eliminate the problem of simultaneous transmissions, because the resulting extended backoff periods may be too large, so that the delayed messages may be ignored or disturb the flow of synchronization. A remedy is to extend the contention-based strategy from the MAC layer to the application layer. Assuming that all nodes in a network are synchronized, then this advanced communication contention takes advantage of choosing a random time of an application based contention window, called *firing offset*. In detail this contention window is determined by two constants, named *minimum firing offset* ($T_{MinOffset}$) and *maximum firing offset* ($T_{MaxOffset}$). Considering the Firefly algorithm, instead of transmitting a firing message at the end of each period, the synchronization message will be transmitted exactly T_{Offset} seconds prior, where T_{Offset} is a random time between $T_{MinOffset}$ and $T_{MaxOffset}$. Figure 4.1 visualizes this behaviour of the offset interval. After receiving such a message, the receiver has to correct the timestamped message by the duration of the firing offset. Otherwise, the receiver will not be able to recover the real period end of the sender. Unfortunately, this strategy may also degrade the precision according to the granularity of the clock used for determining the phase and, therefore, the firing event.

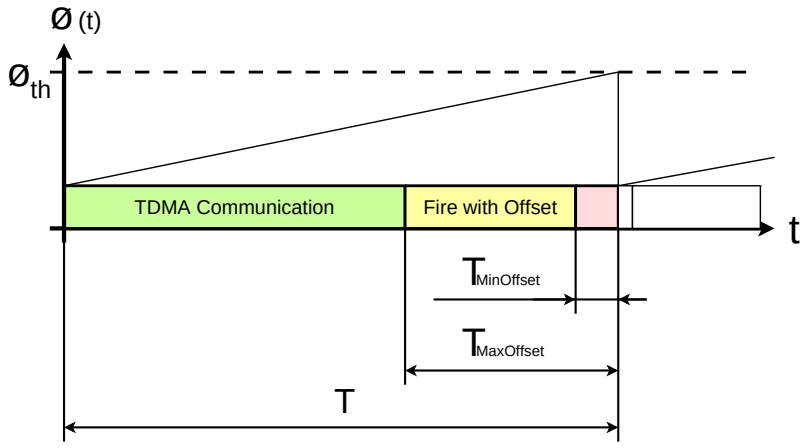


Figure 4.1: Phase diagram with firing offset.

Choosing of $T_{MinOffset}$. In order to avoid that, transmitted messages are received after the end of a period, the minimum firing offset should be greater than the maximum possible communication delay. To consider unpredictable delays and the bounded precision defined by the synchronization window w, we assume the following:

$$T_{MinOffset} > 2 \cdot w + T_{MAC,max}$$

Choosing of $T_{MaxOffset}$. The maximum firing offset depends on the number of nodes which are comprised in the network. For instance, many nodes in an all-to-all topology may require a greater $T_{MaxOffset}$ than a small network comprising only a few number of nodes. This comes from the fact that the synchronization precision mainly depends on the information a node receives from the neighbors. The difference between the maximum and the minimum firing offset defines the bandwith which can be used by the nodes. Consequently, the bandwith should be so large that there is enough time for each node broadcasting its synchronization message. Assuming an all-to-all network comprising N nodes and because the maximum bandwidth required for such a synchronization message is mainly destined by the transmission time, we can define:

$$T_{MaxOffset} > N \cdot T_{MAC,max} + T_{MinOffset}$$

Modified RFA algorithm. The above described assumptions requires a modification of the RFA algorithm considering MAC-delay, firing offset and the simplified phase jump function. In detail, our Firefly synchronization algorithm can be split into five principal operations:

1. **Sort firing events.** The sort operation is necessary in order to get a list of the ascending ordered instantaneous firing events for the phase jump calculation. Otherwise, if the events are processed in the order they arrived, then it's highly likely that the real instantaneous firing event of a later arrived event is prior to the first occurred events. Note that the real instantaneous firing event is calculated by summing the relative timestamp with the firing offset. For example consider that a node has received n firing events during a period. Then it reaches the end of the period and therefore starts this algorithms. Considering $T_{i,stmp}$ denotes the timestamp of event i with $1 \leq i \leq n$, $T_{i,Offset}$ denotes the firing offset of event i and $T_{i,MAC}$ declares the corresponding MAC-delay, then after applying the sort algorithm, the real instantaneous firing events are determined by

   $$\epsilon_i = T_{i,stmp} + T_{i,Offset} + T_{i,MAC}$$

   with

   $$\forall i, j \in \{1, \ldots, n\} : i < j \Longrightarrow \epsilon_i \leq \epsilon_j.$$

2. **Remove least and greatest events.** In order to compensate messages from erroneous nodess, the least and greatest firing events are removed. Otherwise, a single faulty node sending arbitrary offsets may degrade the precision or moreover may keep the network unsynchronized. In this context it should be noted that the removing of such

events also reduces the amount of synchronization information and therefore will also affect the precision.

3. **Merge neighboring events.** It is important to merge several neighboring events, because this avoids overshooting according to too many firing events. For instance, considering the simplified phase jump Function 4.8, then if the number of firing events denoted by n is so large that the overall phase jump reaches the threshold, i.e., $\sum_{k=1}^{n} \phi_k \alpha^{n-k+1} \geq \phi_{th}$, then no phase shift would be performed. However, all these firing events are neighboring and can be seen as a single event. This also corresponds to the natural behaviour of fireflies. If such an insect is exposed to a swarm of synchronized fireflies which flash slightly apart, then the exposed Firefly will only adjust its phase as there is a single flash. According to [Buc88], the insects may also react stronger to a brigher light. But we simplified this behaviour by building clusters defined by a cluster interval, each containing an amount of firing events. The real instantaneous firing event of the cluster is therefore calculated through partial averaging. It should be noticed that the choice of the clustering interval has also an impact on the precision. Therefore, it is advantageous to implement a clustering algorithm similar the one described in [WATP+05] which has the following constraints:
   a) Every node firing must be classified into a group.
   b) Every group contains as many firing events as possible.
4. **Calculate overall phase jump.** The calculation of the overall phase jump is based on the original RFA algorithm, but uses the previous described clusters instead of the atomic firing events. The modified RFA version is declared in Algorithm 2. This algorithm first sets a variable named *overall_phasejump* to 0 which will also define the start condition (start phase) for the next period. Then a loop proceeds all ascending ordered real instantaneous firing events and calculates the local phase jump with respect to the current overall phase jump value for each cluster $\tilde{\epsilon}_i$, where each phase jump is cumulatively added to the overall phase jump variable. Note that the if-statement in line 3 ensures that only event clusters prior the period end (phase threshold ϕ_{th}) are taken for the computation. Afterwards if the calculated overall phase jump is less than the period (phase threshold ϕ_{th}), then this phase is returned. Otherwise, if the if-condition in line 8 is not valid, then the zero phase will be returned. This ensures that a too strong excitation due to many phase jumps makes the calculation predictable and further is used as an important assumption for choosing the coupling parameters.

Algorithm 2 RFA algorithm considering MAC-delay and firing offset.

calc_overall_phasejump()

```
1: overall_phasejump = 0
2: for i = 1 to #cluster do
3:    if ε̃_i ≤ φ_th then
4:       phasejump = Δ(ε̃_i + overall_phasejump)
5:       overall_phasejump = overall_phasejump + phasejump
6:    end if
7: end for
8: if overall_phasejump ≥ φ_th then
9:    overall_phasejump = 0
10: end if
11: return overall_phasejump
```

5. **Adjust phase.** Finally the previous calculated overall phase jump is applied to the phase. This is done by changing the start condition of the phase for the next period. For instance, instead of starting with phase 0, a node starts with the content of the *overall_phasejump* variable returned in the algorithm above. Note that if the *overall_phasejump* $\geq \phi_{th}$, then the start phase is set to 0. Furthermore, it is very important to suppress the firing in the current period, if the start phase is greater or equal half the threshold phase. This is a major assumption for choosing the coupling parameters and further eliminates the occurrence of fixpoints. A fixpoint is for example the state, where two nodes have such a phase, so that after each period they only change their phase states. In other words the phase state of both nodes is the same in each second period. The following short pseudocode in Algorithm 3 illustrates this behaviour:

Algorithm 3 Algorithm for adjusting the calculated overall phase jump.

```
1: overall_phasejump = calc_overall_phasejump()
2: if overall_phasejump ≤ φ_th then
3:    start_phase = overall_phasejump
4:    if overall_phasejump < φ_th/2 then
5:       set offset time for firing
6:    else
7:       suppress firing for the current period
8:    end if
9: end if
```

4.1.2 Lower Bound for the Coupling Parameter α

In order to obtain good synchronization results with respect to different parameter choices, it is important to find bounds for these parameters. Further we want to calculate the optimal coupling factor with respect to the clock drift, so that the Firefly algorithm can still synchronize the nodes. For this reason, we first denote a virtual clock of node k with VT^k. Next we want to get the duration of the synchronization interval of a specific clock k. For this we introduce the term I^k, which determines the interval of clock k with respect to the reference clock z.

$$I^k = z(VC^k_{i+T}) - z(VC^k_i)$$

Moreover we need to define the maximum deviation between the intervals of any clocks in an ensemble and is herein after referred to as the *virtual clock skew*, denoted by T_{skew}.

$$T_{skew} = \max_{\forall j,k}(|I^j - I^k|)$$

Without loss of generality we assume $I^j > I^k$. If the Firefly algorithm is based on Equation 4.7, then the coupling factor α must be greater than $\frac{I^j}{I^k}$. Otherwise, the system will never get synchronized.

The proof is trivial. Consider the clocks are already perfectly synchronized, but because of the virtual clock skew the node with the shorter interval I^k will reach the phase threshold ϕ_{th} earlier than the other one with the interval I^j. Consequently, node k fires and node j receives the fire event at ϕ^j_{fire}, where $\phi^j_{fire} = \phi_{th} \cdot \frac{I^k}{I^j}$. Further the phase jump performed by clock j at the phase ϕ^j_{fire} is denoted with $\Delta(\phi^j_{fire})$. In order to keep the system synchronized, the following equation must be valid:

$$\Delta(\phi^j_{fire}) + \phi^j_{fire} \geq \phi_{th}$$

Otherwise, the system will definitely get unsynchronized. Note that ϕ_{th} is defined to be 1. So the result after substituting $\Delta(\phi^j_{fire})$ with Equation 4.7 is:

$$\alpha \geq \frac{I^j}{I^k}$$

Hence if the equation above is valid, the precision of an ensemble of clocks based on the Firefly algorithm is lower bounded to T_{skew} and the overall synchronization interval is determined by the clock with the shortest period.

$$\Pi \geq T_{skew}$$

4.1.3 Upper Bound for the Coupling Parameter α

It is obvious that the coupling parameter α for the phase jump function must have an upper bound. Otherwise, the clocks will never get synchronized due to a mutual excitation. In detail, if α is so large that the phase jump function $\Delta(\phi)$ at the phase ϕ always returns a phase jump of about $1 - \phi$, than this will definitely keep a network of more than two nodes unsynchronized. It should be noted that if the network is comprised of exactly two nodes, then the system will achieve synchronicity. Otherwise, in the case of three or more nodes, it can be shown that the network will never get synchronized. However, if a network is already perfectly synchronized and we neglect the communication jitter and any inaccuracy, then the system will keep synchronized independent of the choice of the coupling parameter α.

Theorem: The coupling factor α must be chosen, so that the following condition

$$\alpha < \frac{2 \cdot N}{2 \cdot N - 1}$$

is valid, where N denotes the maximum number of fire events a node may receive within a period, i.e., the maximum number of phase jumps a node may perform.

The equation is based on the assumption that the maximum allowed overall phase jump must not be greater than half the threshold phase ϕ_{th}. Therefore, we denote the maximum possible local phase jump with Δ_{max} which equals $\phi_{th} - \frac{\phi_{th}}{\alpha}$. Next we assume the worst case that a node performs at most N phase jumps. This corresponds to the reception of N fire events. Consequently, the equation $N \cdot \Delta_{max} < \frac{\phi_{th}}{2}$ must hold and after solving the equation for α, we come to the result $\alpha < \frac{2 \cdot N}{2 \cdot N - 1}$.

4.1.4 Rate of Synchronization

The authors from [MS90] and [WATP+05] have analyzed the synchronization rate for a network with two oscillators. In this case, they have proven that the time to synchrony is inversely proportional to the coupling factor α and further depends on the initial phase difference of the nodes at $t = 0$, denoted by $\delta = \phi_A^{(0)} - \phi_B^{(0)}$. Therefore, the number of iterations k until synchronicity equals

$$k \approx \frac{1}{\alpha} \cdot \ln \frac{1}{2 \cdot \delta - 1}.$$

Mirollo and Strogatz have also analyzed the case of n oscillators. However considering a multi-hop topology requires a more sophisticated solution and is stated in [LW04].

4.2 Clock Rate Calibration

The concept of clock rate calibration was incorporated into the design approach, in order to combat the problem of frequency deviations due to the use of low-cost sensor nodes in a network. If no clock rate calibration is performed, than this will result in a bad precision and may lead to the loss of synchronization. This is especially a problem in combination with the Firefly algorithm presented above. Note that the rate correction is performed by a virtual clock and therefore is based on the change of the granularity by applying an adjustment value, denoted with $H^k(t)$[1]. In other words this adjustment value changes the threshold for the clock.

Moreover, the general interpretation of the virtual clock concept may result in different implementations. For instance, the realization can behave in more smaller time granules or otherwise in few extended granules. From an objective point of view both implementations are certainly based on the same technique of macroticks and microticks. The first variant uses a number of macroticks to determine the duration of the synchronization interval and therefore needs an additional software counter. The other scheme represents the macrotick as the biggest applicable time unit which corresponds to the complete synchronization interval. The first variant may be more adequate for the theoretical concept of virtual clocks, because thus the software level has again the perception of ticks with a small granularity. In reality this technique produces many interrupts and may disturb the execution flow of a node. Contrary the second version produces only one interrupt for each period. This is indeed better but may be also a problem if the execution start of several tasks are based on a distinct point in time during a synchronization interval (e.g., Time-Triggered-Protocols), because the nominal point of time must be scaled to the overall interval, i.e., the nominal interval with the adjustment value H. Furthermore, if events are recognized during the interval, the timestamp must again be normalized to get a rational timestamp with respect to the standard interval. All these interconversions require some computation and may reduce the performace of the application. This solution may be interesting if there is no need for a frequently virtual clock read or write access during a synchronization interval, because this may increase the computational effort and thus worsens the precision. This comes from the fact that the virtual clock abstracts from the internal processing and every read and write access for changing the clock state must be done with respect to the *real phase threshold*. Assuming that all calculations relate to the notion of phase, then for adjusting the clock state the virtual clock expects a value from 0 to ϕ_{th}. Internally, the virtual clock has to scale this value to the real phase threshold $\phi_{th,real}$ which equals the addition

[1]Note that the adjustment value H is expressed as a phase.

of the nominal phase threshold ϕ_{th} and the adjustment value H. Conversely the timestamp of an event during a period is usually measured in microticks and consequently must be normalized if this timestamp is used outside of the virtual clock. For example if a given clock state $\phi_{set} \in [0 \dots \phi_{th}]$ should be set, then the content of the counting register which corresponds to the internal real phase $\phi_{set,real}$ should equal

$$\phi_{set,real} = \phi_{set} \cdot \frac{\phi_{th,real}}{\phi_{th}}$$

Further if the content of the phase register is read then the value outside of the virtual clock should equal:

$$\phi_{read} = \phi_{read,real} \cdot \frac{\phi_{th}}{\phi_{th,real}}$$

The advantage of this implementation is that the interrupts can be kept to a minimum and is only relevant in applications where frequently interrupts may disturb the progress of an application.

Our approach is based on the second virtual clock strategy, because the other method produces many interrupts which are unacceptable due to the MAC protocol. Consequently, the adjustment value directly maps to the amount of increase or decrease of the phase threshold ϕ_{th}. Let H_i^k be the adjustment value of clock k at period i, then the real phase threshold is $\phi_{th,real,i}^k = \phi_{th} + H_i^k$.

The core concept of our rate calibration algorithm is that a node stores the measured durations of one or more synchronization intervals of all neighboring nodes. In order to reduce the effect of jitter, the last eight measurements are stored in a buffer. The average of all these values results in a better assumption about the real interval duration of a neighboring node. This calculated interval is then used to adopt the own adjustment value H. There exist also the alternative to calculate the averaged interval via linear regression based on the deviations between the measured interval and the nominal interval over several periods. However, the linear regression approach needs a more intensive computation and the experiments show that the overall averaging approach is good enough for rate calibration. In order to adjust the clock speed to be in accord with the neighboring nodes, every node has to get the local view of the neighbor's clock intervals and then calculates the deviation to the own interval. It is important that the term "local view" means that the measurement is based on consistent microticks. In other words the duration between two consecutive microticks of a node must be constant and the measured interval must be independent of the sender's adjustment value H of the virtual clock. In detail, a node always measures the number of consistent microticks which

correspond to the nominal number of microticks at the sender. This method is demonstrated in Figure 4.2. In this diagram, the sender node is slower than the receiver node and therefore the receiver node has to extend its phase threshold ϕ_{th} to $\phi_{th,real}$ by adding the adjustment phase H. However, if such a direct adaption is used in reality, then this will result in a mutual adjustment of the threshold. For this reason, we introduced a smooth adjustment which can also be realized in a simple PI-controller with a dominant proportional part.

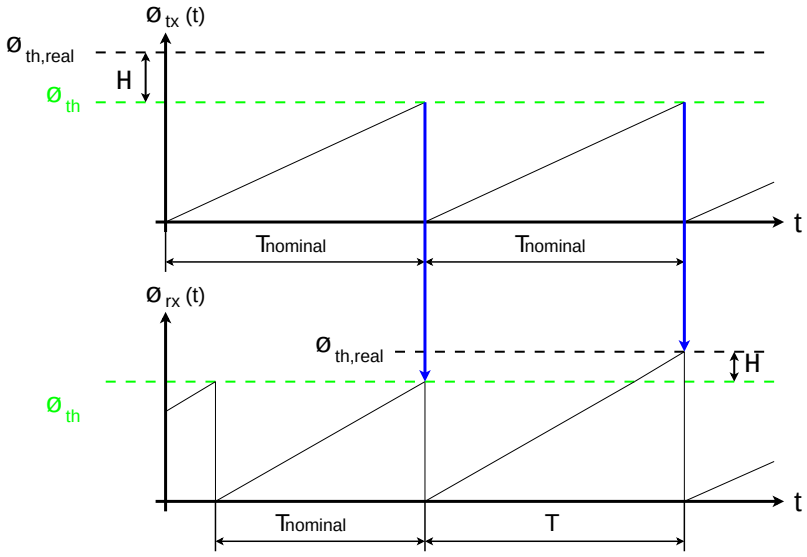


Figure 4.2: The principle of rate calibration.

In general the rate calibration algorithm can be distinguished in the following parts which are described in detail in the next subsections:

1. Interval measurement and buffering.
2. Calculation of the real interval for each individual node.
3. Sorting of these intervals.
4. Remove least and greatest intervals (to introduce fault-tolerance).
5. Calculate overall average interval.
6. Smooth computation of the adjustment value H.
7. Increase or decrease the real phase threshold $\phi_{th,real}$ based on H.

4.2.1 Interval Measurement and Buffering

As mentioned above, a node has to measure the nominal intervals of the neighboring nodes in the own granularity and in absence of the virtual clock. However, the only information a node gets from the neighbors are the fire messages

which may contain some additional data to extract the number of ticks in the sender's microtick granularity. So relating to the phase, this amount of ticks corresponds to the state of the real phase $\phi_{real}(t)$[2] and not the normalized phase $\phi(t)$. Further we denote the receiver node with the letter r and the sender nodes with the variable j which can acquire a value from 1 to n where n declares the maximum number of sender nodes. Additionally, in order to measure the interval duration, every fire event has to be timestamped. For instance, if the fire event e^j_{fire} from the sender j is received at the node r, then the timestamp of this event at the receiver is denoted by $C^r(e^j_{fire})$. To distinguish different fire events from the same node, the events are chronology ordered and furthermore numbered serially. Thus, two consecutive fire events from a node j are labeled with $e^j_{fire,k}$ and $e^j_{fire,k+1}$, where k defines the chronology ordered position in the *rate-calibration buffer* and can take a value from 1 up to the maximum capacity of the buffer, denoted with m. The duration between two fire events from a sender j in the receiver's clock granularity corresponds to the difference between the two timestamps and is declared as

$$I^r_{j,k} = C^r(e^j_{fire,k+1}) - C^r(e^j_{fire,k})$$

. Assuming that the variability in transmission delay due to the contention based access strategy at the MAC layer is relatively large, then we have to decrease the measured interval duration by the MAC delay T_{MAC}. This requires that the measurement of T_{MAC} is based on the same clock used for timestamping. Otherwise, it is all but impossible to scale this delay to the same granularity like the timestamping clock. For this reason, there may be always a small error, especially if the jitter in MAC delay is very big.

The duration between two consecutive fire events does not correlate to the nominal interval at the sender, because it contains the phase jump and the difference of the firing offsets from the first and second fire event. For further explanations we denote the normalized phase offset of a fire event k from a node j by $\phi^j_{offset,k}$ and the corresponding normalized phase jump from the same period with Δ^j_k. All these values are scaled according to the phase adjustment value H^j_k of the corresponding period k. In detail, the following analogon can be postulated:

$$I^r_{j,k} \equiv \phi^j_{offset,k} \cdot \frac{\phi_{th} + H^j_k}{\phi_{th}} + (\phi_{th} - \phi^j_{offset,k+1} - \Delta^j_{k+1}) \cdot \frac{\phi_{th} + H^j_{k+1}}{\phi_{th}}$$

[2]Note that the real phase also corresponds to the physical clock, but periodically resets to 0 if it reaches the real threshold $\phi_{th,real}$.

The right term defines the equivalent number of consistent microticks at the sender node. However, we want to get the number of ticks in the receiver's granularity for the corresponding nominal number of ticks at the sender, denoted by $\tilde{I}^r_{j,k}$. According to the rule of three and after simplifying the right term, we come to the following result:

$$\tilde{I}^r_{j,k} = I^r_{j,k} \cdot \frac{\phi_{th}}{\phi_{th} + H^j_{k+1} + \phi^j_{offset,k} \cdot \frac{\phi_{th} + H^j_k}{\phi_{th}} - (\phi^j_{offset,k+1} + \Delta^j_{k+1}) \cdot \frac{\phi_{th} + H^j_{k+1}}{\phi_{th}}}$$

Now $\tilde{I}^r_{j,k}$ corresponds to the nominal number of ticks of the sender's node.

Although the calculation seems to be clear, it is maybe necessary that the computation is kept to a minimum. Furthermore, if the duration measurement should be performed over several periods (e.g. 3 intervals) in order to reduce the influence of the communication jitter, then a synchronization message must not be lost. Otherwise, the incomplete information of the relevant periods makes it impossible to calculate the overall duration. It should be noted that the duration measurement over several periods only compensates the short term drift of the oscillators. The overall average interval of all synchronized nodes will never be constant over a longer time period (e.g. several days). Instead the interval duration will marginally vary over minutes or hours. But it in our approach there is no demand on the compensation of the long-term drift, because the approach is not aimed at achieving time synchronization. Instead the term synchronicity does tolerate interval variations as long as the deviation is small enough. The short-term drift is always present and has a deeper impact on the precision.

There exist a tradeoff between the synchronization precision and the measurement interval. Whereas a short measurement interval improves the precision and degrades the long-term stability, a long measurement interval degrades the precision and improves long-term stability. For this reason, an easier approach for identifying the amount of ticks at the sender node could be performed by an additional timestamping at the sender node. But this method may increase the communication overhead and therefore degrades the energy efficiency. Let $C^j(e^j_{fire,k})$ be the timestamp from the sender before the fire message is transmitted, then we can denote the equivalence much more simpler with:

$$I^r_{j,k} \equiv C^j(e^j_{fire,k+1}) - C^j(e^j_{fire,k})$$

and similarly $\tilde{I}^r_{j,k}$ can be calculated with the following term:

$$\tilde{I}^r_{j,k} = I^r_{j,k} \cdot \frac{\phi_{th}}{C^j(e^j_{fire,k+1}) - C^j(e^j_{fire,k})}$$

Concluding, the main difference between both approaches is the amount of data which have to be included in a fire message. The Table 4.1 contains a comparison of the data which is needed to transmit and further has to be stored in the receiver buffer and in the rate-calibration buffer for the rate calibration algorithm. Note that although the timestamping version needs no data about the phase offset, it still has to be transmitted and stored in the receiver buffer for the state correction algorithm. But the advantage of this version is that it accepts lost timestamp messages.

Without Timestamping	**Timestamping Approach**
Transmission Data	
node-ID	
phase state $\phi^j(e^j_{fire})$ (for RFA algorithm)	
phase offset ϕ^j_{offset} (for RFA algorithm)	
phase adjustment value H^j	
phase jump Δ^j	sender timestamp $C^j(e^j_{fire})$
Receiver Buffer Data	
node-ID	
phase state $\phi^j(e^j_{fire})$ (for RFA algorithm)	
phase offset ϕ^j_{offset} (for RFA algorithm)	
phase adjustment value H^j	
receiver timestamp $C^r(e^j_{fire})$	
phase jump Δ^j	sender timestamp $C^j(e^j_{fire})$
Rate-Calibration Buffer Data	
node-ID	
receiver timestamp $C^r(e^j_{fire,k})$	
phase adjustment value H^j_k	latest phase adjustment value H^j
phase jump Δ^j_k	sender timestamp $C^j(e^j_{fire,k})$

Table 4.1: Comparison of demands for additional communication data for the different interval calculation approaches.

4.2.2 Calculation of the Average Interval for each Node

Now we have measured the duration of the neighbor's nominal synchronization intervals in the receiver's granularity. However, this interval does not reflect the real interval length used for the Firefly algorithm. For this reason, the receiver

has to scale the measured nominal intervals according to the corresponding sender's latest phase adjustment value, denoted by H^j. The resulting sender's real synchronization interval in the receiver's granularity corresponds to

$$\hat{I}^r_{j,k} = \tilde{I}^r_{j,k} \cdot \frac{\phi_{th} + H^j}{\phi_{th}} = \frac{C^r(e^j_{fire,k+1}) - C^r(e^j_{fire,k})}{C^j(e^j_{fire,k+1}) - C^j(e^j_{fire,k})} \cdot (\phi_{th} + H^j)$$

or with respect to the version without timestamping:

$$\hat{I}^r_{j,k} = \frac{(C^r(e^j_{fire,k+1}) - C^r(e^j_{fire,k})) \cdot (\phi_{th} + H^j)}{\phi_{th} + H^j_{k+1} + \phi^j_{offset,k} \cdot \frac{\phi_{th} + H^j_k}{\phi_{th}} - (\phi^j_{offset,k+1} + \Delta^j_{k+1}) \cdot \frac{\phi_{th} + H^j_{k+1}}{\phi_{th}}}$$

As a next step all scaled intervals of the same node are averaged. This ensures that the error due to the jitter and imprecision in computation is reduced. If the capacity of the rate-calibration buffer is big enough, then the error should be negligible. Our experiments have shown that a buffering over eight periods is good enough and requires not so much memory. Further we denote the average interval at the receiver r for each sender node j with $\bar{I}^r_j$ and equals

$$\bar{I}^r_j = \frac{1}{m-1} \sum_{k=1}^{m-1} \hat{I}^r_{j,k}$$

where m denotes the capacity of the rate-calibration buffer in periods.

4.2.3 Sorting, Removing and Overall Averaging

In order to involve the occurrence of erroneous nodes, we have introduced a simple concept to avoid the worst case. For instance, a network may contain a node with a very bad clock drift which heavily deviates from the other ones. Consequently, the network would take a long time to get synchronized or in the worst case would never get synchronized. This originates from a too big adjustment value H. The precision of a network will never get better than the microtick granularity of the slowest node due to the concept of virtual clocks.

A simple way to exclude such an erroneous node in the clock rate calibration algorithm is to remove the nodes with the greatest and the least clock drift. All other nodes are used for the overall averaging. As a result the nodes in a network based on an all-to-all topology should always achieve a common average drift. The overall average interval $\bar{\mathcal{I}}^r$ is calculated like the averaging of the node-specific intervals. Hence let n denote the number of nodes, then the overall averaged interval is calculated with the following equation:

$$\bar{\mathcal{I}}^r = \frac{1}{n} \sum_{j=1}^{n} \bar{I}^r_j$$

Assuming that the sender nodes are ordered by the clock drift, then the removing of the greatest and least node can be performed by starting the summation at the second node and end it at the last but one.

4.2.4 Computation and Employment of the Phase Adjustment Value

It is obvious that H can be calculated through a simple subtraction of the before described overall average interval $\bar{\mathcal{I}}^r$ and the nominal interval which also equals the nominal phase threshold ϕ_{th}. However, the results from several experiments show that such a direct adjustment causes a continuous increase of the overall average drift. In other words the intervals are getting longer and longer. For this reason, we decided to introduce a smooth adaption. Therefore, the phase adjustment value H should smoothly converge to the overall average interval. The equation for this calculation is stated below.

$$H^r = H^r_{old} \cdot (1 - \sigma) + (\bar{\mathcal{I}}^r - \phi_{th}) \cdot \sigma \tag{4.9}$$

The parameter σ defines the level of smoothness and must be in the range of $(0, 1)$. In detail, if σ is very small, then the adaption will be performed slowly. Otherwise, a great σ results in a fast adaption. As mentioned above the experiments show that a small σ let the overall average interval getting shorter and a greater σ lets it getting larger. It seems that if the measurement interval is greater than one period, then $\frac{1}{2}$ is mostly the swell for the change of the drift behaviour. This property makes it possible to use it as a control parameter for the overall offset drift stabilisation. The overall offset drift therefore means the slow but permanent increase or decrease of the adjustment values H of all nodes.

It is interesting that the formula for the smooth rate calibration presented above can be transformed so that it is similar to a well known closed-loop control system depicted in Figure 4.3. Compared with our rate calibration formula, the calculated overall average interval equals the setpoint of the control system, denoted by w. Similarly the control value u matches the result of Equation 4.9, where $u \hat{=} H^r + \phi_{th}$. In the following we denote the term $H^r + \phi_{th}$ with $I^r_{control}$. Assuming that the disturbance variable d is negligible, then the actual value y can be put on a level with the control value from the previous calculation, declared by $I^r_{control,old}$.

Regarding the Equation 4.9, if we replace the adjustment value H^r with the equivalent term $I^r_{control} - \phi_{th}$ and H^r_{old} with $I^r_{control,old} - \phi_{th}$, then after some transformations we come to the following formula

$$I^r_{control} = I^r_{control,old} + \sigma \cdot (\bar{\mathcal{I}}^r - I^r_{control,old}) \tag{4.10}$$

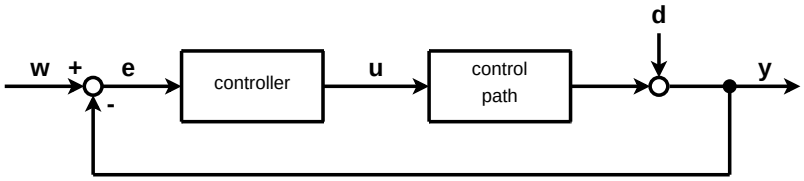


Figure 4.3: The general model of a closed-loop controlled system.

which directly maps to the previous described control loop with a simple proportional control. Further note that the term $\bar{\mathcal{I}}^r - I^r_{control,old}$ indicates the error variable e. An extensive interpretation of the above described proportional behaviour could be the more complex concept of a digital Proportional-Integral-Differential (PID)-controller. For this reason, we introduce the variables e_{old} and e_{old2} which correspond to the error variable from the previous period respectively from the last but one period. Taken together these results can be described in a pseudocode, depicted in Algorithm 4 where the factor q_0 equals the variable σ. In this context the results of several experiments indicate that

Algorithm 4 Calculation of the adjustment value based on a digital PID-controller.

1: $e = \bar{\mathcal{I}}^r - I^r_{control,old}$
2: $I^r_{control} = I^r_{control,old} + q_0 \cdot e + q_1 \cdot e_{old} + q_2 \cdot e_{old2}$ {with $q_0 = \sigma$}
3: $e_{old2} = e_{old}$
4: $e_{old} = e$
5: $I^r_{control,old} = I^r_{control}$
6: $H^r = I^r_{control} - \phi_{th}$

the factors q_1 and q_2 does not improve the precision of the rate calibration. This is logical since a differential part would only enforce the influence of jitter. However, this requires more intensive empirical studies to get a better understanding of the relation between the parameters and the resulting effects.

Average interval drift stabilisation. As mentioned above, the smoothing factor σ has an impact on the overall interval drift. So it seems to be impossible to hold the duration of the common period constant. However, this factor can also be used to compensate this problem. As already mentioned, the midpoint value for the drift change is about $\sigma = \frac{1}{2}$. For this reason, we only have to find a parameter which can be taken to control the value of the smoothing factor. An easy way is to build the average over all received adjustment values H_j during a period. This value must then be bounded to a phase which corresponds to a time within $\pm 10ms$. The truncated phase is then scaled down so that it is in the range of $[-\frac{1}{2}, +\frac{1}{2}]$. Consequently, the smoothing factor will range from 0 to

1. Because a factor of 0 stops the calibration effect of the algorithm, there must be a lower bound of about $\frac{1}{10}$. Assuming that $\bar{H}$ denotes the current average adjustment value, where $\bar{H}^r = \frac{1}{n}\sum_{j=1}^{n} H_j^r$, then the factor σ can be calculated through

$$\sigma = \max\left(\frac{1}{10}, \frac{1}{2} - \bar{\mathcal{H}}^{r,trunc} \cdot \frac{1/2}{\phi(100ms)}\right)$$

with $\bar{\mathcal{H}}^{r,trunc} = \min(\max(\bar{H}^r, \phi(-100ms)), \phi(100ms))$.

4.3 Round Number Synchronization

The round number synchronization is an important aspect and enables a node to get a local view of the global clock. As a result all nodes are synchronized, so that in each period they always have the same tick number. This is a prerequisite for the later introduced TTP.

In order to incorporate the existence of erroneous nodes, which always distribute an incorrect tick and different information to each node, we extended our fault hypothesis and thus introduced an improved round number synchronization. Because it is not easy to implement a distributed algorithm which is able to identify such a node without too much message exchange, we constrained our erroneous node to distribute incorrect ticks which are in the range of 0 and a maximum tick number $\mathfrak{t}_{max}$.

The main principle of the round number synchronization is based on a tick-offset variable, declared by $\mathfrak{t}_{offset}$. Therefore, a node j has to transmit the current tick number defined by $\mathfrak{t}^j$, and the current tick-offset $\mathfrak{t}^j_{offset}$ in each period. So if an erroneous node transmits a random tick number and a tick-offset which in sum are smaller than an upper bound $\mathfrak{t}_{max}$, then the receiving nodes are able to determine that there is something amiss and thus ignore this information.

The detailed detection algorithm is described in the Pseudocode 5. It should be noted that the tick-offset variable $\mathfrak{t}^j_{offset}$ and the current tick-number $\mathfrak{t}^j$ of node j are initially set to 0. Further after each period, $\mathfrak{t}^j$ is incremented by one until a predefined end-of-round number $\mathfrak{t}_{eor}$ is reached. This number is determined by the TTP. If $\mathfrak{t}^j$ equals $\mathfrak{t}_{eor}$, then $\mathfrak{t}^j$ will be reset to 0. To ensure that the algorithm always converges, a node only adapts its current tick number to the greatest one from all received neighboring nodes. Thus, after some time all nodes will have the same tick-number in each period and therefore will not correct their ticks anymore.

It should be noted that this synchronization algorithm does not imply a real fault tolerance, because if there exist a byzantine node, then the information

Algorithm 5 Round number synchronization.

1: $new\_current\_tick = \mathfrak{t}^j$
2: $new\_tick\_offset = \mathfrak{t}^j_{offset}$
3: **for** each received synchronization message **do**
4: rx_msg =next received message
5: **if** $rx\_msg.sync\_state = synchronized$ **then**
6: **if** $rx\_msg.current\_tick + rx\_msg.tick\_offset > new\_current\_tick + new\_tick\_offset$ **then**
7: $new\_current\_tick = rx\_msg.current\_tick$
8: $new\_tick\_offset = rx\_msg.tick\_offset$
9: **end if**
10: **end if**
11: **end for**
12:
13: **if** $new\_current\_tick + new\_tick\_offset > \mathfrak{t}^j + \mathfrak{t}^j_{offset}$ **then**
14: $\mathfrak{t}^j = new\_current\_tick$
15: $\mathfrak{t}^j_{offset} = new\_tick\_offset + new\_current\_tick$
16: **end if**
17:
18: $\mathfrak{t}^j = \mathfrak{t}^j + 1$
19:
20: **if** $\mathfrak{t}^j \geq \mathfrak{t}_{eor}$ **then**
21: $\mathfrak{t}^j = 0$
22: **end if**

of a synchronization messages of such a node is usually not bounded. So this algorithm only considers erroneous nodes. But this algorithm also works in the presence of any network topology, even in asymmetric multi-hop networks.

4.4 Energy Awareness

The energy consumption is an important quality characteristic of each communication protocol used in sensor networks. Sometimes often more than 50 percent of energy is used for idle listening [YHE04]. Therefore, it is necessary to reduce the major energy sources. Some MAC protocols have already incorporated such a concept (e.g. S-MAC, T-MAC, etc.). However, we assume that the underlying MAC layer is only responsible for the medium access control and not for energy improvements. For this reason, we assign the tasks for energy reduction to the upper layers. In the next paragraphs we introduce such a mechanism which is based on the above described biologically inspired

synchronization approach.

In order to reduce power consumption caused by idle listening, it is necessary to turn off the transceiver module as much as possible. In literature a protocol based on such a scheme is also called to be a duty-cycle protocol. In such protocols a node becomes dormant most of the time and only wakes up for a short time if it is necessary. Such a wake up event could be the exchange of synchronization messages. The duty-cycle is determined to be the ratio between the duration used for listening to the medium and the duration of the complete period. Figure 4.4 demonstrates the principle of duty-cycling with respect to the imprecision, bounded by the synchronization window w. This demands that the receiver module must be enabled before any transmission is started. To guarantee this behaviour, the difference of the point in time when the receiver is enabled and the first transmission may start should be greater than the predefined synchronization window w. A good way is to choose a time difference which equals twice the synchronization window. This should also involve the fact that the receiver requires some milliseconds for the startup phase. According to Figure 4.1, the duty-cycle equals $\frac{T_{MaxOffset}+2 \cdot w}{T}$.

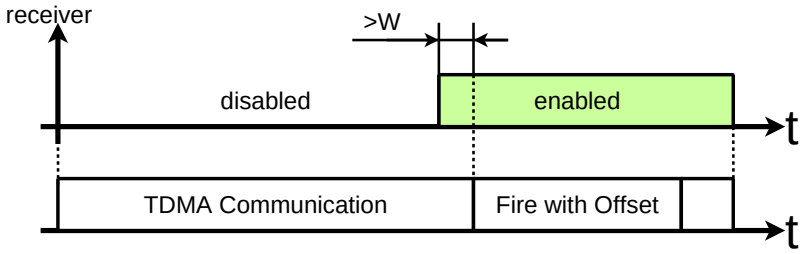


Figure 4.4: The receiver module must be enabled prior the first possible transmission.

4.4.1 Energy Efficiency through Time-Triggered Approach

Until now we have only discussed the synchronization approach to establish a common notion of time. This does not take the data exchange between the nodes into account. For this reason, a time-triggered system based on this time notion is introduced. Such a system takes advantage of the *a priori* known instants of transmission events. Given a sufficient precision of this global time, the sending and receiving units can be turned off for the time, where no communication will take place. In contrast to standard protocols based on idle listening, this approach improves power consumption by reducing the idle listening time. However, similar to the above described low duty-cycle scheme, a node has to enable the receiver unit some time before it expects a data

transmission. This avoids the problem caused by the bounded imprecision in synchronization. For instance, if a node enables the receiver unit at the same point in time when the sender node is expected to transmit, then the enable event would highly likely occur after the transmission start. This originates from the fact that the precision of every synchronization strategy is limited due to unpredictable environmental behaviour. Therefore, the node's local view of the global time with respect to the real-time is temporally varying among each other. If the duration between the early enabling of the receiver unit and the sender's transmission start is greater than the bounded imprecision determined by the synchronization window w, then a node will always be able to receive the data. A good value for this duration is $2 \cdot w$.

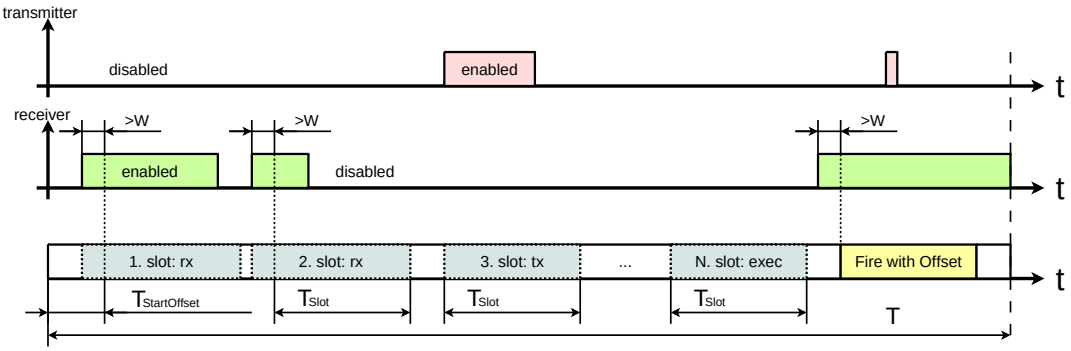


Figure 4.5: TDMA communication during a period.

Figure 4.5 shows the time diagram of a time-triggered approach for a single node. Therein, a period is subdivided into several slots whereas each slot corresponds to either a *receiving slot*, a *sending slot*, an *execution slot*, or an *idle slot*. Concerning the energy awareness, the most important slots are the receiving slots, because they determine how much energy is spent on listening and receiving and thus degrade the duty-cycle. For this reason, it is important to introduce only as many receiving slots as necessary. In the diagram, the first and the second slot are assigned to be receiving slots. Note that the active time for the receiver unit differs between these slots. This comes from the automatic deactivation after the receiver has recognized the end of a transmission.

5 Simulation based on JProwler

In order to accelerate the development process, we decided to simulate our algorithms with a probabilistic wireless sensor network simulator called JProwler. JProwler has been developed by the Institute of Software Integrated Systems at the University of Vanderbilt and is basically configured to simulate the behaviour of Berkeley Motes running TinyOS. For this reason, JProwler also provides the simulation of the standard MAC protocol used in TinyOs. JProwler is widely used in the simulation of wireless sensor networks and originally supports a simple GUI and several simulation models. It is a Java version of Prowler which is used for verifying and analyzing communication protocols of ad-hoc wireless sensor networks. Details about Prowler can be found in [SVML03]. It further provides the opportunity to simulate two different radio models which are the Gaussian Radio Model and the Raileigh Radio Model. In our simulations we always applied the Gaussian Radio Model, because this model is more accurate if the nodes are mainly static. The sources and some documentations can be found at the ISIS-Homepage [1]. Because JProwler is an open source program written in Java, it offers an easy way to modify and extend the sources to meet different needs. This is very helpful to adjust the program to different simulation scenarios. For instance, to be able to reasonably test the clock drift algorithm, it was necessary to extend the simulator's assumption coverage about the behaviour of the oscillator technology used in a virtual nodes. Because the drift mainly depends on the temperature and on the underlying oscillator, the extended simulator is also able to simulate the temperature dependence from several crystal cuts and RC based oscillators. All theses enhancements offers a realistic development environment and further reduces the time to results. In comparison with the testbed experiments, it has been shown that our enhanced simulator delivered almost the same results.

5.1 Simulated Clock Drift Fault Injection

The clock drift fault injection is based on the formula described in Section 2.3 and therefore depends on the simulated ambient temperature. The temperature

[1] ISIS, Institute For Software Integrated Systems: http://www.isis.vanderbilt.edu/Projects/nest/jprowler

range was chosen to be between −40 ℃ and 85 ℃. This should cover the natural temperature appearance. Additionally, the oscillator model of every virtual node can also have an initial clock drift which is independent of the temperature. The initial clock drift should simulate the frequency variations due to fabrication inaccuracies. Beside the clock stability, this is also a major quality criterion of an oscillator.

5.2 Advanced JProwler GUI

The Graphical User Interface implemented in the original JProwler simulator only contains a simple output window which does not allow any user interactions. However, in order to visualize the influence of several parameter choices, it is necessary that the GUI has also some dialogs which enables the user to change the important JProwler-specific or also application-specific parameters on-line. For this reason, several dialogs have been implemented and are described in the next subsections.

5.2.1 The Basic JProwler Setup Dialog

The parameters in the basic JProwler setup dialog are responsible for the complete behaviour of the communication and corresponds to the physical layer of a real node. These variables are initially set to simulate the Berkely's ZigBee2-nodes based on TinyOS. However, as later described we are using Atmel's ZigBee nodes which contain the Atmel specific IEEE 802.15.4 MAC-stack. For this reason, these parameters are modified to simulate the communication delay from the ZigBee nodes.

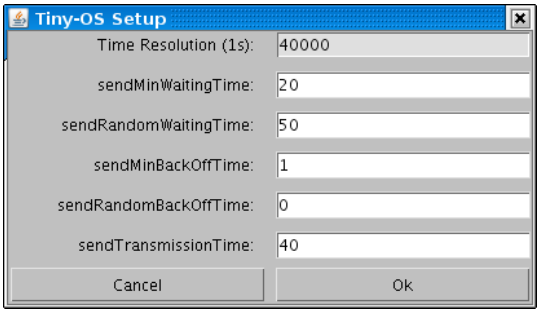


Figure 5.1: The basic setup dialog from JProwler.

Figure 5.1 shows a dialog which contains the parameter choice for the Atmel's ZigBee nodes. The different variables are described below:

Time resolution: This variable has the most important impact on the simulator behaviour, because it determines the amount of discrete simulator steps which should be executed per second. This parameter has an influence on the speed and precision of the results. The standard value is 40000 which corresponds to the 38.4 kbps speed. Considering the files from JProwler, this variable is defined in "*Simulator.java*". It should be noted that every time and time intervals in the simulator is represented using this resolution. For this reason, the value is only displayed in the dialog and thus does not allow any on-line modification.

Minimal send waiting time: The *"sendMinWaitingTime"* variable defines the constant component of the time spent waiting before the start of a transmission. Considering the files from JProwler, this variable is defined in "*ZigBee2Node.java*" and also depends on the time resolution. For instance, if the time resolution is 40000, then a value of 40 corresponds to 1 ms.

Random send waiting time: This parameter defines the varying component of the time spent waiting before a transmission due to the CDMA scheme. Considering the files from JProwler, this variable is defined in "*ZigBee2Node.java*". The value depends on the time resolution described above.

Minimal backoff time: Defines the constant component of the backoff time. Considering the files from JProwler, this variable is defined in "*ZigBee2Node.java*" and depends on the time resolution above.

Random backoff time: Defines the varying component of the backoff time. Considering the files from JProwler, this variable is defined in "*ZigBee2Node.java*" and again depends on the simulator's main time resolution.

Transmission time: This parameter defines the time of one transmission. In reality the value depends on the amount of transmitted data. However, a fixed transmission time is good enough for the simulation and further reduces the software complexity. Considering the files from JProwler, this variable is declared in "*ZigBee2Node.java*" and depends on the time resolution of the simulator.

5.2.2 The Node Configuration Dialog

The node configuration dialog enables the on-line modification of the basic node parameters. These are the three dimensional coordinates for the position and the maximum radio strength. Additional paramaters due to the enhanced

simulator are the initial clock drift, the individual ambient temperature, the underlying oscillator technology and a boolean variable to determine if the node should depend on the configured temperature. Figure 5.2 shows an example of such a dialog. Therein, the button "crystal oscillator" opens a new dialog for the oscillator configuration. The parameters in this dialog are individual for each node.

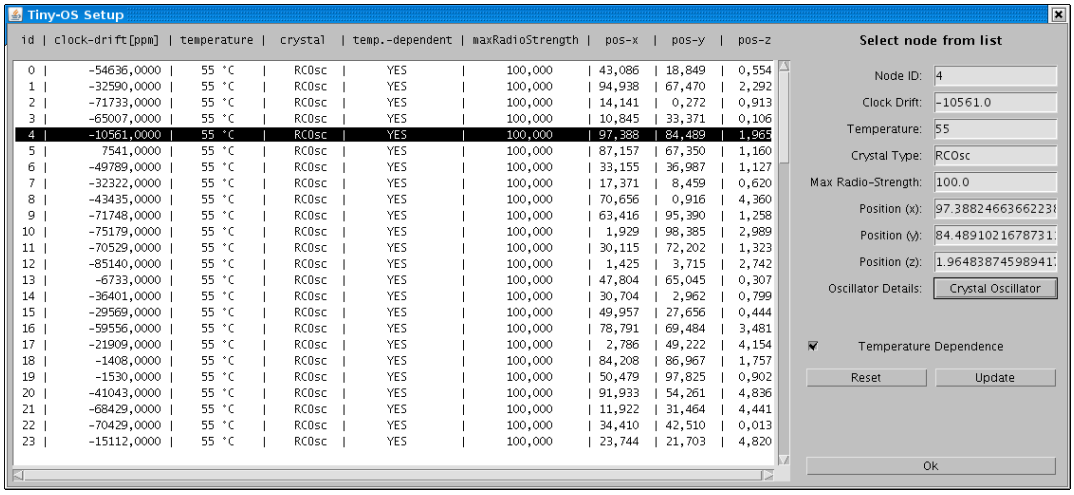


Figure 5.2: The on-line configuration of a virtual node.

The Oscillator Selection Dialog

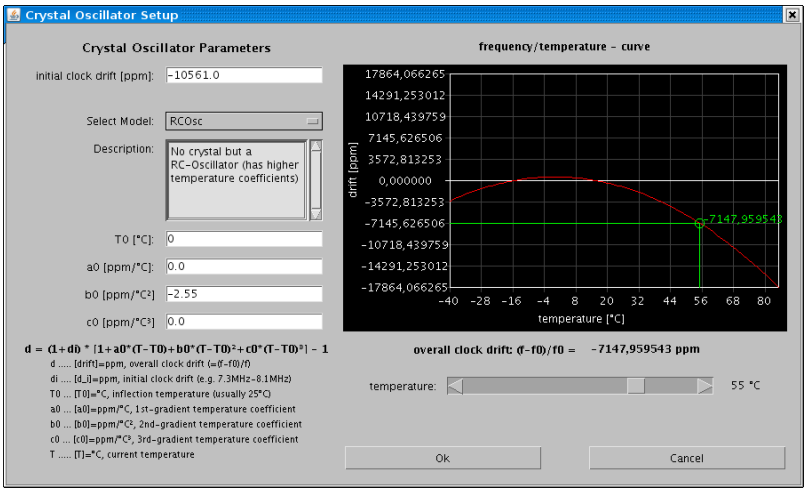


Figure 5.3: Every virtual node is based on an underlying oscillator model.

This dialog is demonstrated in Figure 5.3 and can be used to modify the underlying oscillator technology for each node. Further the dialog makes it

possible to change the temperature coefficients and the initial clock drift during the simulation. If the initial clock drift is very big, then the temperature will have a little influence on the oscillator frequency. This is especially true for the crystal cuts whereas the RC-oscillator usually depends more strongly on the temperature.

In the following we describe the characteristics of some interesting high frequency crystal cuts (e.g. AT, BT, SC), several low frequency cuts (e.g. NT, XY, H) and the RC-oscillator in a short. The temperature coefficients for all these oscillators are chosen randomly within a short range based on the tutorial from [Lic91]. The user has the possibility to change these parameters either statically in the application or during the simulation within the oscillator selection dialog.

AT cut: This crystal cut is the most commonly used for high frequency oscillators and usually has a cubic frequency vs. temperature curve caused by the dominant third order temperature coefficient of about $-0.0013 ppm/℃^2$. The turnover temperature is located at about 25℃. Further such crystals can be used for frequencies ranging from 500kHz up to 40MHz. Figure 5.4 demonstrates the curve progression of a typical AT cut.

BT cut: The BT cut crystal is again used for high frequencies. In contrast to the AT cut family, these crystals have a poorer temperature characteristic which is similar to a downward parabolic curve. This comes from the dominant second order temperature coefficient of about $-0.025 ppm/℃^2$. However, the BT cut allows a higher frequency of up to 50MHz. Figure 5.5 shows the typical frequency vs. temperature curve of such a crystal. Therein, the frequency is most stable around the room temperature, but slows down if the temperature is getting higher or lower.

SC (stress compensated) cut: The frequency vs. temperature curve of a SC cut crystal seems to be linear due to the higher first order temperature coefficient. The curve is visualized in Figure 5.6. This crystal cut is generally more frequency stable to temperature variations and has a first order temperature coefficient of about $0.01 ppm/℃$ to $0.04 ppm/℃$. The turnover point is usually situated around 90 ℃.

NT cut: The frequency vs. temperature curve of an NT cut crystal is also based on a dominant second order temperature coefficient of approximately $-0.036 ppm/℃^2$ and a turnover temperature within $[15℃, 80℃]$. This crystal type is used for frequencies ranging from 3kHz to 85kHz. Figure 5.7 illustrates such a curve with a turnover point of about 80℃.

XY cut: Due to the smaller size, the XY cut is often used for real-time clocks with frequencies in the 3 kHz to 85 kHz range. Crystals based on this cut

have a second order temperature coefficient of about $-0.033 ppm/℃^2$ and a turnover temperature of 25℃. Therefore, the behaviour is similar to the BT cut crystals and the frequency vs. temperature curve demonstrated in Figure 5.5 also equals an XY cut.

H cut: In contrast to other low frequency crystals, the H cut has a dominant negative first order temperature coefficient and is usually used for wide band filters with a frequency range from approximately 8kHz to 130kHz. For this reason, the frequency decreases linearly with an increase in temperature. The first order temperature coefficient is within $[-8 \cdot 10^{-6}/℃, -16 \cdot 10^{-6}/℃]$ whereas the turnover temperature can vary from 16℃ to 80℃. This behaviour is illustrated in Figure 5.8.

RC oscillator: The RC oscillators are widely used for low-cost applications and therefore are important with respect to this thesis. Contrary to several crystal cuts, the RC oscillator has a very big temperature influence. The temperature characteristics generally equals a parabolic curve similar to the BT cuts. The turnover temperature is not situated at the ambient temperature and usually has a greater frequency at this point than the nominal frequency. Beside the strong temperature dependence, the imprecision in frequency calibration for cheap RC oscillators is also the reason why it is difficult to synchronize nodes based on such a technology. The initial drift of RC oscillators is bigger and more varying than that from the crystal cuts.

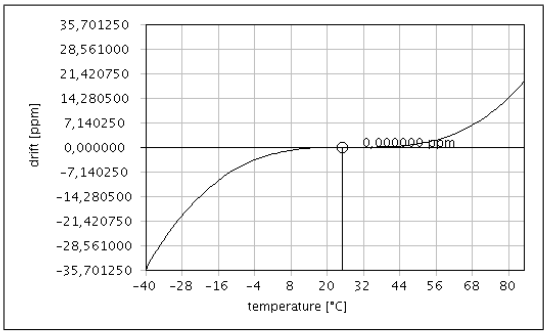


Figure 5.4: The typical cubic curve of an AT cut crystal based oscillator.

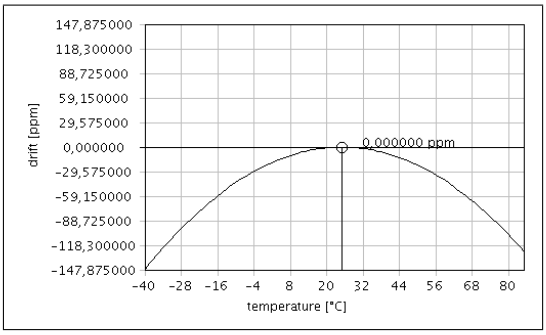


Figure 5.5: The typical parabolic curve of a BT cut crystal based oscillator.

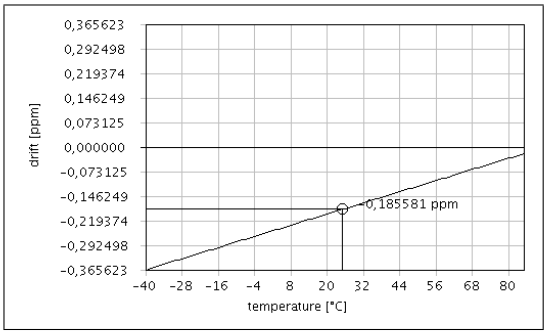


Figure 5.6: The typical curve of an SC cut crystal based oscillator.

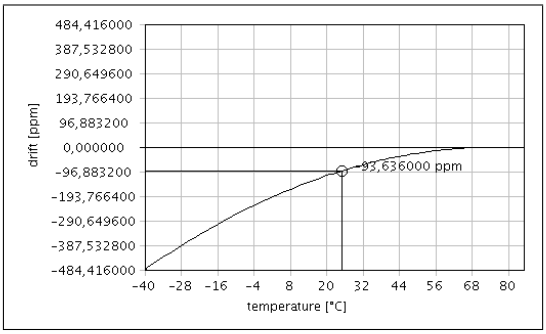


Figure 5.7: The typical curve of an NT cut crystal based oscillator.

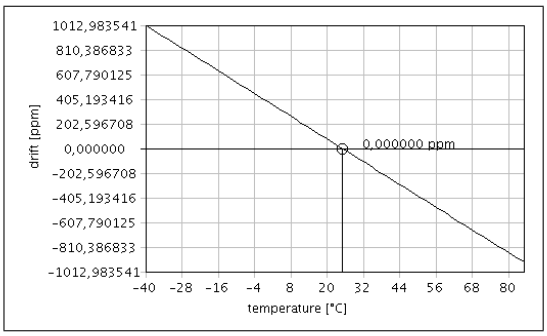


Figure 5.8: The typical linear curve of a H cut crystal.

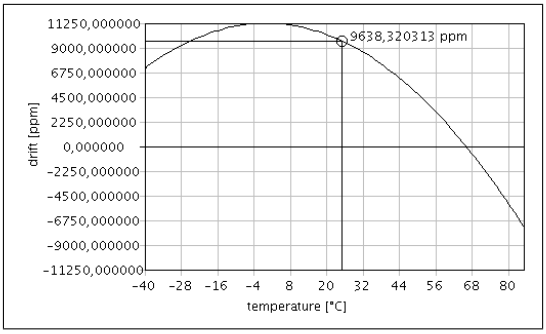


Figure 5.9: The typical curve of an RC oscillator.

5.3 Simulation of Virtual Fireflies

The simulation of the fireflies is done via the enhanced JProwler simulator with an additional Firefly specific graphical user interface. This GUI includes several windows for debugging. For instance, the interval drift window illustrates the current overall average interval and the deviation of all fireflies. Another window displays several debug parameters. The main simulator window demonstrates the communication between the virtual nodes and further includes a phase bar which visualizes the current phase state of all fireflies. Figure 5.10 shows a screenshot of the complete enhanced Firefly simulator.

5.3.1 The Network Topology Window

The network topology window is the main window of the simulator and visualizes the location of the nodes or fireflies. An arrow displays a directed

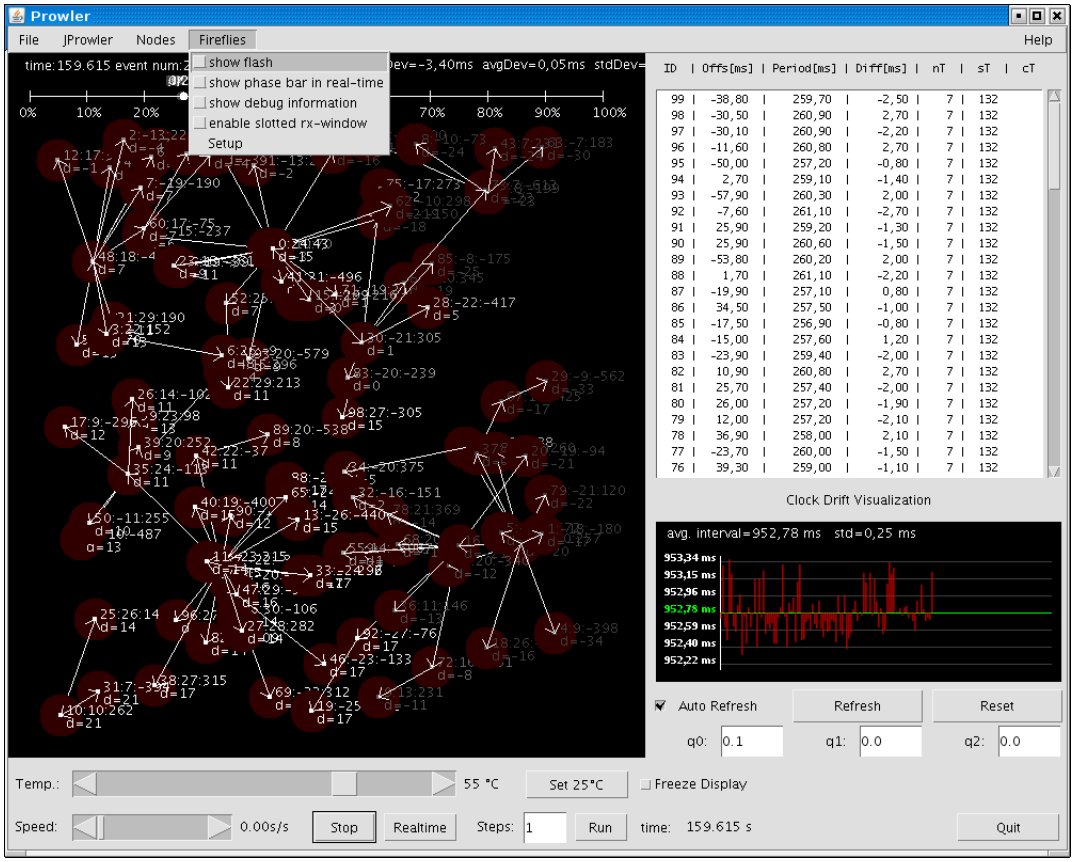


Figure 5.10: The complete GUI-window of the Firefly-specific enhanced JProwler simulator.

transmission of a message during a period. Figure 5.11 demonstrates an example of a network topology window including 100 randomly distributed nodes.

Each Firefly is pictured with a filled bar and several parameters as depicted in Figure 5.12. The first parameter indicates the node-ID whereas the second parameter represents the maximum deviation an individual node has measured with respect to the neighboring nodes during the last period. The time resolution is based on the variable *PHASE_FACTOR* and is initially set to 10000 units per second. Therefore, a maximum deviation of 10 corresponds to 1 ms. This granularity is chosen because it is similar to the one used in the real world application and thus should make the simulation more realistic. Next the third parameter displays the adjustment value which is used for the clock drift calibration algorithm. Finally the parameter d in the second line illustrates the phase deviation with respect to the overall average phase-state in the same resolution as the other parameters. This value is also used to shade the color of all virtual nodes in dependence of the deviation strength. The refresh of the parameters, the color shading, and the phase-bar is done periodically with

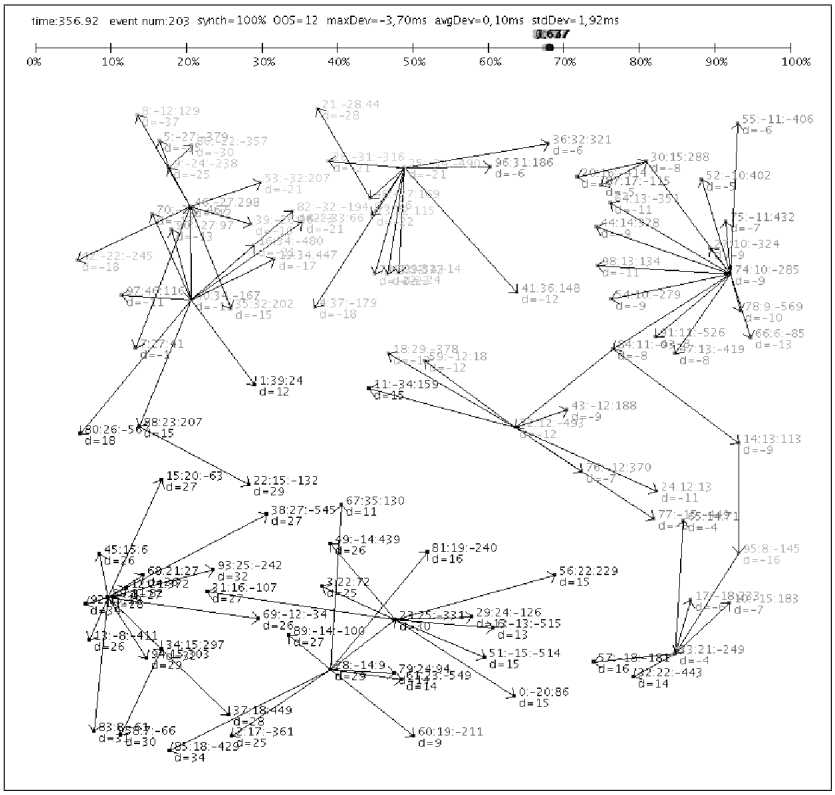


Figure 5.11: The topology window visualizes the simulated network topology whereas the current communication exchange is displayed via arrows.

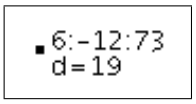


Figure 5.12: The visualized parameters of a virtual Firefly.

a period duration which equals the calculated average interval over all nodes. This gives the user the impression that the phase state keeps constant during each period and therefore enables a better debugging.

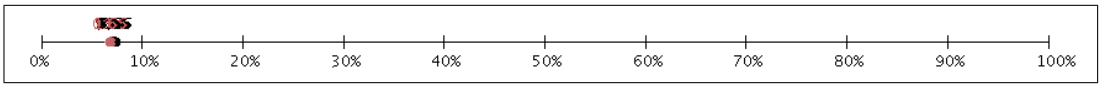


Figure 5.13: The phase-bar is situated within the main simulator output window.

The phase bar is shown in Figure 5.13. Therein, each circle corresponds to the current phase state of a Firefly whereas 0 percent means the beginning of a new period and 100 percent equals the end of an interval. As mentioned above the phase bar is a periodic snapshot with a refresh interval which equals the

overall average interval. The user has also the option to enable a continuously updated view.

5.3.2 The Interval Drift Visualization Window

This window is used to visualize the interval deviation due to the clock drift among each node. The deviations and further the standard deviation is calculated with respect to the average interval. The standard deviation is also a quality criterion for the drift calibration algorithm and is also used to compare the results based on different parameter choices. Figure 5.14 displays such a window for a grouped multi-hop topology which is explained later in the Chapter 6.

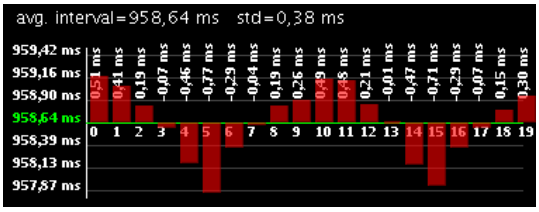


Figure 5.14: With respect to real-time, every virtual node has a different interval duration.

5.3.3 Important Firefly Parameters

The implementation of the naturally inspired synchronization algorithm has several important parameters which determine the behaviour of the fireflies. The most important ones are described below. The Firefly configuration dialog shown in Figure 5.15 enables the user to modify these variables during runtime.

Phase factor: Based on the phase jump function defined in Equation 4.7 the phase is originally assumed to be in the range from 0 to 1. As a result the implemented phase variables have to be represented as floating point variables. This is unacceptable for the use in low-cost controllers. For this reason, we decided to introduce a constant multiplier which quasi scales the phase to the range from 0 to the predefined phase factor whereas the position after the decimal point is truncated. Because the truncation causes a lost in the phase precision, the phase factor has to be very great. To get more realistic simulation results, the phase factor was chosen to have the same dimension as the timer granularity of the real world implementation which is about 32000 ticks per second. Therefore, the phase factor is defined to be 10000.

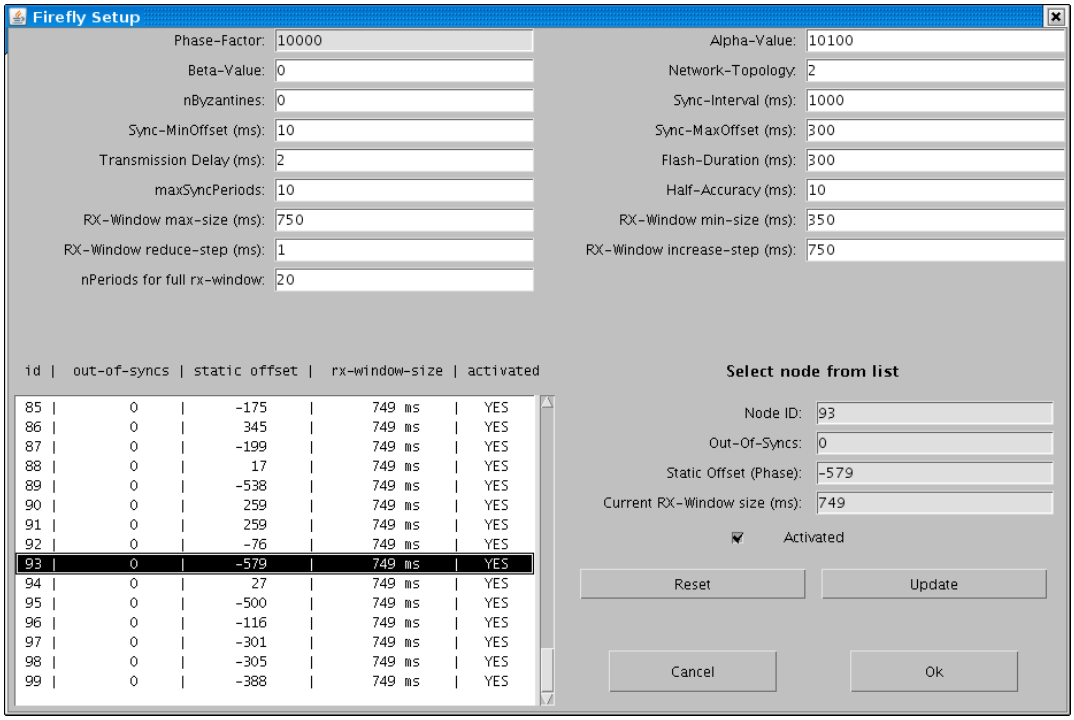


Figure 5.15: Every Firefly can be configured individually.

Alpha value: This variable defines the coupling strength α of the phase jump function defined in Equation 4.7. The upper and lower bound of this parameter is described in Chapter 4. Because of the already described floating point problem, this parameter is scaled with the phase factor and is originally set to be 10100 which corresponds to $\alpha = 1.01$.

Beta value: Assuming Equation 4.4, this variable corresponds to the parameter β and usually has only a little influence on the phase jump function. Further if the dissipation factor b is chosen to be in the order of 5 or higher, then this parameter is very small and thus negligible. For instance, a dissipation factor of $b = 5$ and a pulse strength of $\varepsilon = 1.005$ will result in $\beta = 0.0001$. Therefore, after scaling this value with the phase factor, this variable should then be set to 1. However, several experiments have also shown that the synchronization results are better if this variable is set to 0.

Network topology: The number of this variable defines the simulated network topology. In the current implementation 10 different topologies are supported. These network configurations are described in detail in Table 5.1.

Topology number	Description

0	Two nodes are randomly distributed within an all-to-all communication network. Every node has also an initial random clock drift.
1	100 nodes with an initial clock drift are randomly distributed within an all-to-all communication topology.
2	100 nodes with an initial clock drift are randomly distributed over a field with a size of 100x100 metres. Because the transmission range is limited to about 30 metres, the topology highly likely includes several multi-hop scenarios. This configuration may be used for simulations which most likely occur in the real world.
3	100 nodes with an initial clock drift are distributed within a regular grid communication topology. Therefore, a node can directly exchange messages with at most four other neighboring nodes. A high diameter of this configuration can be used to test the limits of a synchronization algorithm.
4	Ten nodes with an initial clock drift are ordered in a chain and therefore simulate a multi-hop topology.
5	This configuration contains two clusters comprised of 50 nodes which are respectively synchronized but there is no communication between these clusters. After 100 seconds two more nodes are added to the network. These nodes have a bidirectional communication channel to both clusters.
6	20 nodes with an initial clock drift are randomly distributed within an all-to-all communication topology.
7	10 nodes with an initial clock drift are randomly distributed over a field with a size of 1x1 metres. However, all nodes can only hear node 0 whereas node 0 receives nothing. This communication topology should demonstrate how nodes can be synchronized if there exist unidirectional communication channels.
8	Three nodes with an initial clock drift are distributed over a 1x1 metres field. Node 0 and node 1 have a bidirectional communication channel, but messages from node 2 can only be received from node 1. This communication topology enables the experimentation with respect to unidirectional communication patterns.
9	This topology type simulates a grouped multi-hop network and is similar to the standard multi-hop topology, but with the addition that each node is replaced with clusters. The nodes in a cluster are based on an all-to-all communication pattern and have a standard group size of 10 nodes.

Table 5.1: Definition of the supported network topologies.

Number of byzantine nodes: Defines the number of byzantine nodes which should be incorporated in the synchronization algorithm. A byzantine node is a node which always transmits different incorrect messages including the synchronization mode, the phase-state and several other parameters. If this variable is greater than zero, then this parameter defines the number of nodes which should be ignored.

Synchronization interval ($SYNC\_INTERVAL\_ms$): This variable defines the periodic time for the synchronization in milliseconds. The default value is 1000ms. A greater value will reduce the duty-cycle and also increases the time to synchronization.

Minimum firing offset ($SYNC\_MIN\_OFFSET\_ms$): This term refers to the $T_{MinOffset}$ parameter defined in Section 4.1.1 and is depicted in Figure 4.1. Therein, the minimum firing offset declares the offset with respect to the end of the interval where transmission of synchronization messages must not occur. The variable is defined in milliseconds and is initially set to 100ms.

Maximum firing offset ($SYNC\_MAX\_OFFSET\_ms$): This term refers to the $T_{MaxOffset}$ parameter defined in Section 4.1.1 and is also depicted in Figure 4.1. Therein, the maximum firing offset declares the offset with respect to the end of the interval. Transmission of synchronization messages may only occur between the maximum firing offset and the minimum firing offset. In order to get acceptable results in an all-to-all topology comprising 100 nodes, this variable is initially set to 400ms.

Transmission delay: The transmission delay corresponds to the time between the start of a transmission at the source node and after the last bit is received at the sink node. Whereas the propagation delay in wireless technology is usually negligible, the transmission delay is often in the order of milliseconds. This depends also on the amount of data and further on the data rate. In sensor network applications this parameter is generally lower than one milliseconds and therefore is initially set to $375\mu s$. This parameter is also the reason why the Maximum firing offset strongly depends on the number of nodes which are configured in an all-to-all communication manner.

Flash duration: The flash duration is only used in the Firefly visualization of the GUI and has no direct impact on the results. The default value is 300ms which is great enough to notice the flash of the fireflies.

Maximum number of synchronization periods: This parameter is very important and has a deep impact on the time to synchronization, because it

defines the number of periods a node must keep in a predefined precision window to be allowed to change to the sync-state. In detail, a node compares the maximum absolute deviation with the synchronization window and thus is able to decide if the precision is good enough. If the precision is outside this window, then the node resets a period counter. This variable is initially set to 10 periods.

Synchronization window ($HALF\_ACCURACY\_ms$): The synchronization window defines the upper bound of the synchronization. If a node notices an absolute synchronization deviation which is outside the window, then the node has to change to the unsync-state. The worst possible precision is thus twice the synchronization window. The default value for this parameter is 10 ms. A greater value may reduce the time to synchronization but also degrades the precision.

Maximum RX-window size: The RX-window defines the time used by the receiver unit for listening to the medium. The maximum rx-window corresponds to the maximum possible time where all nodes are still possible to synchronize. The initial value for this variable is $(SYNC\_INTERVAL\_ms - (SYNC\_MAX\_OFFSET\_ms - SYNC\_MIN\_OFFSET\_ms) + 4 * HALF\_ACCURACY\_ms)$.

Minimum RX-window size: The minimum RX-window size is an important parameter which determines the duty-cycle after a network achieved synchronicity. In order to receiver every synchronization message during the firing interval, the RX-window must not be smaller than the duration between the maximum firing offset and the minimum firing offset. So if the synchronization window is incorporated into the calculation, this variable should equal $(SYNC\_MAX\_OFFSET\_ms - SYNC\_MIN\_OFFSET\_ms + 6 \cdot HALF\_ACCURACY\_ms)$.

RX-window reduction step: If a node enters the synchronization state, then there is no need to listen to the medium outside the firing interval. To avoid the problem of idle listening a node continuously reduces the RX-window size every period due to this parameter. Note that the window size can not not be smaller than the minimum RX-window size. The default reduction step is 1ms per period.

RX-window increase step: In contrast to the window reduction step, the window increase step is used if a node recognizes an out of sync event. As long as the node keeps unsynchronized, the RX-window size will be increased due to this parameter until it reaches the maximum RX-window size. Usually if a node changes to the unsync-state, it can be assumed that other nodes are also unsynchronized. Therefore, the parameter was initially set to $RX\_MAX\_WINDOW\_SIZE\_ms$ and thus if a node becomes

unsynchronized, the RX-window size will automatically be enlarged to the maximum RX-window size.

Number of periods for full RX-window: To support graceful scalability, it is necessary to incorporate the adding of new nodes to an already synchronized network. The nodes in an already synchronized network usually have reduced the RX-window to the minimum RX-window size and therefore are not possible to receive synchronization messages outside this interval. As a result if the new node has a phase state which is completely different to the synchronized nodes, then the new node will either never get synchronized or the time to synchronization will strongly increase. The initial value is set to 20 periods. A greater value reduces the power consumption but may increas the time to synchronization of the new node and therefore should be carefully chosen.

6 Simulation Experiments

The simulation results discussed in this chapter should give an overview of the achievable quality of the naturally inspired Firefly synchronization. For this reason, several network topologies have been developed and simulated. The results are compared due to different parameter choices like the coupling factor α, and the number of nodes in the network. To be able to compare the results in a useful manner, the results are based on the same evaluation metrics. The next section describes how the measurement results are calculated.

6.1 Evaluation Metrics

In order to compare the simulation results with the outcomes from [WATP+05], the evaluation metrics are similar to that one described in the work from Nagpal. Therefore, the two important evaluation metrics are the amount of time until the system achieves synchronicity, and the quality of precision.

Time To Sync: This metric defines the time until all nodes have entered the synchronization state whereas the time to sync is determined by two parameters. These are the synchronization window w and the number of required periods where a node keeps within this window. In the following we call the amount of required periods *synchronization periods* and is usually set to 10. A node only enters the sync-state, if the maximum absolute deviation with respect to the other nodes is within the synchronization window for 10 out of the last 11 firing iterations.

50th and 90th Percentile Group Spread: This metric differs from that one defined in [WATP+05], because we only refer to one group. Therefore, the group spread in the simulation is defined to be the maximum absolute deviation with respect to the average deviation and thus cannot be greater then the synchronization window w. In order to avoid incorrect results due to settling effects during the startup phase, the start of the group spread measurement is postponed against the time to sync t_s and the time the experiments ends t_e. On this account the group spread measurement is performed during the interval $[t_s + \frac{t_e - t_s}{2}, t_e]$. To get a good overview of the results, we decided to plot the median and the 90th percentile

group spread. The results also contain the maximum absolute group spread to determine the mavericks which generally determine the quality of synchronization.

6.2 General Parameter Settings

Several parameter settings are the same for all experiments and are described only once in this section. For instance, every virtual node is based on a virtual RC-oscillator. Because the real nodes have a nominal frequency of 8MHz which can vary from 7.4MHz to 8.2MHz, every virtual node has also a random initial clock drift between $-100000ppm$ and $100000ppm$. The general values of the other parameters are denoted in Table 6.1.

Parameter	**Value**
Oscillator technology	RC-oscillator
Initial clock drift	± 100000
Interval time	1000ms
Maximum firing offset	300ms
Minimum firing offset	10ms
Phase factor	10000
Alpha value	$10100/10000 = 1.01$
Beta value	0
Transmission delay	$375\mu s$
Synchronization window	10ms
Evaluation end	3600 periods

Table 6.1: The general parameter choice used in all simulator experiments.

6.3 Simulation Results

The next sections shows the simulation results in dependence of several network topologies and parameter choices. Every configuration was simulated several times and the worst case results are presented.

6.3.1 All-to-all Topology Results

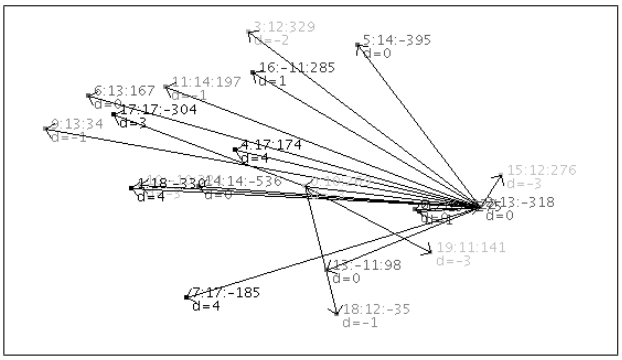
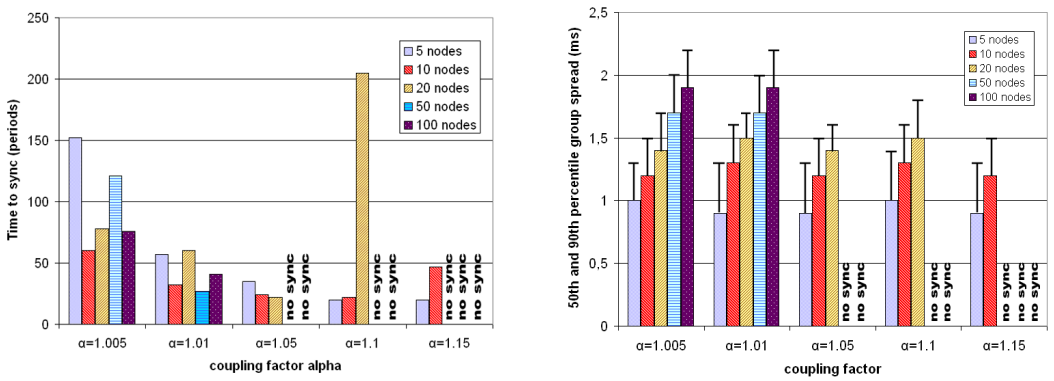


Figure 6.1: A simulation snapshot of an all-to-all topology with 20 nodes.

The all-to-all communication topology is mainly used to measure the quality of the synchronization in dependence of the number of nodes and the coupling factor α. Such a network configuration is also shown in Figure 6.1. Therein, every node is in the transmission range of every other node.



(a) Time to sync diagram (b) Group spread diagram

Figure 6.2: The time to sync and the group spread in dependence of the network size and different coupling factors. The solid bars in (b) represent the 50th percentile group spread, while the error bars correspond to the 90th percentile.

The simulation results based on this topology should give a good overview on the impact of different coupling factors. According to the diagrams in Figure 6.2, the time to sync decreases with an increasing coupling factor α. If the factor is too big, then synchronicity will not be achieved. This effect is due to the upper bound we have already discussed in section 4.1.3. Further it seems that the group spread which corresponds to the synchronization precision is the

same and thus independent of the coupling factor. The high time to sync bar in Figure 6.2(a) with $\alpha = 1.1$ and a number of 20 nodes comes from the fact that the coupling factor was too great.

6.3.2 Multi-hop Topology Results

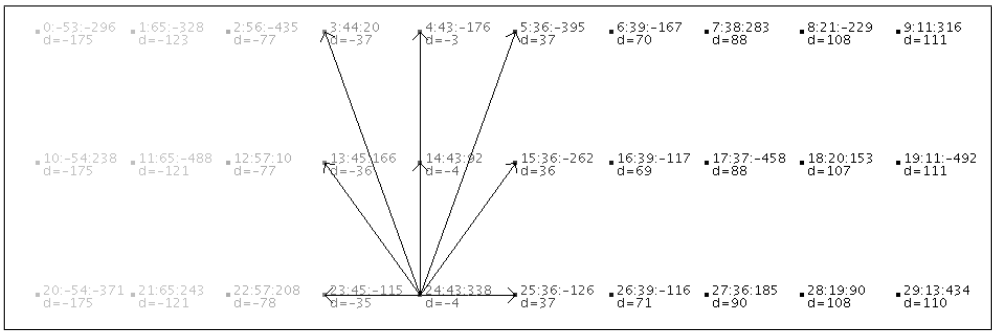
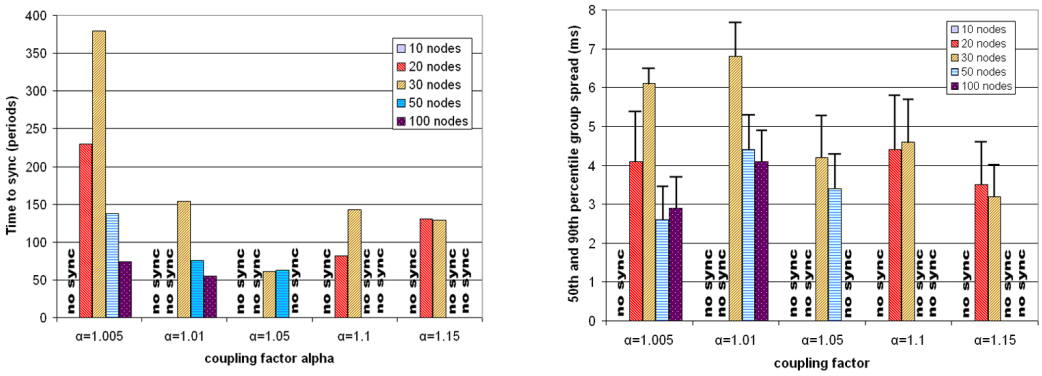


Figure 6.3: A simulation snapshot of a grouped multi-hop topology with a network diameter of 10 and a group size of 3 nodes.

This communication topology is the most important one, because in reality nearly every sensor network is based on a nodes-to-sink communication topology with a communication path consisting several hops. The simplest multi-hop scenario is a network comprising n nodes which are ordered in a chain and can only communicate with the immediate neighbors. We further call the chain size *network diameter*. Such a network with a great network diameter is often very problematic to synchronize, because every hop involves a communication delay which degrades the overall synchronization precision. Our solution is based on grouped multi-hop networks. Therein, the nodes are replaced with clusters comprising several nodes which all have a bidirectional communication path to the immediate neighboring clusters. Note that all grouped multi-hop topologies treated in our experiments have the same network diameter of 10 hops but vary in the cluster/group size. A typical simulation snapshot of a grouped multi-hop network is depicted in Figure 6.3.

The diagrams in Figure 6.4 shows the time to sync and the group spread in dependence of different group/cluster sizes and coupling factors. The network diameter is always the same. The analysis visualized in these diagrams leads to the result that the precision increases with a bigger group size. This effect is caused by the greater information about the interval drift due to the increased number of neighboring nodes. If a node has more neighboring nodes, then the node receives more information about the clock drift and can more precisely calibrate the interval duration which also improves the synchronization precision. However, it is also important to have a preferably small coupling factor.



(a) Time to sync diagram (b) Group spread diagram

Figure 6.4: The time to sync and the group spread in dependence of the cluster size and different coupling factors. Note that the number of nodes must be divided by 10 (network diameter) to get the group size. The solid bars in (b) represent the 50th percentile group spread, while the error bars correspond to the 90th percentile.

On the one hand this increases the time to sync, but on the other hand this also increases the possibility that the network achieves synchronicity. Finally it is difficult to find the best parameter settings for a given multi-hop network, but it is definitely a good choice to have a group size of more than one node. This also increases the dependability and availability of the network.

6.3.3 Regular Grid Topology Results

The regular grid topology is used as an evaluation topology and has only few similarities with real sensor networks. Several experiments concerning this section are based on a squared regular grid topology. We further correspond the network diameter for this topology with the number of border nodes. For instance, a network diameter of 4 corresponds to a regular grid topology with $4 \cdot 4 = 16$ nodes. Such a configuration is also depicted in Figure 6.5. Therein, a node can only communicate with at most four immediate neighbors.

Simulations with a network diameter greater than 4 often did not achieve synchronicity independent of the coupling factor. Therefore, the diagrams in Figure 6.6 only contain the measurement results from a regular grid topology with a network diameter of 4. The diagram in Figure 6.7 leads to the result that the start condition (e.g. start-phase, initial clock drift) is mainly responsible for achieving network synchronicity and the coupling factor has only a little impact on the precision. The random initial drift has a deep impact on the precision, because it seems that the clock drift calibration algorithm does not

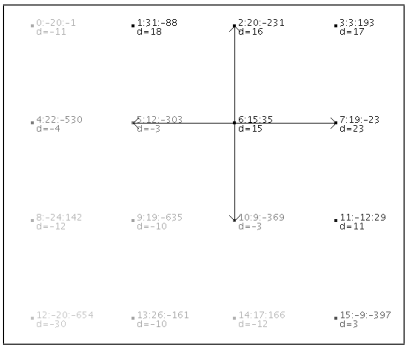
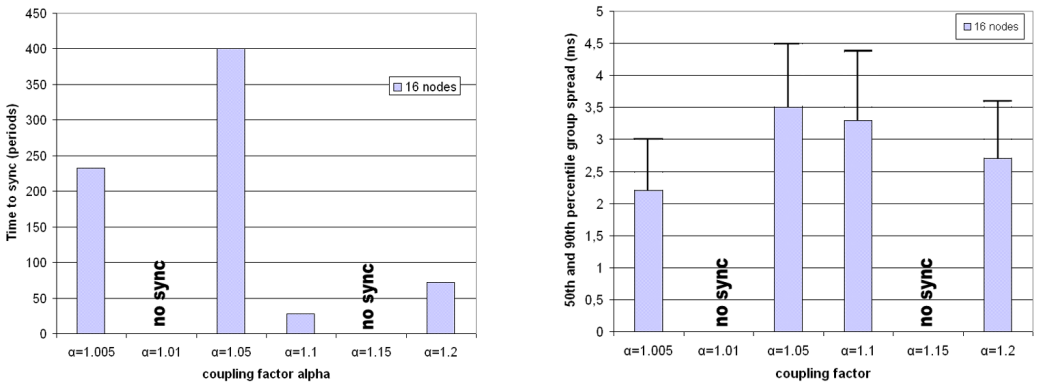


Figure 6.5: A simulation snapshot of a regular-grid topology with 4x4 nodes.

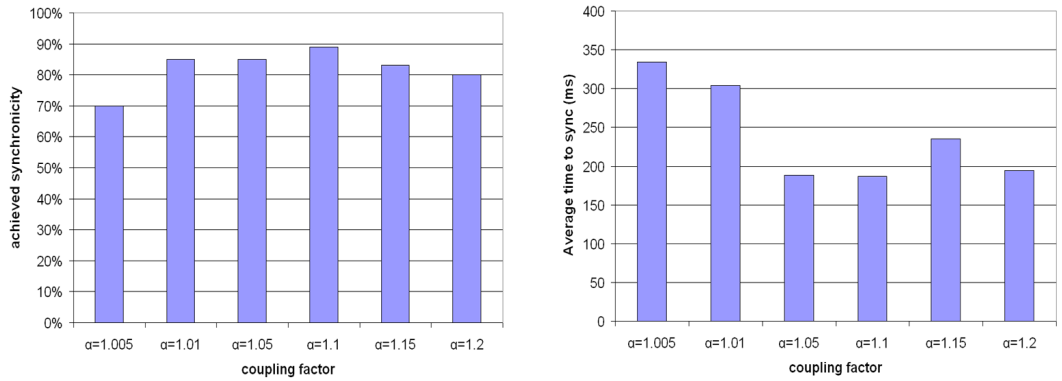
work very well in such a topology. The results may be better if the nodes were replaced by clusters similar to the grouped multi-hop network.



(a) Time to sync diagram (b) Group spread diagram

Figure 6.6: The time to sync and the group spread in dependence of different coupling factors. The solid bars in (b) represent the 50th percentile group spread, while the error bars correspond to the 90th percentile.

The results listed in Table 6.2 show that the maximum absolute deviation (group spread) is often in the vicinity of the synchronization window. As a result it is highly likely that a node becomes out of sync even if this happens after a long time (e.g. $>$ 3600 periods). However, because our simulations are performed with a maximum simulation time of 3600 periods, we observed this behaviour very sparsely. For instance, in several measurements a network achieved synchronicity after about 200-300 periods, but became unstable after 1000 or 2000 periods and, thus, some nodes got unsynchronized and reintegrated within the next few periods. In summary if such a network comprises nodes which have a largely different clock drift, then it becomes difficult to achieve and maintain synchronicity using the given approach.



(a) Achieved synchronicity over 100 simulations

(b) Average time to sync diagram

Figure 6.7: The percentage of achieved network synchronicity out of 100 simulations. A network synchronicity is only achieved if all nodes are synchronized during the last 1000 periods. Figure (b) displays the average time to sync of all simulations which achieved synchronicity.

Parameter choice	Time to sync (periods)	50th-percentile (ms)	90th-percentile (ms)	Maximum deviation (ms)	Standard deviation (ms)
$\alpha = 1.005$	233	2.200	3.000	4.000	0.472
$\alpha = 1.010$	no sync				
$\alpha = 1.050$	400	0.900	4.500	9.000	0.722
$\alpha = 1.100$	28	3.300	4.400	6.600	0.710
$\alpha = 1.150$	no sync				
$\alpha = 1.200$	72	2.700	3.600	5.100	0.631

Table 6.2: Comparison of several parameters in dependence of different coupling factors in a grid topology with 4x4 nodes.

6.3.4 Asynchronous Communication Results

Simple Unidirectional Communication

Other experiments concerns asynchronous communication patterns, especially unidirectional communications. A typical evaluation topology comprises two nodes (node 0 and node 1) whereas node 0 does not receive anything and node 1 receives messages from node 0. Several tests and simulations without the incorporation of different clock drifts lead to the result that the Firefly algorithm does not work well if there exist unidirectional communication patterns. In detail, the phase of node 1 periodically comes into the vicinity of node 0

but deviates more and more over time after the absolute deviation to node 0 is greater than the synchronization window and thus becomes unsynchronized. This behaviour was observed with every coupling factor. As a result a major prerequisite for complete network synchronicity is that there should never exist unidirectional communication paths.

The Influence of Unidirectional Communication

Another experiment concerning unidirectional communication paths is a network topology comprising three nodes which are again numbered serially from 0 to 2. Therein, node 1 has a bidirectional communication path to node 0 but a unidirectional path to node 2. Consequently, node 0 can receive message from both node 0 and node 2, but node 2 does not receive any messages. Instead node 0 is also able to receive messages from node 1. This configuration should allow node 0 and node 1 to get synchronized. However, node 2 may affect node 1 so strong that the node may keeps unsynchronized. We have simulated this behaviour several times without the incorporation of random initial clock drift. The measurements leads to the result that node 2 only becomes for a short time synchronized with the other two nodes whereas the other two nodes always keep in synchronicity. Similar to the first asynchronous topology, we have observed that the phase always drifts apart even if it is synchronized with the other ones. As a result node 2 periodically becomes unsynchronized for a long time. This behaviour was again observed with all possible coupling factors.

Ring Topology Experiments

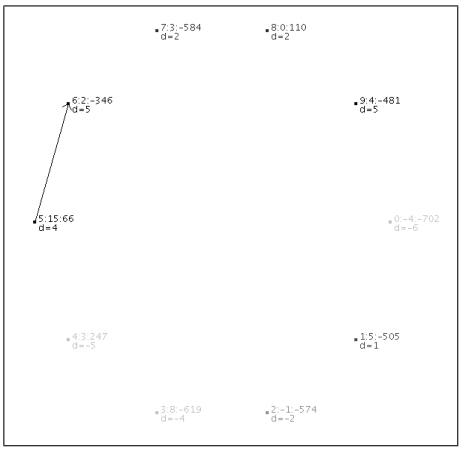


Figure 6.8: A simulation snapshot of a ring topology with 10 nodes and an unidirectional communication path.

Experiments with a ring topology similar to the one pictured in Figure 6.8 have partially disproved the problem of asynchronous communication. The unidirectional ring topology experiments lead to the result that network synchronicity can be reached, even in the presence of asymmetric connections. But the important prerequisite for achieving synchronicity in such a network is that according to the graph theory, for every two nodes v and w, there must exist a closed directed path, with repeated nodes allowed. Note that the cycle length is also an indicator for the convergence time and achievable synchronization precision. Taken together these results indicate that a multi-hop network containing at least two completely different communication paths between two edge nodes will improve the precision even if there exists temporal unidirectional connections.

7 Testbed Environment

The testbed is based on Atmel's demonstration kit ATAVRRZ200 which is described in [Atm06a]. The kit features two component boards: The Display Board and the Remote Controller Board (RCB)s. The Display Board is based on an Atmega128 controller and features an LCD-module. This board also works as a docking station for programming the RCBs. The RCBs therefore are based on an Atmega1281 controller and contain an AT86RF230 (2450 MHz band) radio transceiver.

7.1 The Software Implementation

The implementation of the synchronization algorithm is based on Atmel's 802.15.4 MAC-Stack. Further to be able to test the synchronization, we decided to use a modified version of the already implemented TTP/A protocol. For this reason, the RODL ensures that there will be no collisions if a node wants to transmit data to an other node.

7.1.1 The 802.15.4 MAC-Stack

In order to estimate the achievable precision, we measured several delays caused by the MAC-Stack. We detected that the delay between a send event at the application layer and the instant when the transmission really starts at the physical layer is in the order of milliseconds. The reason for this is, that the events are passed down from the application layer to the physical layer through several sublayers. This is done by storing the messages in a FIFO buffer which is continuously processed. The same delay was also measured at the receiver. If several nodes want to transmitt at the same time, we sometimes measured an overall delay of up to 10ms. This is unacceptable if we want to achieve a precision which is lower than one millisecond. Thus, we implemented a MAC timestamping which is based on the same timer as used for the MAC-layer. It should be noted that the timers used at the MAC-layer are based on a crystal oscillator and therefore are more precise than the timers left for the application-layer which are based on an RC-oscillator. The MAC timestamping

is performed by adding a new 16-bit time field to every message. This time field contains the duration used for passing down the information from the application layer to the physical layer at the sender. The receiver extracts this duration field and stores a timestamp of the receiving event. When the message at the receiver is passed up to the application layer, a callback function is called which also contains the sender duration field and the receiver duration field in the parameter list. The sum of both delays determine the complete MAC delay without the incorporation of the negligible propagation delay.

7.1.2 Implementation of the Firefly Algorithm

The synchronization algorithm was implemented analogously to the implementation in JProwler. A simple RC-oscillator based 16-bit timer was used to generate the synchronization interval with a duration of one second. Assuming a nominal oscillator frequency of 8MHz, we decided to set a prescaler of 256. As a result we get a granularity of 31250 ticks per second. This should be good enough to achieve a synchronization precision lower than one millisecond.

We modified the initial settings of the MAC sublayer to reduce the transmission delay. For this reason, we assumed that it is better to lose a message than to transmit synchronization data which are so much postponed that the time information contained in the message is incorrect. The parameters we modify are described in the MAC PAN information base (PIB) information base in [IEE03] and correspond to the CSMA scheme. For instance, we reduced the minimum backoff parameter and thus set the *macMinBE* attribute to 0. This ensures that a message is transmitted at the same time after it is passed down to the physical layer. According to [IEE03] a value of 0 disables the collision avoidance during the first iteration of the CSMA algorithm. In order to avoid the possibility that a message is postponed due to the CSMA scheme, we also set the maximum number of backoffs the CSMA-CA algorithm will attempt before declaring a channel access failure exponent to 0. This parameter is stored in the *macMaxCSMABackoffs* MAC PIB attribute. Finally our synchronization algorithm is a distributed algorithm and thus does not characterize a master node or a coordinator. On this account every node is configured as a FFD and does not require any association process. However this necessitates the use of predefined addresses. Originally one node of the demonstration kit is configured as a coordinator which assigns a short 16-bit address to all other nodes enabled in the network. Because we wanted to set up a beacon-free network without the need of a coordinator, the software for each node is compiled individually with different predefined short addresses.

As mentioned above we used a 16-bit timer to represet the synchronization interval. The timer is configured in CTC mode where the Input Capture

Register represents the top value. Every 16-bit timer supports three different compare match events. Therefore, three different registers are continuously compared with the current timer value. If the value matches than an interrupt service routine is called. For this reason, we decided to use the output compare C register to determine the transmission event for the firing offset. Considering the energy awareness, we also have to add an event which enables the receiver some time prior to the transmission start. This receiver enable event is performed in the interrupt service routine of the output compare B register. Now the output compare A is left and can be used for the TDMA communication.

7.1.3 The Modified TTP/A Protocol for TDMA Communication

In order to exchange data between several nodes without the occurrence of collisions, we decided to use a TDMA scheme which divides every TDMA round into several slots. The original TTP/A protocol was implemented for the use with a UART and further classifies one node as a master node. See [OMG03] and [EI03] for further details. Because our synchronization algorithm does not require special nodes like a master node, we replaced the synchronization part from the protocol with our distributed Firefly synchronization algorithm. Furthermore, the TTP/A protocol specifies an 8-bit logical name which is individual in a cluster. Because we used the predefined cluster-individual 16-bit short address for the logical name, we had to modify several entry-types so that it is possible to store all necessary data. For this reason, the basic RODL-entry size increased from 4 byte to 6 byte. To reserve space for future needs, two more bytes are added and in the current implementation a RODL entry has a size of 8 byte.

Other important modifications refer to the IFS which is also described in [OMG03]. Every node can contain up to 64 files which have a maximum of 256 eight-byte records in each file. The documentation file is the only file which has to be implemented on all nodes. It has the file number 0x3D and is necessary in order to identify a node. Originally, eight files can be used to determine the slots from six multi-partner rounds. These files are separated into the master slave data round, and the master slave address round. Because we have no master node in our specification, we used all eight files for multi-partner rounds. In our implementation the file-entries does not describe the slots comprised in a round. Instead every file corresponds to a slot whereas the entries correspond to the operation for a dedicated round number. On this account every period is divided into eight slots which correspond to the eight files whereas the file entries describe the operation of a slot for a specific round. In the original TTP/A version, there exist several special files. However,

most of them are not needed for the Firefly synchronization. Therefore, we removed the membership file and the ROSE file. Additionally, the current implementation does not contain the documentation file. Other special files used in the Firefly version of the TTP/A protocol are the configuration file and the RODL file. The configuration file contains among other things the current state of a node. Note that the states of the modified protocol differ from the original TTP/A version and are depicted in Figure 7.1.

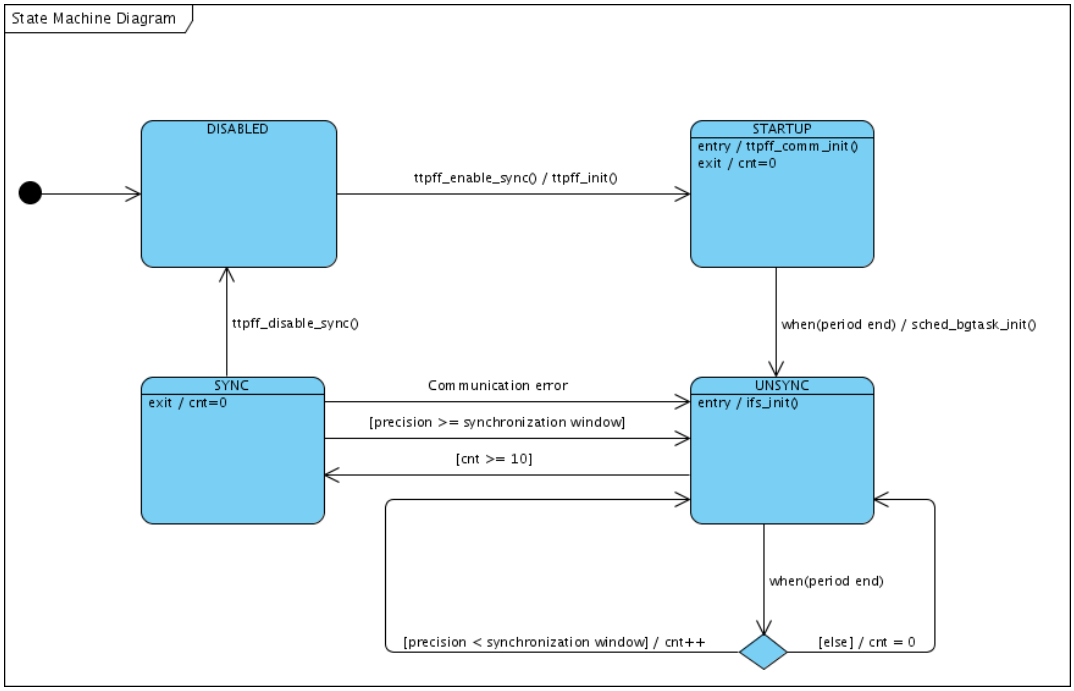


Figure 7.1: State diagram of the modified TTP/A - protocol

7.2 Resource Requirements

Because most of the integrated peripherals are already used by the MAC stack, we want to give a short overview about the task of several timers. Further this section gives an estimation about the required program memory and ram size. The main system clock is based on a RC oscillator and has a nominal frequency of 8MHz. So, if the prescaler of a timer is connected to the internal clock, then the RC oscillator is also responsible for the clock granularity of the timer.

Timer0: The 8-bit timer0 is not used by the MAC layer and can be utilized at the application layer. However, the hardware wiring does not provide the use of an external clock.

Timer1: Timer1 is a 16-bit timer and can be clocked from an external pin. According to the schematics, this external pin is connected to an external $3.68MHz$ crystal with a tolerance of 0.5%. However, this timer is again used by the MAC layer and therefore can not be shared with the application layer.

Timer2: The 8-bit timer2 is the only timer which supports asynchronous operations and therefore can be clocked from an external 32kHz watch crystal independent of the I/O Clock. For this reason, the timer is used by the MAC layer to support sleep commands and thus can not be shared with the application layer.

Timer3: Timer3 is also a 16-bit timer. Because no external clock is connected to the T3-pin, timer3 can only be used with the system clock. Moreover this is the only free 16-bit timer which supports input capture and compare match. For this reason, timer3 was chosen to be used for the Firefly synchronization algorithm.

Timer4: This 16-bit timer can only be clocked from the internal system clock, because the Atmega1281 controller does not support the T4-pin. Therefore, timer4 is free for the use at the application layer. However according to [Atm07] the input capture and output compare functionalities are not available in the Atmega1281 controller. In the current implementation timer4 was used temporarily to measure the duration of different tasks but can also be used at the application layer.

Timer5: Similarly to timer4, this 16-bit timer can only be clocked from the internal I/O clock, because no T5-pin is provided. Further the input capture and output compare functionalities for timer5 are again not available in the Atmega1281 controller. This timer can be used at the application layer.

7.2.1 Memory Analysis

To get an overview of the required memory, a memory analysis was performed individually for the MAC stack and for the synchronization algorithm in combination with the modified TTP/A protocol. The memory requirements from the application layer was assumed to be negligible and is therefore incorporated into the analysis. Table 7.1 shows the memory requirements for the MAC stack. Further Table 7.2 displays the memory requirements for the complete implementation of the Firefly algorithm and the modified TTP/A protocol without the MAC stack. Consequently, there is enough program memory, but the static

memory uses more than 65 percent. So, there are about 35 percent SRAM memory ($\sim 2800 Bytes$) left for the application and the dynamic memory which may be too little for a bigger application.

Type	Available	Used	
Flash ROM	128 kByte	32160 Byte	(24,54%)
SRAM	8 kByte	statically 1704 Byte	(20,8%)
EEPROM	4 kByte	0 Byte	(0%)

Table 7.1: Memory analysis for the MAC-Stack

Type	Available	Used	
Flash ROM	128 kByte	31191 Byte	(23,8%)
SRAM	8 kByte	statically 3722 Byte	(45,43%)
EEPROM	4 kByte	0 Byte	(0%)

Table 7.2: Memory analysis for the synchronization algorithm and the TTP/A Protocol without the MAC stack

8 Testbed Experiments

8.1 Evaluation Metrics

The evaluation metrics for the testbed experiments are similar to that one used for the simulation experiments described in Chapter 6. Because it is not easy to observe the relative deviations over all nodes in a network, we decided that every node sends its individual maximum absolute deviation of the last period to an evaluation node which then calculates the maximum over all received deviations. This value is then taken to compute the 50th and the 90th percentile group spread.

8.2 General Parameter Settings

To be able to compare the testbed results with the simulator results, the parameter configuration has to be the same as used in the simulator experiments. Table 8.1 denotes again the general parameter choice for several testbed experiments. Unfortunately in reality it is not possible to speedup the time. For this reason, we reduced the simulation end to 720 periods instead of 3600 periods.

8.2.1 Calculation of the Transmission Time

In order to calculate the transmission time, we first have to identify the amount of data which is transmitted. The synchronization frame contains the frame-identifier (8 bit), the synchronization state (8 bit), the nominal phase offset (16 bit), the phase adjustment value (16 bit), the sender timestamp (32 bit), the tick-number (16 bit) and a checksum (8 bit). In sum the application needs 13 byte for one synchronization message. The real amount of transmitted data is greater due to the payload of the MAC and the physical layer. According to Section 2.6.5, the complete payload is about 15 byte (9 byte from the MAC layer and 6 byte from the physical layer). Assuming that the system works in the $2.4GHz$ frequency band, then the transmission rate is 250 kbps. As a result the estimated transmission time equals $896\mu s$ and further can be assumed to be about one millisecond.

Parameter	Value
Oscillator technology	RC-oscillator
Initial clock drift	±100000
Interval time	1000ms
Maximum firing offset	300ms
Minimum firing offset	10ms
Phase factor	10000
Alpha value	10100/10000 = 1.01
Beta value	0
Transmission delay	320μs
Synchronization window	10ms
Evaluation end	720 periods

Table 8.1: The general parameter choice used in several testbed experiments.

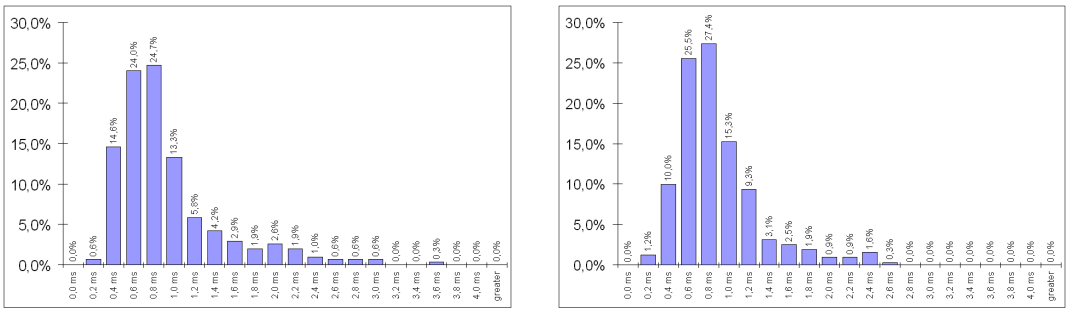
In the synchronization application, the transmission time is hard coded to correct the timestamps. Unfortunately the Firefly algorithm does not establish a time synchronization but a synchronicity. Consequently, the hard coded transmission time is based on the notion of real-time, but the established synchronicity usually does not agree with the real-time. As a result it is impossible to perfectly incorporate the transmission time even if it is locally scaled.

8.3 Testbed Results

8.3.1 All-to-all Topology Results

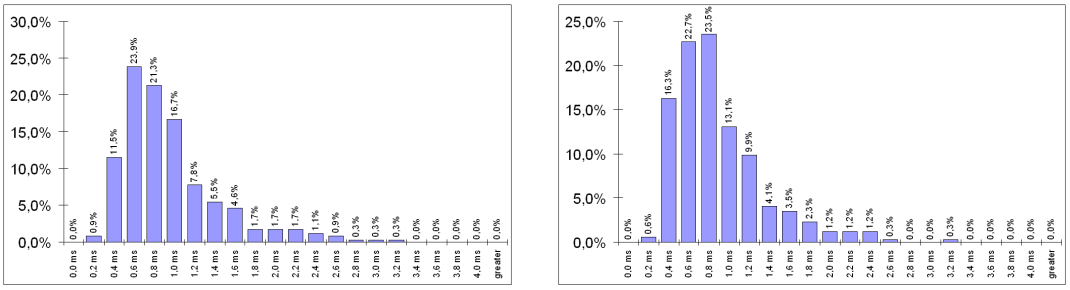
The results of an all-to-all topology comprising 5 ZigBee nodes in dependence of several coupling factors are visualized as histograms. Figure 8.1, Figure 8.2, and Figure 8.3 shows such a diagram for different coupling factors. It is interesting that all histograms have a right-skewed distribution.

Table 8.2 contains the simulation results and the testbed results with the same network configuration. The table content demonstrates that the results are similar. Whereas the 50th percentile group spread of the testbed measurement is always better, the 90th percentile and the maximum absolute deviation are very much higher. This may be caused by an insufficient implementation of the clock rate calibration algorithm. These bad results could also be evoked by the varying transmission delays at the MAC layer.



(a) $\alpha = 1.005$ (b) $\alpha = 1.01$

Figure 8.1: Group spread histogram of an all-to-all network comprising 5 nodes.



(a) $\alpha = 1.05$ (b) $\alpha = 1.1$

Figure 8.2: Group spread histogram of an all-to-all network comprising 5 nodes.

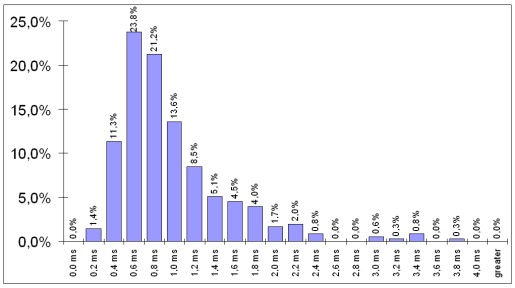


Figure 8.3: Group spread histogram of an all-to-all communication network comprising 5 nodes and with a coupling factor $\alpha = 1.15$.

8.3.2 Multi-hop Topology Results

The results from the multi-hop experiments are important in order to get an overview of the synchronization limits. The first scenario was made up of 5 nodes ordered in a chain where a node can only communicate with the immediate neighbors. This enables a simple measuring setup which is also easy to simulate. The only difference between the simulation and the testbed environment is that the testbed environment does not have an omniscient observer

Parameter choice	Time to sync (periods)	50th-percentile (μs)	90th-percentile (μs)	Maximum deviation (μs)	Standard deviation (μs)
$\alpha = 1.005$	105 (152)	672 (1000)	2005 (1300)	3456 (2200)	538 (257)
$\alpha = 1.010$	79 (57)	704 (900)	1632 (1300)	2592 (2000)	410 (250)
$\alpha = 1.050$	24 (35)	704 (900)	1973 (1300)	3040 (1900)	501 (262)
$\alpha = 1.100$	33 (20)	672 (1000)	1723 (1400)	3104 (2000)	451 (267)
$\alpha = 1.150$	14 (20)	732 (900)	1965 (1300)	3776 (1800)	565 (250)

Table 8.2: Comparison of several parameters in dependence of different coupling factors. The values between the brackets correspond to the simulation results with the same all-to-all network configuration comprising 5 nodes.

which is able to continuously measure the synchronization deviation between any two nodes. In the all-to-all topology experiment, the synchronization information of the last period was gathered at a special evaluation node. But this is not possible within a multi-hop topology even if the topology is virtually set up, because therein the only synchronization information a node can gather is with respect to the immediate neighbors. For this reason, we decided to measure the deviation with the aid of an oscilloscope whereas each node periodically sets an output pin at the same phase state for a short time. In order to include the influence of all hops in the measurements, the oscilloscope was connected to the two edge nodes of a multi-hop network. As a result the oscilloscope must be configured to trigger to the positive edge of the connected output pin and then measure the time difference to the second edge node. Unfortunately this measurement setup does not allow us to automatically gather the synchronization information over several periods. Therefore, we manually made snapshots and only took the diagrams, which display the biggest time deviation. Figure 8.4 shows the measurement setup for a typical multi-hop scenario comprising 5 nodes. Based on this topology, several oscilloscope snapshots over about 10 minutes were made. The two pictures with the greatest deviation are displayed in Figure 8.5. As a result we can say that the precision of a realistic multi-hop network with 4 hops is about 3 ms.

To get an overview of the degradation in precision with respect to the network diameter, another multi-hop experiment consisting 9 nodes was performed. The measurement setup is similar to the previous multi-hop network, but with the difference that the oscilloscope additionally measures the phase state of the centered node. This node is connected to the second oscilloscope channel and thus should give a reference curve which can be compared with the results from the previous described multi-hop topology with a network diameter of 5.

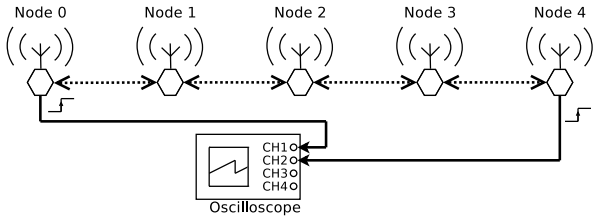


Figure 8.4: The measurement setup for visualizing the deviation between the edge nodes of a multi-hop network comprising 5 nodes.

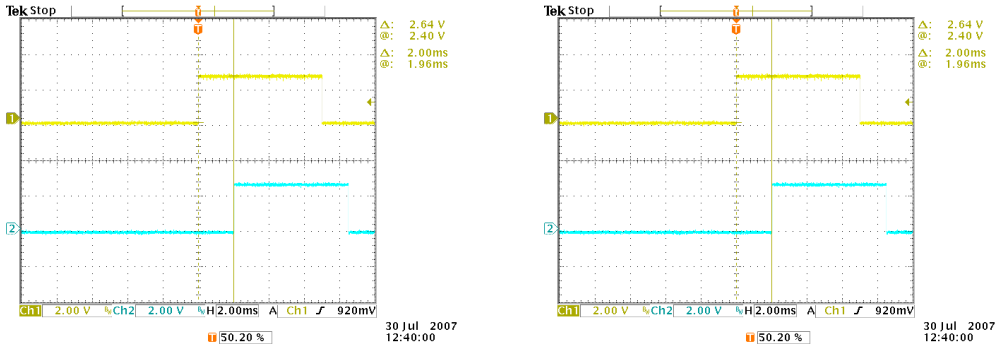


Figure 8.5: Both oscilloscope snapshots are taken from a multi-hop network comprising 5 nodes and with $\alpha = 1.01$. They show a deviation of about $2ms$ up to about $2.5ms$ between the edge nodes.

The measurement setup for this testbed experiment is visualized in Figure 8.6. The testbed results from Figure 8.7 display a maximum deviation of up to $14ms$. It is obvious that such a precision is unacceptable. In summary our synchronization algorithm dramatically degrades with each hop and is therefore not applicable for the use in a real sensor network application. It should be noted that these bad results highly likely come from the big differences in clock drift and that the clock drift calibration algorithm does not work very well in such multi-hop networks.

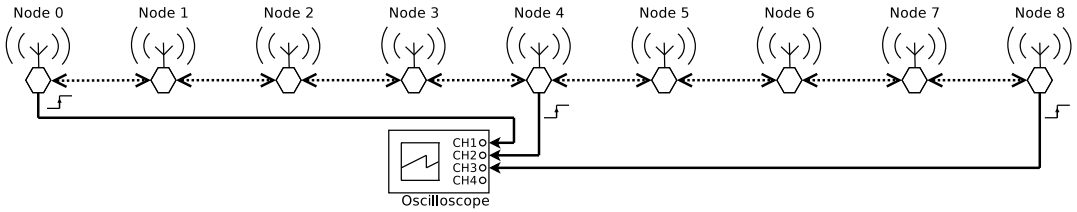


Figure 8.6: The measuring setup for visualizing the deviation between the edge nodes of a multi-hop network comprising 9 nodes.

We further measured the behaviour of a grouped multi-hop network as shown

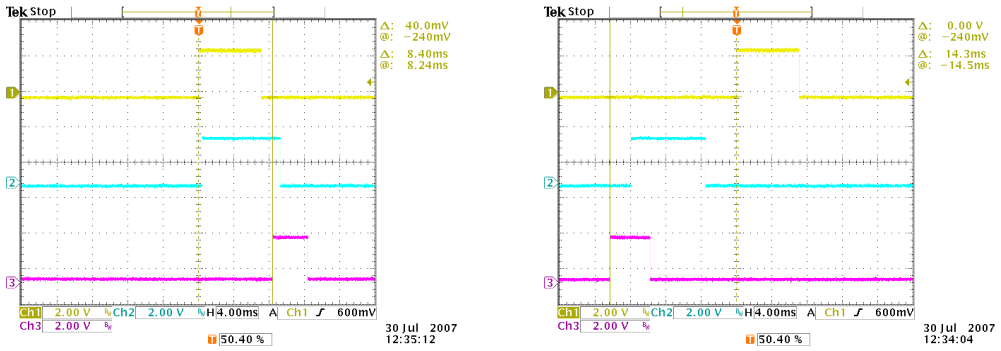


Figure 8.7: These oscilloscope snapshots show the deviation of the phase state between the edge nodes in a multi-hop network with a network diameter of 9. The alpha factor is set to 1.01.

in Figure 8.8. Because only 9 nodes were available for our experiments, this was the only acceptable topology. Therein, the oscilloscope again measures the state difference between the edge nodes. According to Figure 8.9, it can be seen that the precision improved by about 50 percent. As a result in multi-hop networks it is always better to use clusters comprising several nodes instead of single nodes whereas the nodes in such a cluster must be based on an all-to-all topology.

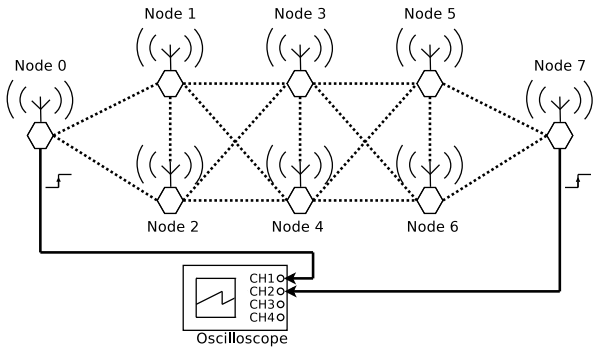


Figure 8.8: The measuring setup for visualizing the deviation between the edge nodes of a quasi grouped multi-hop network comprising 3 clusters with a cluster size of 2 and 2 edge nodes.

8.4 Energy Measurements

Beside the synchronization approach, the energy awareness is another aspect of this thesis, because the energy consumption plays an important role for the

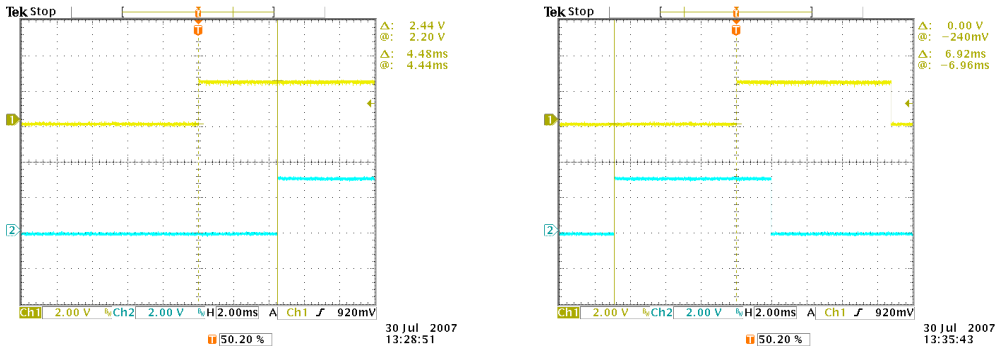


Figure 8.9: The two snapshots show the deviation of the phase state between the edge nodes in a modified grouped multi-hop network with a cluster-size of 2 nodes an 4 hops. The alpha factor is set to 1.01.

device lifetime in battery-powered wireless networks, especially in wireless sensor networks where no infrastructure is available. For this reason, we measured the power usage over time of an RCB device. The measurement setup is based on the usage of a measuring shunt which is inserted between the power supply and the RCB. This configuration is shown in Figure 8.10. An oscilloscope measures the voltage drop on this shunt which has a resistance value of exactly 1Ω. Consequently, a voltage of about $1mV$ corresponds to a current of $1mA$. Because all RCBs are battery-powered with two $1.5V$ AAA-batteries, we have a voltage supply of 3 Volt. The resulting power consumption at each point in time can then be calculated through the multiplication of the voltage supply with the measured current. In order to get the device lifetime, we have to calculate the electrical power and compare it with the electrical energy W_{bat} of the batteries which we assume to be about $3V \cdot 1200mAh = 3600mWh$. The average consumed electrical power P_{avg} can be calculated by multiplying the supply voltage by the average consumed current:

$$P_{avg} = 3V \cdot \frac{1}{T} \int_0^T i(t) \cdot \mathrm{d}t$$

The resulting lifetime in hours is then the ratio W_{bat}/P_{avg}. Because both, the enumerator and denominator, contain the same supply voltage, the fraction can be reduced to the equivalent formula

$$t_{life} = \frac{E_{bat}}{I_{avg}}$$

where E_{bat} corresponds to the battery charge usually denoted in mAh. If so, then the formula determines the device lifetime (t_{life}) in hours. The parameter I_{avg} defines the average current consumption and is similar to the average power consumption

$$I_{avg} = \frac{1}{T} \int_0^T i(t) \cdot \mathrm{d}t.$$

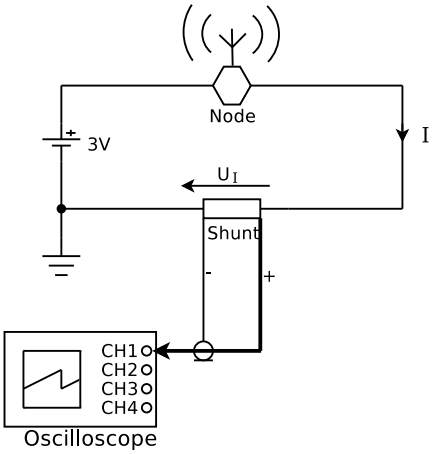


Figure 8.10: The measuring setup for visualizing the consumed current over time.

For further energy calculations, the diagrams in Figure 8.11 show the current consumption for a complete period. Therein, the consumption can be classified into four parts:

1. Firing time
2. Idle time
3. Execution time
4. Transmission time

Firing time: This is the first part in Figure 8.11 and is classified by a continuous current of about $8mA$ over a time of about $100ms$. It corresponds to the interval between the end of the period and the maximum firing offset. The high current is explained by the fact that the receiver module must be enabled during this time. The duration of this part mainly depends on the parameter choice of the maximum and minimum firing offset. In our test application we set these parameters to $10ms$ respectively $60ms$. To get a detailled information about the timing of this part, Figure 8.14 includes several oscilloscope snapshots which display the zoomed in firing time. Therein, Figure 8.14(b) shows the duration between the start and the end of the firing part. This duration of about $60ms$ should equal the difference of the maximum and minimum firing offset which is $50ms$. The missing $10ms$ comes from the fact that the receiver module is enabled exactly $10ms$ before the maximum firing offset due to the synchronization window. The consumed current during this time is about $20mA$. It

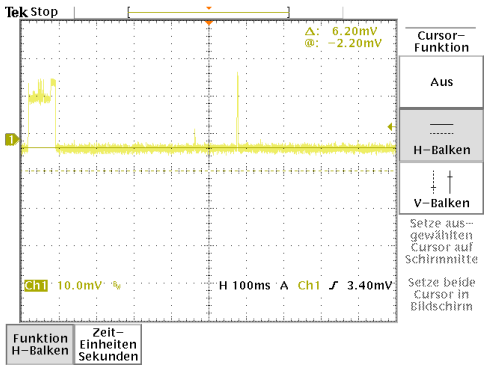
should be noted that the short peaks during these $60ms$ come from the data exchange for the Firefly algorithm. The last $13ms$ with a current of about $24mA$ results from the enabled receiver until the end of the period. This is a safety margin, because a node may start a transmission exactly at the minimum firing offset. If so, then the receiver must be enabled as long as the transmission continues.

Idle time: The idle time is the part where the current consumption drops to a minimum. The reason for the small current lies in the fact that the device is dormant. Relating to Figure 8.11, this part corresponds to the time where the diagrams display a minimum voltage of about $4mV$. As mentioned above this equals a current consumption of $4mA$. The idle time should be the biggest part in each period and thus determines the duty-cycle.

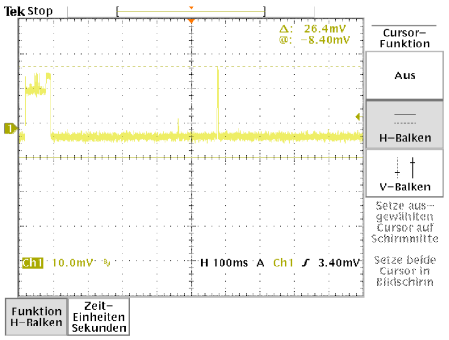
Execution time: The execution time is the time where the RCB device executes some code. In our work this is always the case during the firing time when the device wants to broadcast or receive a message and at the end of each period where the device has to apply the RFA. Other execution tasks must be configure in the RODL file. In the test application used for the energy measurement, the RODL file only contains one execution slot per period. This executed task is responsible for the data preparation for the data transmission in the following period. According to Figure 8.11 this part corresponds to the small current peak and is displayed in detail in Figure 8.13. Therein, the diagram shows a current of about $11mA$ for a duration of $1ms$.

Transmission time: This part corresponds to the time where the device is transmitting data. Normally this is always the case when the RCB wants to broadcast its synchronization message during the firing time. Other transmissions during the period must be registered in the RODL file. This is also the case for our test application and therefore corresponds to the high current peak in Figure 8.11. More detailled information about the duration on this part can be seen in Figure 8.12. Therein, the transmission part consumes a current of about $25mA$ over a time of $4.8ms$. It should be noted that this duration does not equal the calculated transmission time of $1ms$ from Section 8.2.1. This comes from the fact, that the transceiver requires some time for the startup phase. Secondary, the MAC-layer also incorporates some delay which is described in detail in Section 4.1.1. Therefore, we can assume that the maximum MAC delay $T_{MAC,max}$ is about $5ms$. Note that this is not really correct, because the measured time does not involve all delays concerning the MAC layer. For instance, the transceiver module is usually not activated immediately after the MAC layer receives a send request from the application layer.

Furthermore, the CSMA/CA scheme will also extend this duration.

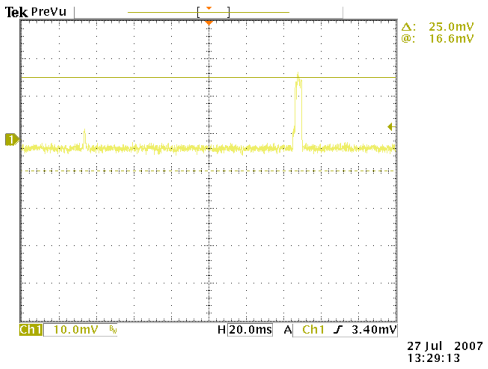


(a) Current measurement during the idle time.

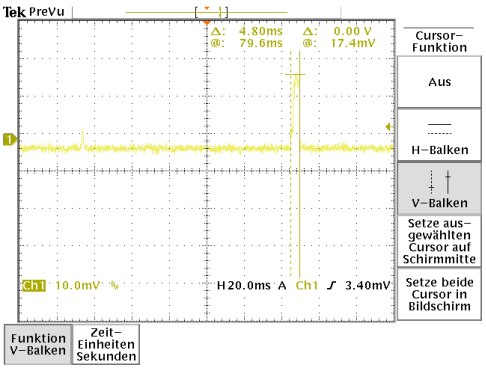


(b) Measurement of the current peak.

Figure 8.11: These two diagrams show the current consumption over a complete period.



(a) The consumed current during a transmission.

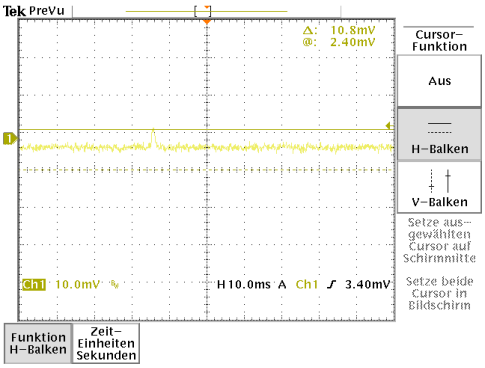


(b) Measurement of the transmission time.

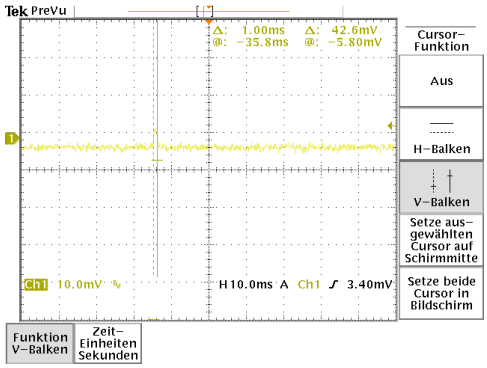
Figure 8.12: These oscilloscope snapshots display the zoomed in transmission part.

To get more information about the energy-efficiency, Table 8.3 sums up all different energy consumers with the corresponding battery discharge in mAs. In the case the period duration T is exactly one second, the values defined in this table results in a battery discharge per cycle of about $E_{device} = 7.358mAs$. Thus, the average current consumption I_{avg} equals $7.358mA$. Assuming that our batteries deliver a charge of about $E_{bat} = 1200mAh$, then the resulting lifetime (t_{life}) in hours can be calculated as follows:

$$t_{life} = \frac{E_{bat}}{I_{avg}} = \frac{1200mAh}{7.358mA} = 163h \simeq 1week$$

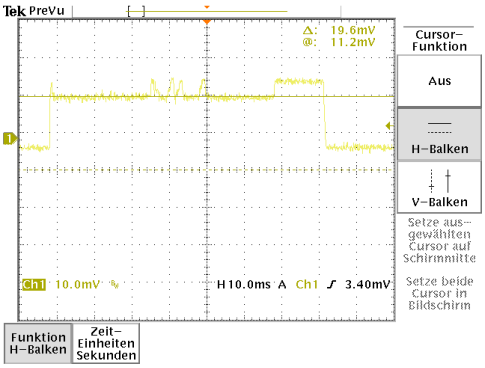


(a) [The consumed current during a task execution.

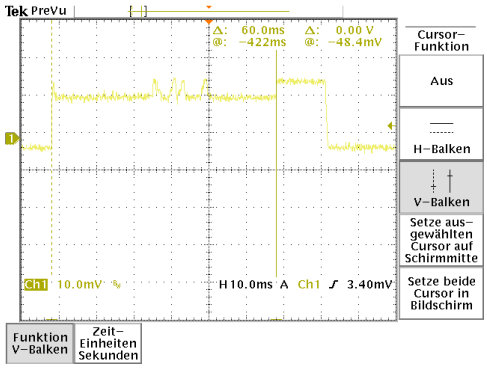


(b) Measurement of the task execution time.

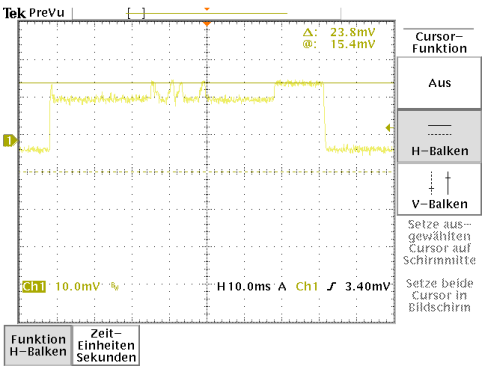
Figure 8.13: These figures visualize the zoomed in part including the task execution.



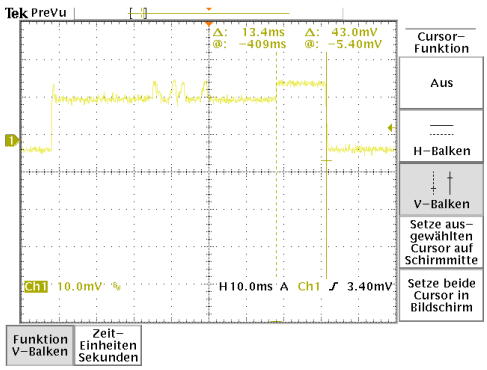
(a) The consumed current during the firing time.



(b) Measurement of the firing time.



(c) Current consumption of the receiver module.



(d) Time measurement.

Figure 8.14: The detailled diagrams including the duration between the maximum firing offset and the period end.

It should be noted that the configured duty-cycle for this result is about 7 percent, but could be reduced by increasing the period time. If we assume that the time slices of the other consumers are for the most part constant, then a larger period time induces also a larger idle time. However, this is usually not true, because a larger period time also entails a degredation in precision. Therefore, the synchronization window w must be increased due to the new period time. To simplify the correlation between the synchronization window and the period time, we set the synchronization window w into proportion with the period time T and denote the new synchronization window by $w_T = w \cdot T$. This is logical due to the correlation with the drift rate. As a result, a doubled period time will degrade the precision by a factor of two and further enforces a larger synchronization window w by the same factor. Concerning the different energy consumers, it is striking to note that only the first part of the firing time depends on the synchronization window. The other parts ($t_{f,2}$, te, and t_t) are constant. According to Section 4.1.1, $t_{f,1}$ is the difference between the maximum and minimum firing time. Furthermore, this part also contains the synchronization window which acts as a safety margin. Taken together the first part of the firing time then equals

$$t_{f,1} = T_{MaxOffset} - T_{MinOffset} + w_T = N \cdot T_{MAC,max} + w_T$$

whereas $w_T = w \cdot T$ and N denotes the maximum number of active nodes which may exist simultaneously in an all-to-all topology. Due to the measurement results from our test application, it follows that for the concrete parameter settings ($T_{MaxOffset} = 60ms$, $T_{MinOffset} = 10ms$, $w = 10ms$, and $T_{MAC,max} \sim 5ms$), the number of simultaneously active nodes in an all-to-all topology should not be greater than 10. The duty-cycle is defined to be the ratio between the sum of the two firing times ($t_{f,1}$,$t_{f,2}$), the execution time (t_e), and the transmission time (t_t) and the complete period time (T). Thus, the duty-cycle is hereinafter denoted by DC and corresponds to the equation

$$DC = \frac{t_{f,1} + t_{f,2} + t_e + t_t}{T} = w + \frac{t_{f,2} + t_e + t_t + N \cdot T_{MAC,max}}{T}$$

which shows that it can not take a value less than the synchronization window w.

To follow up on our special energy example, we further want to calculate the improvement of the lifetime with respect to the lifetime as if no synchronization approach would be established, i.e., the duty-cycle equals 100 percent. In that case the average current consumption is $23.752mA$. Consequently, a duty-cycle of 100 percent reduces the lifetime to about $50\frac{1}{2}$ hours. A comparison among the lifetime with a duty-cycle of 100 percent and the achieved lifetime with

our configured duty-cycle of about 7 percent shows that the synchronization approach improves the lifetime by a factor of three.

The following formula brings all these dependences into an equation:

$$I_{avg}(DC) = I_{idle} \cdot [1 - DC(T)] + \frac{E_{f,1} + E_{f,2} + E_e + E_t}{T}$$

Therein, I_{avg} describes the average current consumption[1] of the device which is a function of the duty-cycle $DC(T)$. Further, $E_{f,1}$, $E_{f,2}$, E_e, and E_t are the corresponding battery discharges for the firing times, the execution time and the transmission time. Note that the battery discharge for the first part of the firing time also depends on the period time:

$$E_{f,1} = I_{f,1} \cdot t_{f,1} = I_{f,1} \cdot (N \cdot T_{MAC,max} + w \cdot T)$$

As mentioned above, the duty-cycle is again a function of the period time T. Taken together we come to the following formula:

$$\begin{aligned} I_{avg} &= I_{f,1} \cdot w + \frac{I_{f,1} \cdot N \cdot T_{MAC,max} + E_{f,2} + E_e + E_t}{T} \\ &\quad + I_{idle} \cdot \left(1 - w - \frac{t_{f,2} + t_e + t_t + N \cdot T_{MAC,max}}{T}\right) \end{aligned}$$

Figure 8.15 visualizes this behaviour. It becomes obvious that the curve is similar to a hyperbola which is shifted up by a constant factor. In other words the average current consumption converges to a constant current which equals $I_{idle} + w * (I_f, 1 - I_{idle})$.

To illustrate the dependence between the lifetime improvement and the period time, we define a new variable named improvement factor which is denoted by η. As the name implies, this factor is the ratio between the improved lifetime and the reference lifetime which corresponds to a duty-cycle of 100 percent at the same period time. Due to the presence of the battery charge in the enumerator and denominator, the fraction can be reduced and the improvement factor equals:

$$\eta = \frac{I_{avg}(100\%)}{I_{avg}(DC(T))}$$

The characteristics of this factor can be seen in Figure 8.16.

[1] Note that the average power consumption P_{avg} is proportional to the average current consumption.

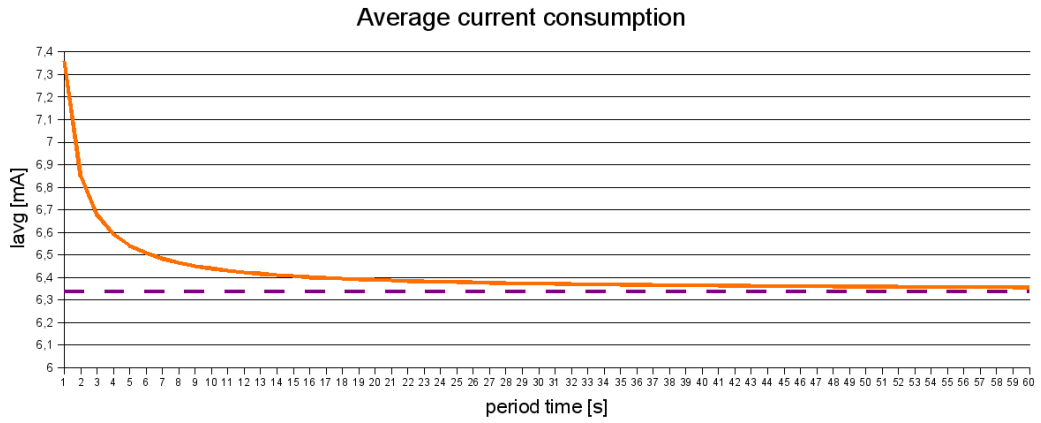


Figure 8.15: This diagram shows the average current consumption as a function of the period time in seconds.

Figure 8.16: This diagram shows the improvement factor η as a function of the period time in seconds.

Energy consumer	**I** [**mA**]	**t** [**s**]	**Battery discharge per cycle**	**Energy consumption per cycle**
Firing Time, Part1	20.0	0.060	1.200 mAs	3.600 mJ
Firing Time, Part2	24.0	0.013	0.312 mAs	0.936 mJ
Execution Time	11.0	0.001	0.011 mAs	0.033 mJ
Transmission Time	25.0	0.005	0.125 mAs	0.375 mJ
Idle Time	6.2	t_{idle}	$6.2mA \cdot t_{idle}$	$18.6mW \cdot t_{idle}$
$\sum$		T	1.648 mAs + $6.2mA \cdot t_{idle}$	4.944 mJ + $18.6mW \cdot t_{idle}$

Table 8.3: Listing of the major energy consumers and their corresponding battery discharge. The variable T denotes the duration of a complete cycle period and $t_{idle} = T - (t_{f,1} + t_{f,2} + t_e + t_t)$ defines the idle time which is the difference between the period time T and the sum of the two firing times ($t_{f,1}$, $t_{f,2}$), the execution time (t_e), and the transmission time (t_t). The energy calculation assumes a working voltage of 3 Volt.

9 Conclusion

An alternative synchronization approach based on the natural behaviour of fireflies was introduced. This synchronization method supports complete scalability and graceful degradation for the use in sensor networks. Contrary to other algorithms like the central master synchronization, this distributed algorithm does not require the use of special time master nodes. Several experiments based on an all-to-all topology have shown that it is possible to achieve a synchronization precision which is within one millisecond. However, if the network is based on a multi-hop topology, then the precision degrades to the order of several milliseconds. This is also the reason why it is difficult to achieve synchronicity in a communication network comprising communication paths including several hops. Further it is hardly possible to synchronize a complete network if it contains asynchronous communication patterns. Due to the fact that in reality many sensor networks eventually form unidirectional communication paths, this algorithm is not really applicable for sensor networks which have to support high dependability and availability. Otherwise if such a network is only used to gather data where it is not dramatic if some messages are lost (e.g., meteorological station for temperature measurement), then the use of such a synchronization scheme could be a choice.

Future work will rely on a better distributed clock drift calibration, because several simulation results have shown that the clock drift has the most important impact on the precision. Another method for improving the clock drift could be the comparison of the clock with a reference clock with a better granularity during the startup phase of a node. This will also increase the time to sync.

Bibliography

[Abb90] L. F. Abbott. A network of oscillators. *Journal of Physics A: Mathematical and General*, 23:3835–3859, 1990.

[All06] ZigBee Alliance. *ZigBee Specification*, December 2006.

[AP68] R. F. Adams and D. O. Pederson. Temperature sensitivity of frequency of integrated oscillators. *Solid-State Circuits*, 3(4):391 – 396, December 1968.

[Atm06a] Atmel Corporation. *ATAVRRZ200 Demonstration Kit AT86RF230 (2450 MHz band) Radio Transceiver*, July 2006.

[Atm06b] Atmel Corporation. *AVR053: Calibration of the internal RC oscillator*. Atmel, May 2006.

[Atm07] Atmel Corporation. *8-bit Microcontroller with 64K/128K/256K Bytes In-System Programmable Flash*, January 2007.

[BB76] J. Buck and E. Buck. Synchronous fireflies. *Scientific American*, (234):74–85, 1976.

[BBL62] R. Bechmann, A. D. Ballato, and T. J. Lukaszek. Frequency-temperature characteristics of quartz resonators derived from the temperature behaviour of the elastic constants. *16th Annual Symposium on Frequency Control*, 16:77 – 109, 1962.

[Bec56] R. Bechmann. Frequency temperature-angle characteristics of at-type resonators made of natural and synthetic quartz. In *IRE*, volume 44, pages 1600–1607, July 1956.

[BL05] A. Bletsas and A. Lippman. Spontaneous synchronization in multi-hop embedded sensor networks: Demonstration of a server-free approach. In *Wireless Sensor Networks*, Second European Workshop, pages 333–341, January 2005.

[Buc88] J. Buck. Synchronous rythmic flashing of fireflies. *The Quarterly Review of Biology*, 63(3):265–289, September 1988.

[CGH+02] E. Callaway, P. Gorday, L. Hester, J.A. Gutierrez, M. Naeve, B. Heile, and V. Bahl. Home networking with ieee 802.15.4: a developing standard for low-rate wireless personal area networks. *IEEE Communications Magazine*, 40(8):70–77, August 2002.

[CK03] C.-Y. Chong and S. P. Kumar. Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, pages 1247 – 1256, 2003.

[DHS84] D. Dolev, J. Halpern, and H. R. Strong. On the possibility and impossibility of achieving clock synchronization. *In Proceedings of the sixteenth annual ACM symposium on Theory of computing STOC '84*, December 1984.

[EGE02] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Fifth Symposium on Operating Systems Design and Implementation*, December 2002.

[EGPS01] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *In Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP 2001)*, May 2001.

[EHK+05] W. Elmenreich, W. Haidinger, R. Kirner, T. Losert, R. Obermaisser, and C. Trödhandl. *TTP/A Smart Transducer Programming, A Beginner's Guide*. Institut für Technische Informatik, Technische Universität Wien,Vienna, Austria, 0.5 edition, January 2005.

[EI03] W. Elmenreich and R. Ipp. Introduction to ttp/c and ttp/a, 2003.

[EP03] Chris Evans-Pughe. Bzzzz zzz – zigbee wireless standard. *IEEE Review*, 49(3):28 – 31, March 2003.

[Ger96] W. Gerstner. Rapid phase locking in systems of pulse-coupled oscillators with delays. *Physical Review Letters*, 76(10), March 1996.

[GNC+01] J.A. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile. Ieee 802.15.4: a developing standard for low-power low-cost wireless personal area networks. *IEEE Network*, 15(5):12–19, 2001.

[HS05] Y.W. Hong and A. Scaglione. A scalable synchronization protocol for large scale sensor networks and its applications. *IEEE Journal on Selected Areas in Communications*, 23(5):1085 – 1099, May 2005.

[HS06] A. Hu and S. D. Servetto. On the scalability of cooperative time synchronization in pulse-connected networks. *IEEE Transactions on Information Theory*, 52(6):2725 – 2748, June 2006.

[HSD+06] S. Hammouda, H. Said, M. Dessouky, M. Tawfik, Q. Nguyen, W. Badawy, H. Abbas, and H. Shaheen. Analog ip reuse in nano technologies. Internet, April 2006.

[IEE90] IEEE Computer Society. *IEEE Standards for local and metropolitan area networks: overview and architecture*. Institute of Electrical and Electronics Engineers, November 1990.

[IEE98] IEEE Computer Society. *Information processing systems – Local area networks – Part 2:logical link control*. Institute of Electrical and Electronics Engineers, December 1998.

[IEE03] IEEE Computer Society. *IEEE Standard for Information technology – Telecommunication and information exchange between systems – Local and metropolitan area networks – Specific requirements. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*. Institute of Electrical and Electronics Engineers, September 2003.

[Kin03] P. Kinney. Zigbee technology: Wireless control that simply works. www.zigbee.org/resources, October 2003. Whitepaper.

[Kop97] H. Kopetz. *Real-Time Systems*. Kluwer Academic Publisher, 1997.

[Lic91] J. A. Lichter. Crystals and oscillators, July 1991. JL9113 Rev. B1.

[LL84] J. Lundelius and N. Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62(1):190–204, 1984.

[LW04] D. Lucarelli and I-Jeng Wang. Decentralized synchronization protocols with nearest neighbor communication. *SenSys'04*, pages 62–68, November 2004.

[MS90] R. E. Mirollo and St. H. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, December 1990.

[MT89] M.D. Mesavoric and Y. Takahara. *Abstract Systems Theory*, volume 116 of *Lecture Notes in Control and Information Sciences*. Springer-Verlag Berlin, Heidelberg, Germany, 1989.

[OMG03] Inc. Object Management Group. Smart transducers interface, 2003.

[Pau02] M. Paulitsch. *Fault-Tolerant Clock Synchronization for Embedded Distributed Multi-Cluster Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, September 2002.

[Pes75] C. S. Peskin. Mathematical aspects of heart physiology. Technical report, Courant Institute of Mathematical Sciences, 1975.

[Pus05] P. Puschner. Experiments with wcet-oriented programming and the single-path architecture. In *International Workshop on Object-Oriented Real-Time Dependable Systems*, volume 10, pages 205 – 210, February 2005.

[RSPS02] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava. Energy-aware wireless sensor networks. *IEEE Signal Processing Magazine*, 19(2):40–50, March 2002.

[SBK05] B. Sundararaman, U. Buy, and A. D. Kshemkalyani. Clock synchronization in wireless sensor networks: A survey. *In Ad-Hoc Networks*, 3(3):281–323, May 2005.

[Sch88] W. Schwabl. *The Effect of Random and Systematic Errors on Clock Synchronization in Distributed Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Inst.-Nr. E182/1, October 1988.

[Sch93] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, 1993.

[Sch95] A. V. Schedl. The short-term stability of crystal oscillators: Experimental results. Technical Report 1/95, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, January 1995.

[Sch96] A. V. Schedl. *Design and Simulation of Clock Synchronization in Distributed Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, April 1996.

[ST87] T. K. Srikanth and S. Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, July 1987.

[SVML03] G. Simon, P. Volgyesi, M. Maroti, and A. Ledeczi. Simulation-based optimization of communication protocols for large-scale wireless sensor networks. *Aerospace Conference*, 3:1339 – 1346, March 2003.

[TAB06] A. Tyrrell, G. Auer, and C. Bettstetter. Firefly synchronization in ad hoc networks. In *MiNEMA Workshop*, February 2006.

[Vig00] J. R. Vig. Quartz crystal resonators and oscillators, January 2000.

[WATP+05] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In *3rd international conference on Embedded networked sensor systems*, pages 142–153, November 2005.

[Wil98] U. Wilensky. Netlogo fireflies model. Internet, 1998. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

[YHE04] W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. In *In IEEE/ACM Transactions on Networking*, volume 12, page 493, June 2004.

A List of Acronyms

ACL Access Control List

AF Application Framework

APS Application Support

BE Backoff Exponent

CAP Contention Access Period

CCA Clear Channel Assessment

CID Cluster Identifier

CLH Cluster Head

COTS commercial off-the-shelf

CFP Contention Free Period

CRC Cyclic Redundancy Check

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance

CW Contention Window

DCXO Digital Compensated Crystal Oscillator

FCS Frame Check Sequence

FFC Firing Function Constant

FFD Full-function Device

FIFO First In First Out

GTS Guaranteed Time Slot

IFS Interface File System

LLC Logical Link Control

LR-WPAN Low-rate Wireless Personal Area Network

MAC Media Access Control

macMaxCSMABackoffs Maximum Backoff Exponent

macMinBE Minimum Backoff Exponent

MaS Mirollo and Strogatz

MCXO Microcomputer Compensated Crystal Oscillator
MFR MAC Footer
MHR MAC Header
MPDU MAC Protocol Data Units
MSA master-slave address
MSD master-slave data
MSDU MAC Service Data Unit
NB Number of Backoff Periods
NWK Network
OCXO Oven Controlled Crystal Oscillator
OSI Open System Interconnection
PAN Personal Area Network
PCO Pulse-coupled Biological Oscillators
PHY Physical
PIB PAN information base
PID Proportional-Integral-Differential
PPDU Physical Protocol Data Unit
PRC Phase Response Curve
PSDU PHY Service Data Init
RAM Random Access Memory
RBS Reference-Broadcast Synchronization
RCB Remote Controller Board
RFD Reduced-function Device
RODL Round Description List
ROM Read-Only Memory
ROSE Round Sequence
RFA Reachback Firefly Algorithm
SAP Service Access Point
SNR Signal-to-noise Ratio
TCXO Temperature Compensated Crystal Oscillator
TDMA Time Division Multiple Access

TTP Time-Triggered Protocol

UWB Ultra Wideband

VCXO Voltage Controlled Crystal Oscillator

WCET Worst-Case Execution Time

ZC ZigBee Coordinator

ZDO ZigBee Device Object

ZED ZigBee End Device

ZR ZigBee Router