FAKULTÄT FÜR !NFORMATIK

Robust Self-organizing Pulse Synchronization in Wireless Sensor Networks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Robert Leidenfrost

Matrikelnummer 0426381

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Priv.-Doz. Dr. Wilfried Elmenreich

Wien, 09.11.2009 _____ _____

(Unterschrift Verfasser) (Unterschrift Betreuer)

Robust Self-organizing Pulse Synchronization in Wireless Sensor Networks*

Robert Leidenfrost

09.11.2009

*Supported by the Austrian Science Fund (FWF) project P18060N04.

Declaration

Robert Leidenfrost
Attemsgasse 5/1/207
1220 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift)

Abstract

Tremendous advance in technology requires and allows us to build complex architectures by decomposing it into smaller manageable and loosely coupled components. The meaningful exchange and comparison of observations among these components then requires a system wide agreement on a common notion of time. This is, for example, an important issue in the case fault tolerance is implemented by replication. Whereas many wired distributed systems provide enough capabilities in order to achieve agreement, Wireless Sensor Networks demand much higher standards of the available energy resources and consequently necessitate an energy-efficient communication protocol. This is usually achieved by synchronized sleep-wakeup schedules. As a consequence, clock synchronization in complex distributed systems is inevitable to provide composability, dependability, and temporal coordination.

This thesis presents a well-studied and simple fault-tolerant distributed clock synchronization algorithm which was modified for the use in sensor networks and extended in order to be self-stabilizing, i.e., independent of the initial configuration, all devices eventually become synchronized. In other words, the presented approach combines the advantage of two different synchronization algorithms. In detail, whereas the convergence to a synchronized system state is ensured in the fault-free case, synchronicity is maintained even in the presence of at most $f < \frac{n}{5}$ Byzantine nodes. The algorithm also works in unstructured multi-hop networks by exploiting the existence of redundant communication links. Several simulation results with respect to different network topologies are presented and promise an improved network-wide synchronization precision, an acceptable convergence time, energy efficiency through a low message complexity, and robustness against different kinds of faults.

Zusammenfassung

Die enormen Fortschritte verschiedenster Technologien erfordern und ermöglichen es, dass komplizierte Architekturen in kleinere und lose gekoppelte Einheiten zerlegt werden, welche überschaubarer und kontrollierbarer sind. Ein sinnvoller Austausch und Vergleich von Beobachtungen unter diesen Einheiten erfordert die Übereinstimmung der lokalen Zeit aller Einheiten mit einer systemweiten gemeinsamen Zeitbasis. Dies ist auch eine wichtige Voraussetzung für die Implementierung von Fehlertoleranz durch Replikation und gilt neben verdrahteten verteilten Systeme ebenso für drahtlose Sensornetzwerke. Zusätzlich haben Knoten innerhalb von drahtlosen Sensornetzwerken hohe Anforderungen an die effiziente Nutzung von Ressourcen und benötigen folglich ein energieeffizientes Kommunikationsprotokoll. Dies wird für gewöhnlich durch synchronisierte Schlaf-Wach-Phasen erreicht. Als Folge ist Uhrensynchronisation eine unabdingbare Notwendigkeit in nahezu allen verteilten Systemen um Komponierbarkeit, Zuverlässigkeit und zeitliche Koordination zu erreichen.

Diese Arbeit basiert auf einen allgemein bekannten, fehlertoleranten, verteilten Uhrensynchronisationsalgorithmus, welcher für die Anwendung in drahtlosen Sensornetzwerken modifiziert und erweitert wurde. Der modifizierte Algorithmus verbindet die Vorteile zweier unterschiedlicher Ansätze, sodass schlussendlich Selbststabilisierung in fehlerfreien Netzwerken und eine hohe Synchronisationsgenauigkeit in fehlerbehaftete Netzwerken in der Anwesenheit von maximal $f < \frac{n}{5}$ Byzantinische Knoten erreicht werden kann. Selbststabilisierung im generellen Sinne beschreibt dabei die Eigenschaft eines Systems, sich selbst aus einem beliebigem Ausgangszustand in einen definierten Endzustand, also die Synchronisation aller Einheiten, zu versetzen. Der Algorithmus funktioniert weiters auch in unstrukturierten, zumindest $(5f+1)$-verbundenen, Multi-hop Netzwerken durch Ausnützung der hohen Redundanz der Kommunikationsverbindungen. Verschiedenste Simulationsergebnisse in Bezug auf unterschiedliche Netzwerktopologien werden diskutiert und zeigen, dass eine annehmbare Konvergenzzeit und eine hohe netzwerkweite Synchronisationsgenauigkeit bei gleichzeitig niedriger Nachrichtenkomplexität und großen zeitlichen Verzögerungen der Nachrichtenübertragungen erreicht werden kann.

Acknowledgements

I am heartily thankful to my supervisor, Wilfried Elmenreich, who made this thesis possible. His encouragement and support were very helpful in order to improve the quality of the work. Further, I would like to thank Johannes Klinglmayer for his constructive comments and feedback. Lastly, and most importantly, I wish to gratefully acknowledge my parents and Iris for their support and understanding.

Contents

CHAPTER 1

Introduction

Current research in wireless technology and the smaller feature size in Very Large Scale Integration (VLSI) chips leads to the integration and interconnection of small, mostly battery-powered, devices like mobile phones as well as simple low-cost sensor nodes. Tremendous potential lies in wireless networking of smart transducers in harsh environments where no external energy source and time reference is available. The general definition of a WSN embossed by early research projects is a distributed, large-scale, ad-hoc, multi-hop, unpartitioned network of largely homogeneous, tiny, resource-constrained, and mostly immobile sensor nodes [RM04]. However, the requirements and characteristics of a WSN may vary due to the wide range of applications today. Whereas the development was originally motivated by the military domain, such networks have found their application in a variety of other domains including but not restricted to environmental, industrial, medical, scientific, and home networks. Several sample projects using WSNs and their classification with respect to the design space can be found in [RM04]. One of the main reasons for the increasing interest in such networks has been the technological advance in miniaturization of Microelectromechanical Systems (MEMS) during the last decade. In [WLLP01], the authors introduced the term of *smart dust* and discuss how a complete autonomous sensing, computing, and communication system could be packed into a cubic-millimeter mote. Although such devices are not yet available on the market, many companies already build centimeter-scale hardware.

The main advantage of this distributed sensing is that it allows a closer placement to the phenomenon and, by means of data fusion, calculate a single measurement value from the data of hundreds of sensor nodes resulting in a better Signal-to-noise Ratio (SNR) as an individual sensor could provide [EGPS01]. This demonstrates that the basic operation in such networks is the efficient aggregation of the measured sensor data to usually a single sink which executes a fusion algorithm to provide an accurate and reliable result [PK00, YKT03, GM04]. In practice, hundreds of such sensor nodes are randomly scattered in a usually inaccessible, dangerous, and possibly hostile environment of interest to reliably monitor a certain physical phenomenon (*e.g.*, temperature, humidity, sound, pressure, seismic vibrations, motion, pollutants).

The importance of clock synchronization in this area can be shown by numerous examples: In [SML+04], a WSN is used to locate snipers and the trajectory of bullets by the use of acoustic sensors. This is a typical sensor-sink based application where the sensors are largely distributed forming a multi-hop ad-hoc network. By comparing the timestamped events of the acoustic shock measured by each node, the sniper can be located with an ac-

curacy of about one meter. The time-stamping in this example necessitates that each sensor is synchronized to a common notion of time which is provided by a clock synchronization protocol. Other examples requiring the nodes to agree on a common notion of time are the relative ordering of events received from different nodes as stated in [Lam78] or configuring a beam-forming array or setting a Time Division Multiple Access (TDMA) radio schedule [ZG04, HE04]. Furthermore, due to the limited power the nodes can harvest or store and the fact that the devices generally must have a lifetime on the order of months to years [RSPS02] without battery replacement, an energy efficient protocol has to be established. To overcome this problem, several sophisticated energy scavenging techniques have been developed. However, beside computation the communication is a key energy consumer. In [PK00], the authors have shown that the energy cost transmitting 1Kbit of data at a distance of 100 meters is about 3 joules and equals the execution of 3 million instructions on a general-purpose processor with 100 MIPS/W. According to [YHE04], the major sources of energy waste are packet collisions, overhearing, control packet overhead, and idle listening. Note that many Media Access Control (MAC)-protocols such as IEEE 802.11 spend more than 50 percent on idle listening [YHE04]. For this reason, most existing protocols focus on low duty-cycling, where the nodes try to reduce energy waste by synchronously switching on the transceiver only if it is required [ACFP09]. Several other approaches have been proposed to improve energy efficiency focusing mostly on clustering mechanisms, routing algorithms, energy dissipation schemes, synchronized listen/sleep schedules. Likewise, many of these techniques require some kind of synchronization among the nodes.

Intensive research over the last few decades have lead to wealth results about distributed clock synchronization in wired networks. Several algorithms assume a combination of internal and external clock synchronization in order to achieve a time synchronization with high accuracy. For instance, Network Time Protocol (NTP) [Mil91] is a well-established protocol in computer networks and achieves accuracies on the order of a few microseconds with respect to real-time. Note that distributed synchronization algorithms generally require the nodes to periodically acquire knowledge about the state of the global time counters of the other nodes [Kop97]. This communication is usually done by exchanging messages among the nodes. Such an approach is less suitable for designing a distributed synchronization algorithm for WSNs which we have to stand for, because this dissipates a lot of bandwidth. Furthermore, an external time reference like Global Positioning System (GPS) is broadly not ubiquitous available in WSNs, because the field of application has in general no infrastructure providing such signals (*e.g.*, indoors or underwater). Additionally, the high power demand of these peripheral devices prohibits the use on low-cost sensor nodes with a small and finite energy source. Last but not least, the complexity of NTP and the limited bandwidth as well as the unstable network topology in wireless networks make the use beyond all questions. Several other aspects why NTP is not the best choice are discussed in [ER03]. As a result the nodes often implement internal software-based clock synchronization, because this is cost-efficient and more appropriate. In detail, these types of algorithms are aimed at achieving high clock precision with respect to all other clocks, but may strongly deviate with respect to real-time. In contrast, time synchronization requires that the nodes additionally approximate the real time.

Examples for fault-tolerant distributed internal clock synchronization algorithms in wired point-to-point communication networks are Fault-tolerant Averaging (FTA) [KO87], Fault-tolerant Midpoint (FTM) [LL84a], Differential Fault-tolerant Averaging (DFTA) [ND00], Differential Fault-tolerant Midpoint (DFTM) [FC95], Fault-tolerant Daisy-chain clock synchronization [Lön99], and so on. Note that most of these algorithms are based on a periodic resynchronization where the processors exchange their clock values or perform remote clock reading at the resynchronization points. This implies a high communication bandwidth.

However, the broadcast medium and the more complex challenges in wireless networks, comprising up to several thousands of sensor nodes, makes the direct deployment of these algorithms unsuitable. For instance, omission failures, unidirectional links and message collisions can not completely be avoided. As an example, in [GKW+02], empirical studies have shown that a dense WSN results in an increasing number of messages lost due to contention loss. Furthermore, many wireless systems have high requirements on energy consumption, fault tolerance, availability, scalability, graceful degradation, and dependability. This is much more difficult to incorporate if the topology is created ad-hoc and, additionally, may change over time. For this reason, WSNs often necessitate algorithms, which provide robustness and some kind of self-configuration for clock synchronization and message routing in the case of a changing topology.

1.1 Motivation

Clock synchronization is an important issue in WSNs as already mentioned above. Good surveys on clock synchronization algorithms in wireless networks can be found in [ER03, SY04, SBK05, Fai07, RLK+09]. Other interesting surveys regarding traditional resp. fault-tolerant clock synchronization are stated in [Sch87, Cri89, SWL90, AP98]. However, most of them do not discuss completely distributed internal clock synchronization algorithms without the need for dedicated nodes. In contrast, many of them establish a master/slave like synchronization. In other words, designing such a distributed algorithm for autonomous WSNs, especially considering multi-hop ad-hoc networks, is an interesting and new challenge in this scientific domain. Above all, considering Byzantine failures which can behave arbitrarily in an adversary manner is an additional important issue (for more on *Byzantine failures*, see [LSP82]). Whereas many proposed algorithms already assume such a malicious system model, they generally act on the assumption of initially synchronized clocks and do not consider the case that the number of transient faults may exceed the maximum number of tolerating faults. This may result in a corruption of the local and consequently the global state of the nodes such that they never return to a consistent global state, even if the network behaves coherently again. On this account, self-stabilizing clock synchronization algorithms avoid this dilemma by not assuming initially synchronized nodes. In detail, an algorithm is called *self-stabilizing* if it can tolerate transient faults in the sense that if the transient faults leave the system in an arbitrary state and the system behaves coherently for a sufficiently long period of time, then the system converges back to a consistent global state. An introduction and a short survey on self-stabilization can be found in [Dij74] resp. [BS00, Sch93]. An extensive study was done by S. Dolev in [Dol00]. Recently, Dolev *et al.* have investigated several algorithms which provide both self-stabilizing digital clock synchronization and Byzantine tolerance based on a lock-step round execution [ADG92, DW93, DW04, DD05, DDP06, HDD06, Hoc07, DH07b, BODH08]. Other important work was done by Gouda *et al.* [GH90], Papatriantafilou *et al.* [PT94], and Malekpour [Mal06]. These protocols can be classified into probabilistic and deterministic approaches. A further characteristic is given by the number and type of faulty nodes.

Lock-step round execution in our case means that nodes execute in lock-step by regularly receiving a common "pulse" in tight synchrony. The *digital clock synchronization* problem was first studied by Even and Rajsbaum [ER90] and describes the problem how clocks eventually can operate in step in a synchronous system where all clocks have an identical initial integer value, but starting at different time instants, such that as long as the nodes remain correct, they will continue to hold the same value increased by one at each step [ADG92]. Note that the digital clock synchronization problem differs from the traditional clock syn-

chronization problem [Sch87, Cri89] in the following way that, in *clock synchronization*, a synchronous system is not assumed. In detail, in the presence of bounded drift rates, the clocks have to be maintained such that they never drift too far apart. However, digital clock synchronization may be implemented on top of clock synchronization.

Alternatively, digital clock synchronization may also use pulse synchronization as an underlying building block. However, whereas several pulse synchronization algorithms have been proposed [DDP08, DD08, DH07a] providing mostly an optimal linear convergence time with respect to [DHS84], all of them have a high message complexity or are too complicated and assume a system model which is not appropriate for the use in WSNs. For instance, both [DH07a] and [DD08] are created upon the execution of the SS_BYZ_AGREE algorithm [DD06] each cycle. Since this algorithm has a message complexity of $O(f \cdot n^2)$ [DH07a], a node has to transmit more than one message per cycle. This is unacceptable in a WSN, where the number of messages sent per cycle should be kept to a minimum due to the energy efficiency. The only interesting work with respect to our system model is the biologically inspired algorithm stated by Daliot *et al.* [DDP08], because therein, each node transmits exactly one message per cycle. Although it has a convergence time of $O(f)$ cycles, this algorithm is more appropriate for the use in wireless networks. In other words, it is better to have a longer convergence time than a high message complexity in each round.

Note that the first report regarding pulse synchronization in biological species was done by J. Buck and E. Buck. in [BB76, Buc88] and deals with the synchronous flashing of thousands of male fireflies observed in the Southeast Asia. Based on the observations, they have devised two different synchronization models, namely the *phase-advance* and *phase-delay* synchronization model. Independently, Peskin devised a mathematical model of the oscillations of the neurons in the cardiac pacemaker and is known as the pulse-coupled integrate-and-fire model [Pes75]. However, most important work was done by Mirollo and Strogatz in [MS90, SS93], where they developed the general Pulse-coupled Biological Oscillators (PCO) model, which corresponds to the phase-advance synchronization principle. Several other biological examples can be found in [SS93]. Note that with respect to WSNs, the PCO model was already adapted by Wernerl-Allen *et al.* in [WATP$^+$05] for the use in wireless networks and is known as the Reachback Firefly Algorithm (RFA). In detail, the original PCO model assumes that each node broadcasts a message when a node invokes a new pulse (or resets the counter to start a new cycle). Note that the message complexity in this case is optimal with respect to pure distributed synchronization algorithms. In a dense wireless network, however, a lot of sensors may compete for the same wireless communication medium at the same time which may result in strongly delayed or even lost messages due to the Carrier Sense Multiple Access (CSMA) scheme. This makes a direct deployment useless, but can be circumvented by assuming sparsely connected networks or in the extreme case a cell based structure. For this reason, the main adaption belongs to an additional *message staggering delay* such that the messages are sent some random time prior the original pulse invocation. This relaxes the MAC contentions. Note that this principle can be applied to all other similar algorithms originally developed for wired networks, where messages are sent simultaneously (*e.g.*, FTA, FTM, *etc.*). This already reflects a general modification and adaptation scheme of distributed synchronization algorithms from wired to wireless networks. Additional analyzations regarding the RFA were done in [LE08]. Although this algorithm is self-stabilizing and provides the necessary message complexity, it is not resilient to Byzantine nodes. However, independently of the proposed RFA scheme, Daliot and Dolev developed a Byzantine tolerant variant of the PCO model in [DDP08], where the message complexity keeps the same and as well provides a convergence time of $O(f)$ cycles with a near optimal synchronization precision. With an additional message staggering delay, this algorithm seems to be suitable for the use in wireless networks, because

they additionally assume a similar system model as proposed in this thesis (*e.g.*, bounded clock drifts, bounded message transmission delay).

1.2 Objectives

This thesis deals with the adaptation of a fault-tolerant distributed clock synchronization algorithm for WSNs and additionally considers the aspect of self-stabilization. Throughout the next chapters, the following research questions are treated and discussed:

- Do there already exist protocols for fault-tolerant clock synchronization in wireless networks? If so, how can they be classified and compared?
- How can a multi-hop WSN be globally synchronized even under the presence of Byzantine faults?
- What is the achievable precision under various failure models and network topologies?

To come straight to the result, a well-studied distributed clock synchronization algorithm named FTA was combined with a modified version of the RFA approach and consequently provides the advantages of both, a high achievable synchronization precision in the presence of Byzantine faults, and robust self-stabilization in the presence of erroneous nodes that do not behave in an adversary manner. More precisely, let $f < \frac{n}{5}$ be the maximum number of tolerable faulty nodes. The RFA algorithm was taken and modified such that a node discards the f largest and the f smallest time differences. The resulting approach is then used to establish a more robust coarse synchronization. If a node notices that all received events (excluding the f largest and f smallest time differences) are within a predefined precision, then the node performs a state transition from the coarse synchronization protocol to a fine synchronization protocol. The fine synchronization protocol then can be any distributed fault-tolerant synchronization algorithm, which originally required initially synchronized clocks. In this thesis, the FTA algorithm was chosen for the fine synchronization due to its simplicity and elegance. Clearly, both protocols make use of the same messages and therefore do not affect the message complexity. As a proof of concept, the algorithm has been evaluated by simulation with different topologies. The results are promising and give realistic figures for the precision of the pulse synchronization and the achievable savings in power consumption due to a coordinated listen/sleep schedule. This has the inherent advantage that the clock synchronization no longer suffers from a single-point of failure as it would be in a central master clock synchronization approach. Furthermore, this type of synchronization does not need any explicit cooperation between the nodes. To sum up, this thesis focuses on the synchronization of largely distributed and completely autonomous mobile wireless networks comprising hundreds of nodes and additionally evaluates the behavior in multi-hop topologies. To give an application example, assume that each node is equipped with a sensor and an actuator and all execute the same program. The objective of this example could be that all nodes simultaneously activate their actuators in dependence of the fused data. In other words, the nodes have to achieve agreement in a decentralized network even under the presence of byzantine faults. In the literature, a variant of the aforementioned application is also known as the firing squad problem. Recently, in [LH08], Leu *et al.* have already designed such an agreement algorithm for WSNs. In detail, the authors propose a protocol for an architecture without a sink, where several nodes are placed in a room and form an autonomous cluster targeted to measure the temperature (e.g, to establish a fire fighting application). If a node measures a temperature higher than 50 degree Celsius, then it sets the initial value to 1, otherwise to 0. The proposed protocol ensures agreement among all nodes in consideration of Byzantine faults such that either all nodes or none of them activate the actuator.

Since the protocol is based on rounds, the nodes have to establish a round based synchronization which can be ensured by a digital clock synchronization built upon our proposed pulse synchronization algorithm.

1.3 Structure of the Thesis

The thesis is structured in the following way: Chapter 2 first gives general insights in the area of distributed systems and clock synchronization. In detail, several properties and design principles of such systems are discussed. This chapter also gives an overview about different types of synchronization which are suitable for establishing a common notion of time among the sub-systems. Chapter 3 presents a modification of the formal message-passing model according to Attiya *et al*. [AW04] in order to be more applicable for WSNs. This chapter also formally defines the self-stabilizing pulse synchronization problem which is the main focus of this thesis. Since clock synchronization is a wide-spread scientific area and contains a lot of different problem definitions and solutions, Chapter 4 presents the most interesting scientific papers related to the topic covered in this work. Chapter 5 then contains the step-wise development of an efficient and robust pulse synchronization algorithm which is applicable in single-hop as well as in multi-hop networks. Simulation results with respect to different network topologies and parameter choices are presented and compared in Chapter 6. A discussion about the outcome of the simulations is given in Chapter 7. The thesis finally ends up in Chapter 8 which reviews the results and presents an outlook of the future work regarding this topic.

CHAPTER 2

Fundamental Concepts

OVERVIEW

This chapter explains several *concepts*, *terms*, *definitions*, and *requirements* for efficient clock synchronization in wireless distributed systems. Note that the semantics of some terms may differ due to the different applications and purposes of clock synchronization and distributed wireless systems. Further, throughout this work the terms *system* or *architecture* are used to refer to a distributed system.

The first part introduces the definitions and design principles of a distributed system with respect to WSNs. Afterwards, the focus lies on clock synchronization and a taxonomy for classifying the algorithms.

2.1 Distributed Systems

There are many definitions for a distributed system which primarily depend on the application area and research field. A loose characterization is given in [TS06] which defines a *distributed system* to be a collection of independent computers that appears to its user as a single coherent system. With respect to WSNs, such a system is characterized as follows: *A distributed system comprises lots of homogeneous and loosely coupled sensor nodes forming a connected communication system and performing a decentralized task in order to maintain a reliable global state based on the unreliable local states of the individual nodes.* Such a decentralized task could be data fusion or simple information gathering in order to control a single or distributed set of actuators based on the reliable representation of the distributed sensor values. Note that the concept of homogeneous nodes is a typical aspect in WSNs and differs from many distributed systems in other application areas, because WSNs generally contain many autonomous nodes of the same type which have the same hardware and software and perform the same tasks. Secondly, loosely coupled nodes are of more importance, since the wireless nodes are mostly battery powered and hence should exchange messages very seldom in order to maintain some kind of energy conservation. The unreliability of communication links in the wireless medium is a further aspect which communication protocols have to take into account. Distributed systems are usually assumed to form a connected communication system. This is reasonable, because unconnected nodes are not able to participate in the synchronization process with the rest of the network. Note that such a communication system can reach from simple fully connected networks to complex networks including many hops which may complicate the communication exchange between two far away nodes. The decentralized task is a representation of the distributed software implemented by the user. The goal of such a software is to provide a reliable service or global state which must be continuously available, even in the presence of different types of network-, communication-, or node-failures. For instance, some nodes may run out of energy and consequently become inactive. This already illustrates why the individual nodes cannot be assumed to be reliable.

Nodes, Communication Systems, Software, and States. Generally, a distributed system consists of multiple, autonomous components, which are called *nodes*. A node consists of 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 behavior 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's 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 to exchange 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. The term *component* or *node* is used to describe a part of the distributed system which cannot be decomposed for a given level of abstraction. In contrast to a com-

ponent, a system can be decomposed into subsystems. A component is characterized by its autonomy (self-containment), the fact that it is used as a building block of a system, and the optional provision of some kind of service to its environment through a well-specified interface [KS02, EPS04]. In this thesis a node is assumed to be composed of a hardware including a processor, I/O interfaces, the software running on it, and the component's state. A component provides a service through its *service interfaces*. The *external state* of a component declares the part of the component's state which is perceivable at the service interfaces. The remaining part corresponds to the *internal state*. Thus, the *behavior* of a component, system, or node is defined by the sequence of its external states.

The following sections describe several orthogonal design principles of distributed systems. Some of them are based on the goals of a distributed system according to Tanenbaum *et al.* [TS06].

2.1.1 Accessibility

A very important property of a distributed system is that the shared state of a node has to be easily accessible for the applications and users. This enables the application to efficiently control the sharing and exchange of its own state. The shared state could be nearly anything of the local state which a node wants to share among other nodes. A typical content contained in a shared state could be the state of the measured entity or the state of an actuator connected to a node. It could also be a fusion of the shared states of several other nodes in order to obtain an agreed value among all participants.

2.1.2 Distribution Transparency

A distributed system which intentionally pretends the applications and users to be a single virtual system despite the fact that its processes and resources are physically distributed is said to be transparent. The International Standards Organization (ISO)'s Reference Model for Open Distributed Processing (RM-ODP) [IEC96, p. 5] identifies eight different types of distribution transparency which should be considered in the specification and implementation of a distributed service in order to hide system complexity and are listed below. A more detailed description can be found in [Put01]. Note that a resource or an object are equivalent definitions for a shared state of some node.

Access transparency masks differences in data representation and hides the user and application from the methodology used to access the shared state. This is especially important in the case of a heterogeneous distributed system where the nodes run different operating systems and thus may have different naming conventions with respect to the file system or the data representation.

Location transparency describes the property of a service to provide an interface for accessing a distant object without the knowledge of its physical location. Instead, logical naming is usually used for binding the interface to the resource.

Migration transparency allows the system to change the location of an object without affecting its access scheme.

Relocation transparency is similar to migration transparency, but stronger in the sense that it allows the system to unnoticeably change the location of a resource while it is in use.

Replication Transparency hides the users or applications from the fact that there exist multiple instances of the same resource used to enhance dependability and/or performance.

Failure transparency masks the failure and possible recovery of one or more resources from the application or user as well from other resources to guarantee a continuous operation. This makes the system resilient to failures in order to enable fault tolerance. Note that this is the major aspect classifying a distributed system.

Persistence transparency hides the fact that node resources needed by an object are not continuously available (*e.g.*, due to scheduling reasons). This is usually done by deactivation and a later reactivation of the objects involved in the interaction with this object.

Transaction transparency maintains the consistency of the objects involved in an overlapping or concurrent execution by masking the scheduling and a possible recovery of the transaction actions.

In [TS06], the last two transparencies are represented by *concurrency transparency*. Concurrency transparency generally describes the masking of the effect that several applications or users compete for the same shared object. This can be implemented by both persistence transparency or transaction transparency.

2.1.3 Openness and Flexibility

Openness is one of the most important architectural aspects of a distributed system and primarily refers to standardized *interface design* such that the services and interfaces offered by a component comply with standard rules that describe the properties of the interface. An interface is a common boundary between two or more components or subsystems. In [KS02], Kopetz *et al*. distinguish between the data and temporal properties of an interface. The *data properties* describe the syntax and semantics of the data crossing the interface. Whereas the syntax is usually formally specified in an Interface Definition Language (IDL), the semantics are documented by means of natural language [TS06]. The *temporal properties* defines the temporal conditions to maintain validity of the messages crossing the interface.

Blair and Stefani state in [BS98] that a complete and neutral interface definition is mandatory for interoperability and portability. *Interoperability* defines the degree to which two systems, implemented by different manufacturers and offering interfaces as specified by a common standard, can work together by using each other's services. The IEEE society defined the term interoperability as "*... the ability of two or more systems or components to exchange information and to use the information that has been exchanged.*" [IEE90]. In contrast, *portability* defines the degree to which a distributed system can be replaced by a different distributed system that implements the same interface without any modification. For comparison with IEEE, portability is defined as "*... the ease with which a system or component can be transferred from one hardware or software environment to another.*" [IEE90]. Referring to [TS06], a careful interface definition is also required to provide an extensible distributed system. *Extensibility* means that the distributed system allows the composition, addition, and replacement of components solely based on their interface definition independent of their realization and implementation of different developers. With respect to IEEE, *extensibility* defines the "*... ease with which a system or component can be modified to increase its storage or functional capacity.*" [IEE90]. A distributed system is defined to be *flexible*, if it provides interoperability, portability, and extensibility. According to IEEE, *flexibility* is also defined as "*... the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.*" [IEE90].

2.1.4 Scalability

In [Hil90], Hill states that there exists no generally-accepted exact definition of scalability. Scalability may be necessary, if a distributed system has to be upgraded or extended to improve some system specific properties or Quality of Service (QoS) (*e.g.*, system load, system complexity, administrative complexity, performance, maintainability, timeliness, reliability, *etc.*). In order to meet such demands, the system must allow a proper handling of new resources or components. According to [Neu94], scalability of a system can be threefold.

1. *Size scalability*. Size scalability ensures that the system allows the addition of new components without increasing system complexity or degrading any other system specific properties to some extent. This type of scalability is the most common aspect of a distributed system and requires that the system is flexible.
2. *Geographical scalability*. This type concerns the scalability of the system with respect to the distance between any two nodes in the system. In other words, the system guarantees to work properly, even if two nodes lie far apart.
3. *Administrative scalability*. This dimension of scalability means that the system keeps manageable even if the number of independent administrative organizations scales up. Examples are network domains with different policies regarding security, resource usage, or management.

Note that scalability does not necessarily mean that it can handle infinitely many components, administrative domains, or components which are infinitely far apart. Instead, if the term scalability is used in some context, it has to be properly defined with respect to some *degradation function* and a *degradation bound*. Examples for the degradation function can be the uncertainty of the communication or the complexity of the system as a function of the network size. As long as the result of the degradation function is below the degradation bound, the system has a good command of scalability. Otherwise, the system may fail or brake down. Note that this does not mean that the bound will eventually be exceeded since the degradation bound can also increase at the same rate like the degradation function. In this case, we speak about a *soft degradation*. Otherwise, if the degradation function can eventually exceed the degradation bound, then we speak about a *hard degradation*. A hard degrading system is usually a system which is designed with respect to some intended maximum scalability (*e.g.*, predefined maximum system size).

A further system distinction can be made with respect to the rate of degradation. For instance, if the degradation function grows at most linearly, we speak about a *graceful degradation*. Otherwise, if the function grows exponentially, we say the degradation is not graceful.

Thus, scalability can be seen as a methodology which guides the design and implementation process [Sat88] and always comes along with a properly definition with respect to the relevant system properties.

2.1.5 Composability

A distributed system is said to be *composable* with respect to a specified property (*e.g.*, accessibility, timeliness, testability, reliability, *etc.*), if the property is maintained at the system level after system integration in the case it was already established at the subsystem level. Note that composability implies that the system has to be open and flexible. Typically, a composable distributed system emerges new services at the system level which were not established at the subsystem level (*e.g.*, synchronization or agreement).

In [KS02], Kopetz *et al*. identify four principles a distributed system has to maintain with respect to the Communication Network Interface (CNI)s and real-time in order to support composability. These are listed below.

1. *Independence of components*: In order to establish the two-level design methodology, the components have to be designed and developed independently based on the precise component service and interface specification in the value and time domain which was done in the architecture-level design process.
2. *Invariance of component services*: Composability requires that the properties provided by a component must be maintained after the integration into an encompassing system or system-of-systems. This has to be considered during the component-level design and is known as the *stability-of-prior-service* principle.
3. *Size scalability*: Composability requires scalability with respect to the performance of the communication system. In detail, a properly working system comprising n components should not be affected after integrating the $n+1^{st}$ component. This may be established by the use of a dynamic resource manager which also has to consider the *critical instant* where all components request for the resource at the same time.
4. *Replica determinism*: This is a sub-principle of scalability in the case fault tolerance is implemented by replication. Then the system has to support *replica determinism*. That is, all correct members of a set of replicated components must produce the same output at virtually the same time with respect to the internal system-time [Pol94].

2.1.6 Dependability and Security

Dependability and Security are two different concepts of a system, but share some common attributes. The definition of *dependability* is twofold [LR04]. Originally, *dependability* defines the ability of a system to deliver some kind of service which can justifiably be trusted. An alternate definition of *dependability* is the ability of a system to avoid service failures to guarantee an acceptable reliability level. The definition and taxonomy of both concepts have emerged over a refinement process of several decades and dates back to 1980. The latest work by Laprie *et al*. at the time of writing this thesis is according to [LR04]. This publication is self-contained and presents a detailed and excellent elaboration of both concepts primarily targeted on students and practitioners of this field. On this account, only the important parts and definitions in order to understand this work and the association with several terms are presented here. In addition, some parts are extended where necessary.

Dependability is a generic concept and subsumes the attributes of availability, reliability, safety, integrity, and maintainability. According to Laprie *et al.*, other concepts similar to dependability are *high confidence*, *survivability*, and *trustworthiness* [LR04]. *Security* focuses mainly on confidentiality and shares the attributes of availability and integrity with the dependability concept. Figure 2.1 visualizes the relationship between the attributes, threats, and means of dependability and security. Note that security encompasses only a few of the listed attributes and the attribute of confidentiality only relates to the security aspect of a distributed system. A detailed description of the previous introduced terms will follow throughout this section.

2.1.7 Self-organization and Emergence

Generally there exist two types of natural systems: unorganized and organized systems. The composability property of distributed systems classifies them as organized systems, since they are usually made up of hierarchically organized building blocks. A system which is organized in a global manner is defined as an *external organized* or *statically organized* system. The opposite are *self-organized* systems.

The best appropriate informal definition with respect to WSNs is stated by Dressler and defines *self-organization* as "*... a process in which structure and functionality (pattern) at*

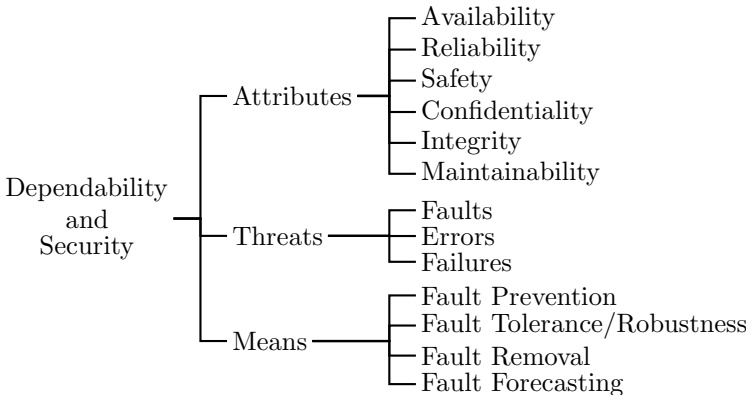


Figure 2.1: Dependability and security tree [LR04].

the higher level of a system emerge solely from numerous interactions among the lower-level components of a system without any external or centralized control. The system's components interact in a local context either by means of direct communication or environmental observations, and, usually without reference to the global pattern." [Dre07].

Simply said, *self-organized systems* are organized in a local manner in order to emerge a coherent global behavior, *i.e.*, the individual components only act with respect to the information of its neighbors.

Furthermore, Dressler defines *emergence* as the provision of a system's behavior "... *by the apparently meaningful collaboration of components (individuals) in order to show capabilities of the overall system (far) beyond the capabilities of the single components*" [Dre07]. Thus, in contrast to self-organization which refers to general emerging patterns at the system level, emergence refers to the appearance of properties (or patterns) at the system level that were not previously observed as a functional characteristic of the system or its components.

The concept of self-organization originates from biology and was first analyzed by Ashby [Ash62]. A detailed discussion on self-organization in biological systems is done by Camazine *et al*. [CFS+03]. Self-organization can be used to achieve robustness and adaptivity to changing environmental conditions [GFH+03]. Furthermore, self-organization can be seen as a collective term for several self-X capabilities (*e.g.*, self-configuration, self-management, self-diagnosis, self-protection, self-healing, self-repair, self-optimization, self-stabilization, *etc.*) [Dre07]. According to [HdM08], the main properties of self-organization are:

- *Autonomy*: The individual components interact only the local information of its neighbors and without any external (global) control.
- *Emergence*: The local interaction among the components emerges new coherent patterns and services at the system level that cannot be understood by simply combining the provided properties and services of the individual components. In other words, "the whole is more than the sum of its parts".
- *Robustness/Adaptivity*: Changing environments do hardly affect the behavior of the system.
- *Decentralization*: The system is not globally controlled by a few entities. Instead it is self-controlled by all components and participants as a result of their interactions.

Self-organized systems reduce the amount of global state information by achieving the system behavior (emergence) based on local interaction or probabilistic approaches only [Dre07]. Thus, self-organized distributed systems can be a remedy to the *complexity crisis* [KC03] of the growing complexity of integrated and networked architectures, especially

if constrained resources (*e.g.*, limited communication bandwidth) prohibit the maintenance of a global state information [Dre07]. In other words, self-organization introduces a new concept for controlling and managing completely autonomous or massively distributed systems beyond the concept of classical distributed systems. This is of particular importance to WSNs which are very large and out of control of humans. On this account, self-organization in WSNs has received a great deal of attention (*e.g.*, for energy efficient protocol design including scheduling, topology discovery and control, and localization).

Formal definition

To the best of our knowledge, the first appropriate formal definition for classifying a system to be self-stabilizing was given by Herrmann *et al*. in [HWM06]. Due to the importance of self-organization to this thesis, this formal model is stated below.

The formal model from Herrmann *et al*. requires the definition of three terms, namely adaptivity, structure, and decentralization.

Herrmann *et al*. use the definition of adaptivity from Zadeh [Zad63] and is formulated as follows: Consider a system $\mathfrak{S}$ with S_γ be the set of all possible time-dependent input functions for $\mathfrak{S}$ where γ defines a parameter set describing its environment. The family of all input functions with respect to a specified set of environmental conditions Γ then corresponds to $\{S_\gamma | \gamma \in \Gamma\}$, in short $\{S_\gamma\}$. Let $P(S_\gamma)$ be a function that measures the performance of $\mathfrak{S}$ with respect to all input functions of S_γ. System $\mathfrak{S}$ is said to *perform acceptably well* under S_γ, if $P(S_\gamma)$ is in a prescribed class W of performance functions: $P(S_\gamma) \in W$. Zadeh then defines adaptivity of system $\mathfrak{S}$ as follows:

Definition 1 (Adaptivity)**.** *A system $\mathfrak{S}$ is adaptive with respect to $\{S_\gamma\}$ and W if it performs acceptably well (i.e., $P(S_\gamma) \in W$), with every source in the family $\{S_\gamma\}$, $\gamma \in \Gamma$. More compactly, $\mathfrak{S}$ is adaptive with respect to Γ and W if it maps Γ into W.*

The structure of a system is a measure for self-organization. However, Herrmann *et al*. state that a concrete definition is not possible since it varies depending on the specific system [HWM06]. Therefore, the authors abstractly define the *structure* of a system to be "*... the property of a system by which it constrains the degrees of freedom of its components.*" [HWM06]. Nevertheless, the authors suggest that in most cases the concept of *entropy* from thermodynamics and from Shannon's theory of information [Sha01] can be used as a measure for structure, since entropy is very similar and describes the measure for the degree of disorder in the system [HWM06]. The definition of information entropy with respect to Shannon is given in Equation 2.1.

$$H(P) = -\sum_{s \in S} P(s) \cdot \log P(s) \tag{2.1}$$

Therein, S belongs to a discrete *state space* (*i.e.*, all possible states) of the system, and P is a probability distribution according to S such that with respect to a single state $s \in S$, $P(s)$ defines the probability that the system remains in it. As a result, the entropy decreases if the system resides in a smaller subspace of its state space (*i.e.*, the system contains some kind of order/structure). In other words, similar to self-stabilization, a self-organizing system usually contains one or more attractors which eventually brings the system into a stable state of structure.

For a formal definition of a decentralized system, Herrmann *et al*. first define a system as follows [HWM06]:

Definition 2 (System)**.** *A system $\mathfrak{S}$ comprises of a set of well-defined interacting components. $\mathfrak{S}$ is identified with the set of its components, and any subset of $\mathfrak{S}$ is called a subsystem. Furthermore, every system $\mathfrak{S}$ has well-defined function $f_{\mathfrak{S}}$.*

Note that the term “software system” is replaced by “system”, since it is more general in the context of this thesis. Furthermore, with respect to the definition of a distributed system, $f_{\mathfrak{S}}$ may correspond to the system behavior as the sequence of external states. With respect to Herrmann *et al.*, decentralization is defined via the absence of a central controller [HWM06]:

Definition 3 (Central controller)**.** *A central controller $\mathfrak{C}$ of a system $\mathfrak{S}$ with respect to $f_{\mathfrak{S}}$ is a subsystem of $\mathfrak{S}$ that controls the actions of the remaining subsystem $\mathfrak{S}' = \mathfrak{S} \setminus \mathfrak{C}$ such that $\mathfrak{S}$ is able to perform $f_{\mathfrak{S}}$ but $\mathfrak{C}$ is unable to perform $f_{\mathfrak{S}}$ in isolation (i.e., without $\mathfrak{S}'$).*

According to Herrmann *et al.*, the classification of a system to be self-organizing requires a precise definition of the system's key elements (*e.g.*, system components, environment, good versus bad system structure). Based on these definitions, the class of self-organizing systems *SO* is defined as follows [HWM06]:

Definition 4 (Class of self-organizing systems)**.** *SO is defined as the class of self-organizing systems and $\overline{SO}$ as its complement. A system $\mathfrak{S}$ is in SO under a description $\mathfrak{D}$ (denoted as $S_{\mathfrak{D}} \in SO$), if $S_{\mathfrak{D}}$ (1) is adaptive, (2) adapts by changing its structure, and (3) does not employ central control.*

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. 45ff].

2.2.1 Concepts of Clocks

In distributed systems, the participants often measure events at a specific point in time. This measurement is done by the use of the node's own local clock which acts independently of each other node. This may be problematic, since most applications and algorithms used in sensor networks are based on the comparison and aggregation of events measured by many different nodes. In other words, every node must have a local view of a global time so that the measured events among different nodes can be reordered in a distinct way. For this concept some new terms must be introduced.

Hardware Clock. Every processor p_k is equipped with a local clock for keeping track of time. This local clock is referred as the *hardware clock*[1] HC_k of processor p_k. Such a clock is implemented as a timer which consists of a *counter* and a physical *oscillator mechanism* that periodically generates events with a well-defined frequency in order to continuously increase the counter. This periodic event is called *microtick* whereas the duration between any two microticks is called *granularity*. The granularity is conditioned by the parameters of the physical oscillator (*e.g.*, oscillator type, frequency, ambient temperature, *etc.*) and is responsible for the digitalization error. The *digitalization error* denotes the difference between the digital clock value and a corresponding continuous-valued reference time.

Hereinafter the time of hardware clock HC_k is called $microtick^k$. Further, microtick i of clock k is denoted by $microtick_i^k$. The time measurement of some event e with the hardware clock HC_k is denoted by $HC_k(e)$.

Reference Clock. A *reference clock* z is a clock with a very small granularity compared to the hardware clock and is usually not observable by the nodes. Hence, in most cases the resulting digitalization error is negligible.

[1] In the literature, the term *physical clock* is often used and equals our definition of hardware clock.

Whenever an event e is timestamped by the reference clock z and z is the single reference clock in the system, then $z(e)$ is called the *absolute timestamp* of event e.

The concept of a 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 practice, a set of n hardware clocks will always slightly drift apart from each other, since the underlying oscillators and crystals cannot provide a perfect and constant nominal frequency.

The drift of a hardware clock HC_k is determined by the ratio between the actually measured duration of the granularity of this clock and the nominal expected number of microticks n^k of reference clock microticks. Since the clock drift varies over time, it is calculated for a distinct granule between microtick i and microtick $i+1$:

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

Hence, a perfect clock has always a drift of 1. In the literature, the drift is sometimes defined with respect to a reference clock z that has the same granularity like the hardware clocks, *i.e.*, for a perfect hardware clock HC_k, the equation $HC_k(microtick_i^z) = i$ is always valid. As a result, the alternative definition of the drift of hardware clock HC_k with respect to z for two natural values $t_2 > t_1$ equals:

$$drift_i^k = \frac{HC_k(microtick_{t_2}^z) - HC_k(microtick_{t_1}^z)}{t_2 - t_1}$$

Note that if it is obvious from the context that t denotes the time value of reference clock z, then we simply write $HC_k(t)$ instead of $HC_k(microtick_t^z)$.

In most cases, the clocks are accurate enough and have a clock drift that is close to 1. For reasons of notational convenience, another term called drift rate is introduced. The *drift rate* of hardware clock HC_k at the instant of microtick i is indicated by ρ_i^k and declares the absolute deviation of the clock drift with respect to the drift of the perfect clock:

$$\rho_i^k = \left| drift_i^k - 1 \right|$$

In reality, the drift rate of a clock results from the drift rate of the underlying oscillator technology and varies over time due to environmental influences (*e.g.*, temperature, vibration, voltage level, aging effects). Furthermore, the amount of uncertainty of the change of the drift rate strongly differs with respect to oscillator technology (*e.g.*, different cuts of a crystal). However, in most cases the environmental influences can be constrained to keep within a specification such that the drift rate of a resonator can be assumed to be bounded by a maximum drift rate ρ_{max}^k. For instance, in the case of a real-time hardware clock, then it can be assumed that the clock stays within a linear envelope of the real-time and is formally defined as follows:

Definition 5 (Bounded drift)**.** *For all times $t_2 > t_1$ and a given maximum drift rate ρ of a hardware clock HC_k,*

$$(t_2 - t_1)(1+\rho)^{-1} \leq HC(t_2) - HC(t_1) \leq (t_2 - t_1)(1+\rho).$$

Note that due to simplification reasons in proofs, the bounded drift condition is sometimes defined as

$$(t_2 - t_1)(1+\rho)^{-1} \leq HC(t_2) - HC(t_1) \leq (t_2 - t_1)(1-\rho)^{-1}$$

or

$$(t_2 - t_1)(1-\rho) \leq HC(t_2) - HC(t_1) \leq (t_2 - t_1)(1+\rho),$$

because both conditions include the original bounded drift condition.

For notational convenience, the drift rate is often quoted in *parts per million* (ppm). A typical watch crystal has about 20ppm that results in an error of about $1.73 sec/day$.

Oscillator Stability 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 two paragraphs describe the drift of crystals and RC-oscillators in more detail.

Stability 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 consequently negligible [Sch96]. It should be noted that this generally applies to oscillators based on crystals, but the clock drift additionally changes due to environmental influences and mainly depends on the oscillator type and cut of the crystal. In [Vig00], Vig categorizes the environmental impacts into time-, temperature-, acceleration-, ionizing radiation-, and other influences. The time can influence the frequency in a short term (*e.g.*, noise) or in a long term (*e.g.*, aging). Short term means in the order of seconds up to minutes. In contrast, long term means in the order of days. 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].

According to Armengaud *et al*. [ASH07] and to Sullivan *et al*. [SAHW90], the instantaneous drift rate of node's p crystal oscillator can be modeled as shown in Equation 2.2.

$$\rho_p(t) = \rho_p^i + \rho_p^a(t) + \rho_p^n(t) + \rho_p^e(t) \tag{2.2}$$

Therein, ρ_p^i denotes the initial drift rate at start-time which is assumed to be constant over time. $\rho_p^a(t)$ incorporates the drift variation due to aging effects, $\rho_p^n(t)$ considers the drift rate jitter due to short term noise (*e.g.*, radiation), and $\rho_p^e(t)$ adds the jitter resulting from other environmental effects (*e.g.*, temperature). Table 2.1 lists the value range with respect to the different drift parts according to [Sch95, Com97]. These values show that the initial drift dominates the other parts, especially if small time durations in the order of seconds and a nearly constant environment (*e.g.*, no abrupt strong temperature variations) are considered. These assumptions usually hold in most cases and can be formalized in a bounded-drift-variation model as stated in Definition 6.

Definition 6 (Bounded drift variation)**.** *The amount of instantaneous drift rate variation $d\rho(t)/dt$ of a clock for all t is bounded according to*

$$\left| \frac{d\rho(t)}{dt} \right| \leq \vartheta,$$

where ϑ is the oscillator stability.

Table 2.1: The range of the different drift rate components [Sch95, Com97].

Component	Drift rate
ρ_p^i	up to 10^{-5}
$\rho_p^a(t)$	up to 10^{-7}
$\rho_p^n(t)$	$10^{-8} \cdots 10^{-12}$
$\rho_p^e(t)$	up to 10^{-5}

A more established methodology for defining the short-term stability of an oscillator in the time domain is expressed by the use of the Allan variance [iee99] and is defined in Equation 2.3:

$$\sigma_y^2(\tau) = \frac{1}{2(N-2)\tau^2} \sum_{k=1}^{N-2} (x_{k+2} - 2x_{k+1} + x_k)^2, \tag{2.3}$$

where

N	is the number of time measurements
τ	is the nominal sample time
$x_k, x_{k+1}, \ldots$	are time residual measurements after removing systematic effects (*e.g.*, frequency drift) at $t_{k+i} = t_k + \tau$ for all $k \geq 1$.

Note that the removing of the systematic effects is required in order to get an unbiased variance $\sigma_y^2(\tau)$.

Table 2.2 lists typical Allan deviations for different oscillator technologies according to [Kli97].

Table 2.2: Typical Allan deviations for different oscillator technologies [Kli97].

	$\tau = 1$ *sec*	$\tau = 1$ *day*	$\tau = 1$ *month*
Quartz	10^{-12}	10^{-9}	10^{-8}
Rubidium	10^{-11}	$10^{-12} \ldots 10^{-13}$	$10^{-11} \ldots 10^{-12}$
Cesium Beam	$10^{-10} \ldots 10^{-11}$	$10^{-13} \ldots 10^{-14}$	$10^{-13} \ldots 10^{-14}$
Hydrogen Maser	10^{-13}	$10^{-14} \ldots 10^{-15}$	10^{-13}

Stability of RC-oscillators. RC-Resonators are generally used for internal oscillators in controllers. Due to the fact that it is cheaper to use the implemented oscillator than to extend the controller with an external crystal oscillator, many applications for embedded systems not aimed at high clock accuracy make use of the internal RC-oscillator for generating the system clock and consequently the microticks of the local hardware 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.

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 latter condition is usually no problem whereas the change of the frequency of an oscillator seems to be a problem for so, because many controllers are equipped with 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 by the use of virtual clocks which abstract from hardware dependent parameters. Throughout this thesis the term *local clock* or *logical clock* is sometimes used instead of virtual clock.

A processor p_j is said to implement a virtual clock VC_j, if it abstracts the hardware clock HC_j. A realization of such a virtual clock providing the two adjustment criteria is based on the concept of macroticks. A *macrotick* is represented by a tick of the virtual clock and comprises of a number of microticks that are generated by a hardware clock. Let $T_j^{th}(t)$ denote this amount of microticks at p_j. In other words, $T_j^{th}(t)$ can be seen as the actual threshold value at time t. Any software implemented on p_j then reads the time only from VC_j. As a consequence, the granularity of the virtual clock is based on the number and granularity of the comprised microticks.

By adjusting $T_j^{th}(t)$ over time, the granularity and consequently the time duration of one macrotick can be increased or decreased. Let T_{nom} denote the *nominal threshold level* and $H_j(t)$ the *absolute adjustment value* such that $T_j^{th}(t) = T_{nom} + H_j(t)$. The corresponding *relative adjustment value* is $h_j(t) = \frac{H_j(t)}{T_{nom}}$. Algorithm 1 shows the principle of this virtual clock. Therein, HC_j is a variable which stores the content of the last invoked macrotick.

Algorithm 1: Virtual Clock $VC_j(t)$: code for p_j

1 **Init:** $HC_j := HC_j(t)$, $VC_j(t) := 0$, $T_j^{th}(t) := T_{nom}$

2 **upon event** $HC_j(t) = HC_j + T_j^{th}(t)$ **do** `// threshold reached`

3 $\quad HC_j := HC_j + T_j^{th}(t)$

4 $\quad$ adjust $H_j(t)$

5 $\quad T_j^{th}(t) := T_{nom} + H_j(t)\ VC_j(t) := VC_j(t) + 1$

6 Periodically increment $HC_j(t)$ according to the oscillator frequency and actual drift

In practice, the implementation of such a virtual clock is simply realized by a counter register that is continuously incremented with respect to the microticks of the underlying hardware clock. In every incrementation event, the value of the timer or counter is then compared with respect to the actual threshold value T_j^{th}. If the comparison matches, then the register will be reset to 0 and a macrotick incrementation is invoked.

Offset. The *offset* is defined as the time difference of the clocks[2] of two different processors p_j and p_k at the microtick respectively macrotick i measured with respect to the reference clock z.

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

In the case of virtual clocks, we would have

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

Precision. In an ensemble of clocks, the *precision* defines the maximum offset in reference clock microticks between any two clocks during a period of interest. So the precision of an ensemble with n clocks over all interesting microticks i is defined as

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

[2]The clocks can be either virtual or hardware clocks.

or with respect to virtual clocks as

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

Note that the real-time is usually used as the reference clock for measuring the precision.

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

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

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

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

Note that a good accuracy implies a good precision, but from a given precision nothing can be said about the accuracy.

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 time-stamping 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 sensor networks as this will increase the communication and therefore also strongly increases the energy consumption. Consequently, alternative mechanisms like clock rate calibration must be incorporated. 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. This type of synchronization is also sometimes referred to *synchronicity*. The duration for a period of resynchronization is called *resynchronization interval* R_{int}. After a resynchronization event, the clocks run free and may drift apart with respect to a maximum drift rate ρ as stated in Definition 5. 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 z can be $\pm\rho \cdot R_{int}$. Therefore, a new term called *drift offset* Γ is introduced with $\Gamma = 2 \cdot \rho \cdot R_{int}$.

The Synchronization Condition. Due to uncertainties in communication delay and clock drift, the synchronization is usually bounded within a small interval and affects the precision. Further, convergence-based synchronization algorithms [Sch87] additionally cannot synchronize the nodes better as expressed by its convergence function Θ. The *synchronization condition* then states that such a synchronization algorithm can only synchronize the ensemble of nodes with precision Π, if the following equation holds:

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

In other words, for a given Θ and Γ, the worst case precision cannot be better than $\Theta + \Gamma$.

Lower Bounds for Internal Clock Synchronization. Any *convergence function based synchronization algorithm*[Sch87] 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 [LL84b] and depends on the number of clocks N and the communication jitter ε. Assuming the clocks have no clock drift, Lundelius and Lynch state that the worst case precision in an ensemble of clocks is lower bounded to

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

According to the aforementioned synchronization condition, we can deduce that a convergence based synchronization algorithm cannot have a convergence function better than $\Theta < \varepsilon \cdot (N-1)/N$. Thus, the worst case synchronization precision is limited to $\varepsilon \cdot (N-1)/N + \Gamma \leq \Pi$.

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. Considering 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 periodic resynchronization of the clocks with respect to the reference clock and keeps the ensemble within a bounded precision 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 between two consecutive macrotick of a virtual clock or in the case of a VCXO by adjusting the supplied voltage. However, in the case of an implemented rate correction but in the absence of an external clock synchronization, it might occur that the drift rate of all virtual clocks continuously increase or decrease in the same way, because the uncertainty in communication delay makes a perfect rate adjustment impossible. This effect is known as the *common mode drift*. To avoid this common drift, the rate correction algorithm should incorporate a compensation to control this effect. In the case of a simple fully connected network, this can be done by an evaluation of the average rate correction terms among all clocks which should be close to zero. However, in the general case where the network topology is not known *a priori*, the effect of the common mode drift usually can only be reduced to some extent but not completely eliminated.

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. This is done by the use of the aforementioned virtual clock. However, in many cases it must be guaranteed that the correction term is bounded by some maximum value. This is usually formalized in a worst case correction term.

2.2.3 Pulse Synchronization

In contrast to the previously investigated traditional clock synchronization, *pulse synchronization* refers to the internal synchronization of periodic time intervals instead of a continuously increasing time such that all nodes invoke the pulse indicating the beginning respectively ending of the time period together *Cycle* time apart. In other words, all nodes invoke pulses within a short interval that refers to the precision of the pulse synchronization and afterwards wait for about a Cycle time before invoking again a pulse. In the literature, this type of synchronization is sometimes referred to as tick or beat synchronization.

The cycle concept used by the nodes is established via the concept of pulse clocks. Therefore, we denote the *pulse clock* of processor p_j by $PC_j(t)$ which abstracts the hardware clock $HC_j(t)$ in the following way:

Algorithm 2: Pulse Clock $PC_j(t)$: code for p_j

1 **Init:** $HC_j := HC_j(t)$, $T_j^{th}(t) := T_{nom}$

2 **upon event** $HC_j(t) = HC_j + T_j^{th}(t)$ **do** `// threshold reached`

3 $\quad HC_j := HC_j + T_j^{th}(t)$

4 $\quad$ adjust $H_j(t)$

5 $\quad T_j^{th}(t) := T_{nom} + H_j(t)$

6 $\quad$ **invoke** pulse

7 $PC_j(t) = \left\lfloor \Phi_{th} \cdot (HC_j(t) - HC_j)/T_j^{th}(t) \right\rfloor$

8 Periodically increment $HC_j(t)$ according to the oscillator frequency and actual drift

In other words, the time duration between two consecutive pulses equals $T_j^{th}(t)$ microticks. The constant T_{nom} defines the nominal number of microticks which represents the cycle time. However, in many situations it is necessary to adjust the time duration of a cycle to some extent. On this account, the variable $H_j(t)$ represents an adjustable element which can be externally set.

Note that the value domain of a pulse clock should always be the same with respect to some predefined *threshold value* Φ_{th}. That is, the pulse clock is periodically cycling from 0 up to a predefined $\Phi_{th} - 1$ and can be formally expressed as

$$PC_j(t) \equiv \left\lfloor \Phi_{th} \cdot \frac{HC_j(t) - HC_j}{T_j^{th}(t)} \right\rfloor, \tag{2.4}$$

where HC_j stores the number of microticks of the last pulse invocation.

Since many embedded systems do not support floating point operations, the calculation of Equation 2.4 incorporates a digitalization error which increases if Φ_{th} decreases. For the rest of this paper, Φ_{th} is assumed to be large enough such that the digitalization error is negligible. On this account, a continuous *phase clock* can be supposed. Such a clock is characterized by the fact that it provides a normalized *phase variable* $\varphi_i = PC_j(t)/\Phi_{th}$ which is defined in the range $\varphi_i \in [0,1)$. Definition 7 gives a formal description of this normalized pulse clocks.

Definition 7 (Phase clock)**.** *The phase clock $\varphi_j(t)$ of processor p_j is a phase variable that has the following properties:*

1. $\varphi_j(t) \in [0,1)$,
2. $d\varphi_j(t)/dt = 1/T$, *where T denotes the cycle period and*
3. $\varphi_j = 0$ *at the beginning of a cycle.*

To give a formal definition of pulse synchronization, a definition of the pulse state is required and presented below according to [DD08]:

Definition 8 (Pulse state)**.** *The pulse state $P(t)$ of a system comprising of n nodes at real time t is defined as*

$$P(t) = (\varphi_0(t), \varphi_1(t), \ldots, \varphi_{n-1}(t)).$$

The formal definition for a pulse-synchronized set of nodes in the phase domain is given in Definition 9.

Definition 9 (Pulse synchronization - Phase domain)**.** *A set of nodes N is called pulse-synchronized for a given maximum phase deviation Φ_Π at real time t, if for all nodes $p_i, p_j \in N$, either $|\varphi_j(t) - \varphi_i(t)| \leq \Phi_\Pi$, or $|\varphi_j(t) - \varphi_i(t)| \geq 1 - \Phi_\Pi$.*

Similarly to [DD08], the term of a *synchronized pulse state* is introduced in Definition 10. It should be noted that in the case that all nodes in a system are pulse-synchronized, we say that the system has achieved *synchronicity*. This state corresponds to the term of internal synchronization. In contrast, *time synchronization* refers to both the previously introduced term of external synchronization and internal synchronization.

Definition 10 (Synchronized pulse state)**.** *Let G be the set of all possible pulse states of a system. Then $P \in G$ is a synchronized pulse state of the system at real time t, if the set of correct nodes is pulse-synchronized at real time t.*

However, a more interesting definition belongs to the maximum allowed deviation in real time between a set of pulse synchronized nodes:

Definition 11 (Pulse synchronization - Time domain)**.** *A set of nodes N is called pulse-synchronized with precision Π till real time t_0, if for all nodes $p_i, p_j \in N$ and $t \geq t_0$, p_i and p_j are synchronized according to Definition 9 for some given Φ_Π, and there exists some t' with $|t - t'| \leq \Pi$ such that either $\varphi_i(t) = \varphi_j(t')$, or $\varphi_i(t') = \varphi_j(t)$.*

2.2.4 Clock Desynchronization

Clock Desynchronization is the logical opposite of synchronization and is an interesting and novel approach, first published by Degesys *et al*. [DRPN07, PDN07, DN08]. Therein, the authors propose a self-stabilizing algorithm DESYNC for periodic resource sharing without the necessity of a global clock. In detail, with respect to the previously defined pulse clocks, the algorithm guarantees that the nodes eventually invoke their pulses as far away as possible from all other nodes, *i.e.*, a set of n nodes pulse at evenly spaced time intervals (T/n) throughout the time period. Furthermore, the algorithm automatically adapts to the actual number of participating nodes in the system. This can be used for a collision-free TDMA communication for transmitting messages in a broadcast medium (similar to a round-robin schedule), or the organization of sleep cycles for energy reduction. However, convergence has been proven only for fully connected networks, since multi-hop networks may suffer from the hidden-terminal problem.

Since the clock desynchronization primarily addresses the desynchronization of pulse clocks, the term *pulse desynchronization* is used instead. With respect to the definition of pulse state (Definition 8), pulse desynchronization is formally defined as follows:

Definition 12 (Pulse desynchronization)**.** *A set of nodes $N = \{p_1, p_2, \ldots, p_n\}$ is called pulse desynchronized at real time t, if for all nodes p_{i_k}, $1 \leq k < n$, with $\varphi_{i_k}(t) \leq \varphi_{i_{k+1}}(t)$, the following holds:*

$$\varphi_{i_{k+1}}(t) - \varphi_{i_k}(t) = 1 - \varphi_{i_n}(t) + \varphi_{i_1}(t) = 1/n.$$

Definition 13 (Desynchronized pulse state)**.** *Let G be the set of all possible pulse states of a system. Then $P \in G$ is a desynchronized pulse state of the system at real time t, if the set of nodes is pulse desynchronized at real time t.*

Clearly, due to the presence of drift and other influences, the set of nodes of a system will only approximate the desynchronized pulse state.

2.2.5 Digital Clock Synchronization

Digital clock synchronization assumes an underlying pulse synchronization as an established basic-level concept such that the nodes can execute in lock-step rounds by periodically receiving a common pulse. For this, each node p_i has an integer variable $p_i.clock$ which counts

the pulses (or beats) of the underlying pulse model. Digital clock synchronization then aims at synchronizing all nodes with respect to their clock variables. In the literature, this type of synchronization is sometimes referred to as round or slot number synchronization. For the sake of simplicity, if it is clear from context, this type of synchronization is always simply denote as *clock synchronization*,

According to Ben-Or *et al.*, a clock synchronized system is defined with respect to the pulse number as follows [BODH08]:

Definition 14 (Digital clock-synched state)**.** *A system is digitally clock-synched at pulse r with value $Clock(r)$, if at the end of beat r, all nodes have the same clock value, and it is equal to $Clock(r)$.*

Since the counter variable is usually bounded either due to the limited bit width or due to a limited value domain, the digital clock synchronization problem is expressed with respect to a modulo operation [BODH08].

Definition 15 (Digital k-Clock synchronization)**.** *The digital k-Clock synchronization problem consists of*

1. *(Convergence) starting from any state, eventually the system becomes clock-synched with value $Clock(r)$, and*
2. *(Closure) from this point on the system stays clock-synched such that at beat $r+i$ it is clock-synched with value $Clock(r)+i \pmod k$.*

2.2.6 Threats to Clock Synchronization in WSNs

The only attacks considered so far proceed from the assumption of faulty nodes that can fail arbitrarily. The attacker models presented below are especially aimed at degrading the time synchronization in wireless networks. Since clock synchronization is usually achieved through the exchange of time-sensitive messages, it is easy for an adversary to forge and modify these messages such that the time synchronization degrades or even fails. Different aspects, effects, and countermeasures against time synchronization attacks are stated in [MRS05]. Generally, the attacks can be classified into internal and external attacks.

External Attacks

In contrast to internal attackers, an external attacker does not compromise the nodes or pretends the other network nodes to be a correct node. Instead, they exploit several features of the easily accessible communication medium. In other words, the content of the exchanged messages are not modified, but the messages itself may be delayed or destroyed. According to Ganeriwal *et al*. [GČHS05], an external attack can affect time-sensitive messages in the following three ways:

1. The attacker modifies the content of the message.
2. The attacker forges or replays messages.
3. The attacker delays the reception of messages, also known as the *pulse-delay attack*.

The first two attacks usually affect the integrity of the message and therefore can be detected by security primitives. However, the presented algorithms in this thesis do not assume the existence of such primitives. In contrast to the other attacks, the pulse-delay attack is hard to treat, especially in the case of multi-hop topologies.

Pulse-delay attacks. The pulse-delay attack concerns the delay attack of time-sensitive messages and was first described by Ganeriwal *et al*. [GČHS05]. This attack proceeds in three steps.

First, the attacker *eavesdrop* the communication channel for a new message transmission. If so, then he buffers a copy of this message. The second phase is performed meanwhile the first step. Therein the attacker simply *jams* the communication channel during the transmission of the time-sensitive message such that the reception at the receiver is prevented. For the sake of completeness, it is assumed that the jamming cannot be detected at both the sender or the receiver. Ganeriwal *et al*. state that the radio transmitting techniques of Direct-sequence Spread Spectrum (DSSS) and Frequency-hopping Spread Spectrum (FHSS), both used in many available sensor network platforms, are still vulnerable to broadband jamming. In the third and last step, the attacker *replays* the buffered copy of the synchronization message resulting in a delayed reception.

Wormhole attacks. Wormhole attacks and defenses against these attacks were analyzed by Hu *et al*. [HPJ01, HPJ03]. A wormhole attack is a serious threat in multi-hop wireless networks and is performed in the following three steps:

1. The attacker first gathers the packets or a part of them at one location in the network,
2. then tunnels them to another (possible far away) location,
3. and retransmits the packets or forwards the bits there into the network.

The tunneled transmission is usually a low latency and high bandwidth communication channel that spans over several hops of the normal wireless transmission range. The transmission over such a channel leads to a sooner reception of the packets at the destination. However, such "wormholes" may also delay or even drop the time-sensitive packets.

Wormhole attacks are especially problematic for establishing routing protocols in ad-hoc multi-hop networks, since a node may wrongly assume that it is in the transmission range of some other node. Hu *et al*. proposed a defense against wormhole attacks, called *packet leashes*. A leash is the additional information added to the packets in order to restrict the transmission distance. They distinguish between geographical and temporal leashes. The *geographical leash* restricts the distance of packet transmission. The *temporal leash* restricts the lifetime and consequently again the communication distance of a packet. Based on the leash information, the receiver is able to detect if the communication delay of a packet is smaller than the leash allows. However, such an approach requires synchronized clocks which again illustrates the importance of clock synchronization in wireless networks.

Denial-of-Service (DoS) Attack. The most critical attack in WSNs is radio *jamming*. This means that the attacker continuously transmits unwanted and disruptive packets such that he occupies the full range of communication bandwidth. As a result, the concerned sensor nodes in that region are unable to communicate with each other. This attacker model is sometimes referred to as a *babbling idiot* failure of the concerned sensor node. However, the attacker can also be a malfunction of a device which suffered from, for example, strong radiation resulting in an erroneous state and consequently in such a misbehavior. The size of the affected region heavily depends on the available power, height above ground, antenna design, and obstacles [WSS03] in front of the attacker. The most serious problem of jamming attacks is that either nothing or only with prohibitive amount of cost and complexity can be done to avoid or prevent the network against it.

On this account Wood *et al*. elaborated an alternative approach which exploits that fact that WSNs usually yield large redundant multi-hop networks due to the small radio range of an individual sensor node or attacker with respect to the large size of the network distribution [WSS03]. In detail, the network tries to obtain the information about the location and shape of the jammed region and consequently excludes this area from routing and other communication dependent functions. This is done by the use of a jamming detection and mapping protocol. In detail, nodes within the border of a jammed region notify the outside

world of this dilemma. The receivers outside this regions the form groups which collaborate with each other such that the affected region is shielded from the rest of the world.

Internal Attacks

An internal attacker may takeover a certain number of sensor nodes in the sense that it knows all secret keys and consequently is able to completely control these nodes. These nodes then are called *compromised nodes*. As a result, compromised nodes appear as legitimate nodes to the other ones and can forge, modify, or eavesdrop messages. Note that a non-compromised node may also misbehave due to the cause of faults or environmental effects and a subsequent error in the system state. Such nodes are also dedicated as internal attacks. Generally, internal attacks ca be classified according to the following fault types:

Definition 16 (Crash fault)**.** *A node suffers from a crash fault, if it stops executing the algorithm at some time.*

Definition 17 (Omission fault)**.** *A node suffers from an omission fault, if it randomly omits the transmission of messages.*

Note that for sake of simplicity, message losses due to environmental influences and interferences are assumed to result from omission faults at the origins although the sender is non-faulty.

Definition 18 (Timer fault)**.** *A node suffers from a timer fault, if it does not satisfy the bounded drift assumption of Definition 5. The node transmits messages with a minimum inter-transmission time in order to prohibit radio jamming.*

Definition 19 (Byzantine fault)**.** *A node suffers from a Byzantine fault, if it behaves arbitrarily, i.e., the receivers may receive messages with different arbitrary contents. Nodes suffering from Byzantine faults can also collude to perform the worst case damage. The node transmits messages with a minimum inter-transmission time in order to prohibit radio jamming.*

Note that the assumption of a minimum inter-transmission time as stated in Definition 18 and Definition 19 can be achieved by the use of a hardware implemented guardian. In the case of Byzantine faults in a broadcast medium, the receivers usually receive the same possible faulty message. On this account, an additional fault type must be defined:

Definition 20 (Consistent fault)**.** *A node suffers from a consistent fault, if it behaves Byzantine, but the receivers either receive the same message or no message at all.*

Sybil attacks. A compromised node may present multiple identities in order to undermine the redundancy mechanisms. Douceur first analyzed such an attack in the context of peer-to-peer networks and named it *Sybil attack* [Dou02]. He showed that it is impossible to defeat a Sybil attack without the existence of a trusted and logically centralized authority that certifies identities. Later, Newsome *et al*. elaborated Sybil attacks in WSNs in more detail and proposed a taxonomy of the different forms of the Sybil attack [NSSP04]. They further elaborated a promising approach against the Sybil attack named *random key distribution* which associates the key of a node with its identity.

2.2.7 Classification of Clock Synchronization Algorithms

During the last few decades, an enormous amount of more than hundred clock synchronization algorithms for distributed systems and WSNs, respectively, were proposed. However, only during the last few years more emphasis was put on clock synchronization in

wireless networks since traditional algorithms designed for wired systems do not provide the robustness especially required in WSNs. Typical aspects of WSNs which have to be considered are the limited energy of battery-powered devices, the limited bandwidth of the broadcast medium, and especially the unstable network connection (*e.g.*, high population of nodes, small communication range, hidden terminal problem, interference problems, multi-hop networks, changing topologies in mobile ad-hoc networks, omission failures, high communication jitter, *etc.*). The resulting large variety of synchronization algorithms makes the identification and selection for a specific application without an appropriate classification technique nearly impossible. On this account, a well suited taxonomy that covers all these algorithms is inevitable. Therefore, the classification techniques according to Sundararaman *et al*. [SBK05] and Römer *et al*. [RBM05] are combined with our own contribution and is presented in Figure 2.2.

It should be noted that the presented taxonomy primarily addresses the synchronization of clocks with respect to physical time (*i.e.*, external clock synchronization) and does not consider algorithms which are aimed solely on the internal synchronization for providing synchronized logical clocks. Further note that this thesis presents a scheme for pulse synchronization without the approximation to an external time reference. However, several aspects of the classification methodologies also fit to purely internal (logical) clock synchronization (*e.g.*, pulse synchronization or digital clock synchronization).

According to Sundararaman *et al.*, a synchronization protocol can be classified with respect to the *synchronization issues* and *application-dependent features* [SBK05]. The different classification entities according to these aspects are illustrated in Figure 2.2.

Asymmetric versus Symmetric

Asymmetric clock synchronization algorithms require the existence of dedicated nodes that serve as synchronization masters or play a different role than other nodes. A typical example is the master-slave clock synchronization which corresponds to the client-server principle. Therein, a dedicated node acts as the master (server) that provides the reference time and the others as the slaves (clients). The slave then synchronizes their clocks according to the master. Note that this centralized approach implies several problems for WSNs and are already discussed within the distribution transparencies. This approach is sometimes referred to as *server-based* or *centralized* clock synchronization.

Within a *symmetric clock synchronization* scheme, each processor executes the same algorithm and consequently plays the same role. A typical symmetric clock synchronization algorithm is a *peer-to-peer clock synchronization* protocol, where each node communicates with each other in order to establish clock synchronization. This prevents the existence of a single point of failure as it would be in the case of a master-slave synchronization fashion. Furthermore, the peer-to-peer scheme usually provides more flexibility, adaptivity, and robustness which are important requirements in WSNs. This approach is also known as *server-less* or *distributed* clock synchronization.

Internal versus External

These two aspects of a synchronization algorithm were already introduced in the previous section. In short, *internal clock synchronization* aims at reducing the minimizing the precision among the participating clocks.

In contrast, *external clock synchronization* requires the presence of an external standard source of time such as Universal Time Coordinated (UTC) that provides a reference time. The nodes then have to synchronize according to this reference time such that the accuracy with respect to this reference time and consequently the precision is minimized. Note

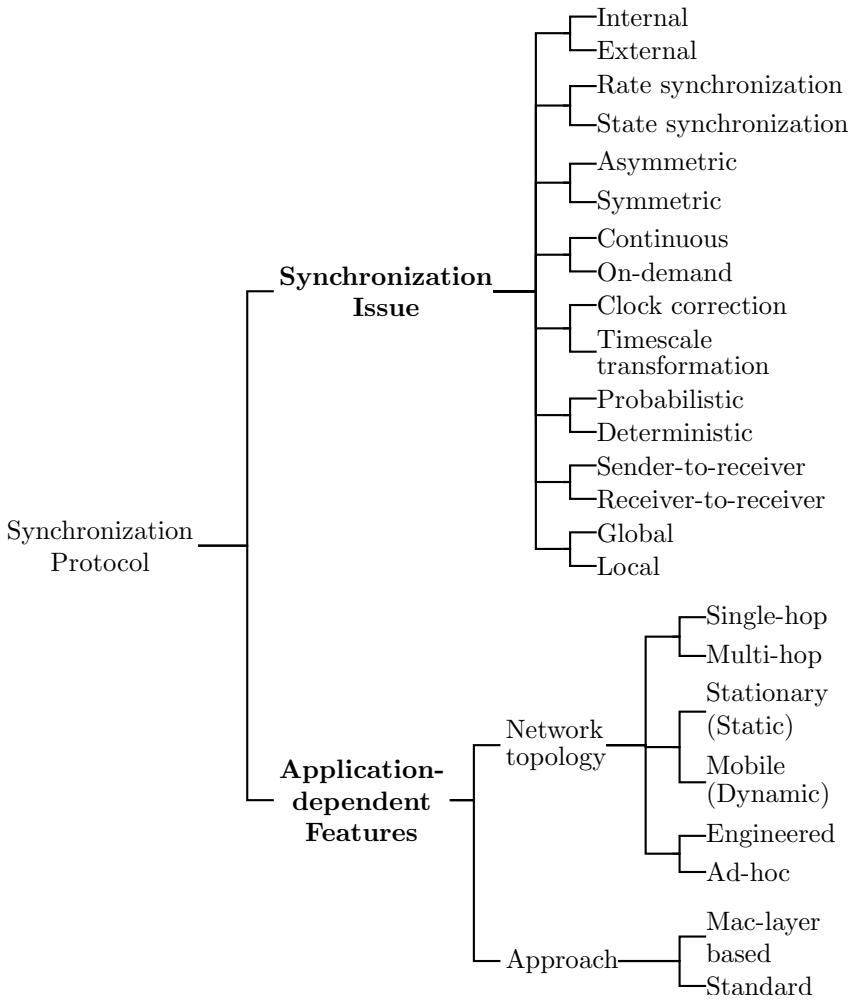


Figure 2.2: Classification of clock synchronization protocols in WSNs.

that external synchronization implies that the synchronization is performed in a master-slave fashion.

Rate versus State

Rate synchronization concerns the calibration of the clock frequency and consequently its clock drift in order to establish an approximate agreement with respect to the clock drift among all participating nodes. As a result, all clocks measure the same interval duration between the global appearance and disappearance of an event. Note that rate synchronization is not necessary for establishing a simple logical time synchronization like the Lamport's happens-before relation.

Contrary, *state synchronization* belongs to the agreement of all nodes' clock values such that all clocks take the same value at some specific time instant. This is necessary for combining the temporal relation between the observance of events at different nodes.

Continuous versus On-demand

Continuous and on-demand clock synchronization belong to the lifetime aspect of a clock synchronization protocol during which the condition of synchronized clocks is feasible. In classical wired distributed system, the *continuous* approach is a well established method.

Therein, the nodes are continuously, usually in a periodic manner, resynchronized such that the synchronization precision among all clocks can be bounded to some value. The periodic resynchronization approach is sometimes referred to *proactive synchronization*.

In contrast, *on-demand synchronization* or *reactive synchronization* does not require the clocks to be periodically synchronized. Instead, the clocks are free-running until the recognition of some event. This strongly reduces the communication complexity and consequently energy consumption. According to Römer *et al*. [RBM05], there exist two different kinds of on-demand synchronization, namely event-triggered and time-triggered:

- *Event-triggered* on-demand synchronization exploits the fact that synchronization is only required immediately after the recognition of an event. In other words, the occurrence of an event is time-stamped, and afterwards triggers a resynchronization point.The real time-stamp with respect to the other clocks can then be back calculated after synchronization is completed. *Post facto synchronization* is a typical representative of this synchronization scheme [EGE02].
- *Time-triggered* on-demand synchronization is also known as *pre facto synchronization* and is used if an event (*e.g.*, sensing or actuating) has to be triggered simultaneously at many different nodes. This can be either performed immediately where the nodes are globally triggered by the reception of a special message, or via *anticipated synchronization*. The latter type requires that the nodes are synchronized such that they autonomously perform the dedicated task at the same time instant somewhere in the future and in the absence of a globally visible triggered event. Anticipated synchronization is especially important in networks where such nodes are several hops away from the other nodes.

Clock Correction versus Timescale Transformation

Clock synchronization can be performed in two different ways. *Clock correction* is based on the adjustment of the local clocks according to the clock state and/or the clock rate. The local clocks can be corrected either on-demand or continuously. According to [AP98] and [RSB90], clock state correction methods are based on the obtained clock estimates of the neighboring nodes and can be divided into three general groups as stated below. Note that the clock estimates may be either performed through broadcasting, remote clock reading, or other techniques. In the case a history of several consecutive clock readings are available, a node may also perform *linear regression* [EGE02] or simple averaging to estimate a neighbors clock more precisely.

- *Convergence-averaging algorithms*: Algorithms based on this method apply a convergence function on a set of clock estimates of the neighboring nodes. The node then adjusts the local clock to the result of the function. As the name implies, the convergence function calculates some kind of averaging (*e.g.*, midpoint, median, mean, *etc.*). In [Sch87], Schneider gives an overview of several fault-tolerant convergence functions developed before 1987 that additionally tolerate Byzantine faults. Generally, resilience against up to f malicious faults is provided by excluding the f lowest and f highest values in the set of clock estimates before applying the convergence function. Another technique is based on discarding those clock estimates that exceed some threshold. An overview of other convergence functions is also presented in [AP98].
- *Convergence-non-averaging algorithms*: According to [RSB90], this technqiue corresponds to discrete-update algorithms where all nodes periodically synchronize to a single synchronizer. However, in contrast to a master-slave principle, all nodes seek to become this single synchronizer. This election mechanism may be established through a *leader election algorithm*. Alternatively, the nodes may initially agree on a common

schedule such that each node has the same probability to become the synchronizer. Fault tolerance is usually achieved through detection of the faulty synchronizer and a consequent re-election of a new synchronizer. In other words, in the case the synchronizer is faulty, the other nodes take over and synchronize without the faulty node.

- *Consistency algorithms*: Consistency algorithms establish agreement among the clock values of all nodes by using an *interactive consistency algorithm* [LSP82]. In the presence of Byzantine nodes, this algorithm ensures that each non-faulty node has the same consistent (possible faulty) view of all other nodes in the network. A node then usually selects the median from the consistent set to be the new clock value. However, in contrast to the convergence based algorithms, the consistency algorithms have much more overhead with respect to the information exchanges.

In contrast, the *timescale transformation* approach establishes a common notion of time by leaving the clocks untethered and transforming the local time of one node into local times of other nodes if required (*e.g.*, for reasons of comparison). This is usually done by maintaining a table which contains the transformation values for each participating node in the network. Event-triggered on-demand clock synchronization can be grouped into timescale transformation techniques.

Probabilistic versus Deterministic

As the name implies, *probabilistic clock synchronization* algorithms provide a probabilistic guarantee on the worst case synchronization precision respectively accuracy. Probabilistic algorithms usually require fewer message transmission. This is also the reason why most wireless protocols are of this type.

The deterministic upper bound of the synchronization precision in *deterministic clock synchronization* protocols are of peculiar interest in distributed real-time systems.

Sender-to-receiver versus Receiver-to-receiver

Sender-to-receiver clock synchronization is a standard approach which generally is performed in three steps:

1. A receiver requests a timestamp from the sender.
2. The sender sends a time-stamped message back to the receiver.
3. Based on the received time-stamp, the receiver synchronizes its clock to the sender and additionally calculates the end-to-end delay by measuring the round-trip time.

A disadvantage of this method is that the communication jitter in wireless networks is usually very high. As a result, the calculation of the round-trip time usually has to be based on the average of several measurements which consequently implies an increase in the message overhead. The sender-to-receiver approach can be sub-classified into one-way dissemination and two-way exchange.

- *One-way dissemination* allows the transmission of time information only from the synchronized nodes to the unsynchronized nodes and not conversely. However, one-way synchronization schemes are incapable of handling external attacks like pulse-delay attacks [GPČS08].
- *Two-way exchange* based algorithms synchronizes the nodes by exchanging packets between synchronized and unsynchronized nodes.

In the presence of multi-hop networks with high propagation delay, one-way protocols usually result in high average synchronization error and the two-way protocols result in high variance but low average error [HDQK09]. It should be noted that some synchronization algorithms rely on a *N-way exchange* such that both the initiator node and the receiver node

can synchronize their clocks according to a recorded set of up to N time samples. A typical example for such an approach is the SPS-SE protocol proposed in [San07].

Receiver-to-receiver clock synchronization exploits the existence of a physical broadcast medium. In detail, the broadcast medium provides the claim that any two nodes receive the same synchronization message in a single-hop transmission at nearly the same time. By exchanging the reception times of a reference broadcast among both receivers, the nodes can synchronize to each other. The main advantage of this approach is that it eliminates the communication delay that does not belong the receiver.[3] The only disadvantage may be the fact that this approach does not allow the receivers to synchronize to the sender of the reference message.

Global versus Local

This classification type belongs to the *scope* of synchronization which defines the nodes in the network that have to be synchronized. For instance, *global clock synchronization* protocols are aimed at the synchronization of all nodes within a connected network that may be a large multi-hop topology. This can be done either by a completely distributed algorithm, or step-wise by first executing a clustering algorithm and selecting cluster heads for inter-cluster and intra-cluster clock synchronization. This synchronization type is also known as *network-wide clock synchronization*.

Local clock synchronization protocols are aimed at the synchronization of a subset of nodes in the network, usually a cluster where each node is in the transmission range of each other node. For instance, the focus may be the synchronization of all nodes which are at most one hop away from each other to a single cluster-head. Such algorithms can be part of a network-wide synchronization protocol (*e.g.*, by creating a clustered network through a clustering algorithm).

A special case of the local synchronization is the *pairwise* synchronization approach which is targeted on high precision clock synchronization between pairs of neighbor nodes (either through sender-to-receiver or receiver-to-receiver based techniques).

Single-hop versus Multi-hop Network Topology

A *single-hop network* is a fully connected network where each node is capable of communicating with each other node in the network. Many synchronization algorithms assume such a network topology.

In WSNs, however, the nodes usually randomly distributed and thus span in an uncoordinated way over several hops resulting into a *multi-hop network* topology. This makes the synchronization more complex, especially for the startup synchronization. Furthermore, each hop degrades the synchronization precision to some extent. Note that if the network is created in a controlled and coordinated way, then intermediate nodes can be dedicated for acting as a gateway between several distinct network domains.

Static versus Dynamic Network Topology

In a *static network topology*, or also known as a *stationary network*, the nodes in the network do not change their local position. This simplifies the synchronization strategy.

In contrast, the structure of a *dynamic* or *mobile network topology* may change over time, that is the actual communication links may disappear and new links may be created dependent on the communication range of a sensor node. This requires robust and adaptive synchronization algorithms as presented in this thesis.

[3] Note that the propagation delay in a wireless medium is usually negligible compared to other delays.

Infrastructure-dependent versus Ad-hoc Network

Infrastructure-dependent networks are structured artificially (*e.g.*, hierarchically) in order to provide a fixed infrastructure for clock synchronization or other services. Examples are networks where compositional design evolves into a structured and robust network topology (*e.g.*, Triple Modular Redundancy (TMR) systems, dedicated gateway nodes for cluster-wise synchronization, *etc.*). A further example belongs to the cellular wireless networks used for mobile phones. However, in WSNs the large amount of nodes makes the manual positioning of the nodes nearly impossible and consequently belong almost exclusively to the type of ad-hoc networks.

Ad-hoc networks are networks which are created on-demand for a specific purpose without any available infrastructure. In contrast to infrastructure-dependent networks, the nodes in ad-hoc networks act as an end system and additionally as a network element [Dre07, p. 68].

Several synchronization algorithms for ad-hoc multi-hop networks assume a structured network (*e.g.*, tree-like structure or clustered network). Such structures can be established through the execution of special distributed algorithms. The different network structures are explained in more detail below. Whereas structured networks usually require a re-structuring in the case the network topology changes (*e.g.*, due to mobile nodes), synchronization algorithms based on unstructured networks are mostly immune to topology changes.

Unstructured. Unstructured networks do not make any assumption on the connectivity among the sensor nodes. However, since unconnected networks make no sense in the case of clock synchronization, the network is assumed to be at least connected. In the case fault tolerance is required, additional assumptions may be required. For instance, a k-connected network is a network where each node has at least k disjoint paths to any other node.

Clock synchronization algorithms designed for unstructured networks do not try to first discover the network and then establish some kind of structure. In contrast, such algorithms exchange information in the same way between any pair of groups or nodes. In other words, any node executes the same algorithm and there exist no dedicated nodes that serve as a gateway or master node. Such algorithms are typically *completely local*, that is, a node does not require the knowledge about the global state (*e.g.*, who is the cluster head). Furthermore, such algorithms are very effective in the presence of mobility, because they do not require to maintain and update the global state about the network structure in the case the topology changes.

Cluster structure. Many clock synchronization algorithms assume the existence of a clustered network. A cluster typically consists of a single cluster head (CH) and several cluster members. Each node in a cluster can communicate with each other node in the same cluster. A clustered network is a network where the nodes are partitioned into a set of clusters. In large ad-hoc multi-hop WSNs, such a partitioning is usually established by executing a *clustering algorithm*. For instance, the algorithms proposed in [MK06, MFLT05, NOKM08] divide the nodes into non-overlapping and approximately equal-sized clusters. An excellent survey on clustering algorithms is given in [AY07].

Tree structure. A tree structure is the most desired one in the fault-free case, because clock synchronization in such networks is very deterministic and achieves the best precision and accuracy. Furthermore, the root of the tree is assumed to be a more powerful device which is synchronized to an external time server (*e.g.*, UTC time). Algorithms that are based on a tree structure usually make use of a pairwise synchronization along the edges of the tree. Similar

to the clustering algorithms, there exist a lot of algorithms that construct a tree structure out of an unstructured ad-hoc multi-hop network (*e.g.*, [vGR03]).

Ring structure. A physical ring structure is a topology structure that hardly occurs in a WSN. However, there exist algorithms that create a virtual ring which is a communication path that passes each node at least once. A typical synchronization protocol that assumes such a structure is the all-node-based synchronization approach as stated by Li *et al*. in [LR06]. According to [AP98], clock synchronization algorithms that are based on a virtual ring scheme are classified to be symmetric or peer-to-peer based.

MAC-layer Based versus Standard Synchronization

The MAC-layer is part of the Data Link Layer of the Open System Interconnection (OSI) model and is responsible for providing a reliable and collision-free communication. The MAC-specific services may depend on the underlying physical layer.

Whereas *standard synchronization protocols* do not utilize the MAC-specific services, *MAC-layer based protocols* exploit some specific features provided by the MAC-layer in order to reduce energy consumption or communication jitter (*e.g.*, through the support of MAC-layer time-stamping).

CHAPTER 3

System Model of WSNs

OVERVIEW

The system model enables a common foundation for a formal analysis and comparison of the synchronization protocols. Therefore, we first define the used computational and communication model and then introduce an appropriate clock model.

3.1 Communication Model

With respect to WSNs an appropriate system model is the *bounded delay model* in a message-passing system that supports a broadcasting environment. For this, we first adapt the formal definition of a message-passing system from Attiya *et al*. [AW04]. In such a message-passing system, the processors exchange information by sending messages over communication channels. In wireless networks, these communication channels belong to the transmission range of a node through the wireless medium. The connections among the nodes can be represented by a directed communication graph where the nodes correspond to the component of the system and a directed link from node p_i to p_j is present if and only if p_j is within the transmission range of p_i. If the connection is bidirectional, then both nodes are in the transmission range of each other. In this case we say that both nodes are *neighbors*.

The pattern of connections between the nodes is the *topology* of the system. In this thesis, we will only treat *connected* topologies. We denote the *network* to be the collection of all connections in the system. An *algorithm* is based on the topology of a transition system and consists of the local program of all nodes in the system. The local program of processor p_i is modeled as a state machine with state set Q_i. A message-passing system consists of n processors $p_0, \ldots, p_{n-1}$ where each processor corresponds to a distinct node or component in the communication graph. We say that an algorithm is *anonymous*, if it does not depend on the existence of unique identifiers for the nodes. Consequently, all processors have the same state machine. Additionally, we say that an algorithm is *uniform*, if it does not depend on any system-specific parameters (*e.g.*, system size, topology, maximum number of connections, *etc.*). Since we do not assume a point-to-point communication, rather a broadcast communication, each node p_i consists of a single $inbuf_i$ component which holds all messages that have been delivered to p_i but have not yet been processed with an internal computation step.

In a message-passing system, there exist two kinds of events: Delivery and computation events. The *delivery* of message m from p_i to p_j, denoted by event $del(i, j, m)$, is modeled via a First-in First-out (FIFO) queue. Therefore, let $queue_{i,j}$ contain all messages sent by processor p_i to p_j, but not yet delivered by p_j. If p_i sends a new message m to p_j, then m is enqueued. In the case p_j delivers message m, that is it receives m from p_i, then m is dequeued from the front of the queue. Consequently, a bidirectional connection between p_i and p_j is modeled by two FIFO queues $queue_{i,j}$ and $queue_{j,i}$. The computation event $comp(i)$ represents an atomic computation step of processor p_i. We further assume an interleaving computation model. Therein, at each time instant only a single processor is allowed to execute an atomic computation step. Each computation consists of an *internal computation* and a possible single communication operation (*i.e.*, a transmission or delivery event). However, in wireless networks the harsh environment may induce message losses during the transmission due to unreliable communication channels and other environmental effects (*e.g.*, hidden terminal problem, interferences, *etc.*). In this case, we assume that the message loss results from an omission failure at the sending node. In other words, we assume that the communication network is reliable and all faults belong to the nodes.

The state of a message-passing system at a particular time is described via the *configuration* C and comprises the state of every processor and the content of every queue, that is

$$C = (q_0, \cdots, q_{n-1}, queue_{1,2}, queue_{1,3}, \cdots, queue_{i,j}, \cdots, queue_{n,n-1})$$

where $q_i \in Q_i$ is the state of p_i, $0 \leq i < n$, and $queue_{i,j}$, $i \neq j$, is the FIFO queue from p_i to p_j. Based on the definition of the system state, the future behavior can be determined solely from its current state.

The behavior of a system over time is expressed as a *sequence* of configurations C_i, $i \geq 0$, alternating with events Φ_i which can be either a computation or delivery event. In our case, such a sequence usually must satisfy two conditions: Safety and liveness condition. A *safety condition* holds in every finite prefix of E. In contrast, a *liveness condition* is a logical predicate that eventually holds a certain (or infinite) number of times. In the case the sequence satisfies all required safety conditions, then we say that this sequence is an *execution*, denoted as $E = C_0, \Phi_1, C_1, \Phi_2, \cdots$. If an execution E additionally satisfies all required liveness conditions, then we speak about an *admissible execution*.

In our bounded delay computation model, an execution is *admissible*, if each processor infinitely often performs computation events (processors do not fail) and every message sent is eventually delivered within a bounded transmission delay. The schedule of an admissible execution is an *admissible schedule*.

An *execution segment* $\alpha(E)$ is a finite (or infinite) sequence of the form

$$\alpha(E) = C_0, \Phi_1, C_1, \Phi_2, \cdots$$

which ends in a configuration in the case α is finite and satisfies the following two conditions:

- If $\Phi_k = del(i, j, m)$: In this case, m is the next message present in the FIFO queue $queue_{i,j}$, that is m is the oldest message in $queue_{i,j}$ which is then dequeued and afterwards added to the $inbuf_i$ component of p_i resulting in a change in configuration such that $C_{k-1}\Phi_k C_k$ with $C_{k-1} \neq C_k$.
- If $\Phi_k = comp(i)$: Based on the actual state of p_i and the set of messages stored in the $inbuf_i$ component, p_i performs an internal computation step, that is it changes its state from q_i to q_i' according to its transition function and adds at most one message m to each FIFO queue $queue_{i,j}$ where p_j is a neighbor of p_i. This transmission behavior realizes a *broadcast* communication.

In the case the starting configuration C_0 of α is an initial configuration, we simply say that α is an *execution* of the specified algorithm. The *schedule* (or *schedule segment*) of an execution (or execution segment) E is the corresponding sequence of events $S(E) = \Phi_1, \Phi_2, \cdots$. In the following, if we talk about executions we always consider timed executions. A *timed execution* is an execution that globally associates a nonnegative real number with each event (*i.e.*, the time at which that event occurs). We assume that the time starts at 0 and strictly increases for each individual node. Note that events that occur at the same time must not occur at the same processor.

3.1.1 Bounded Transmission Delay

The communication delay in WSNs is highly nondeterministic due to its contention-based media access strategy. Generally, the delay of a message between sending it from a sender to a receiver involves the following five parts.

1. *Send time* [t_{send}]: The time between the start of message construction at the *sender* and the time instant when the message was passed to the MAC layer. This time duration may be highly variable, because it includes delays introduced by the operating system (*e.g.*, scheduling delays, interrupt handling, *etc.*).
2. *Access time* [t_{acc}]: The time the *sender* waits for a clear channel assessment such that it thereafter is able to start a highly probable collision-free transmission over the communication channel. This time duration strongly depends on the implemented media access strategy and its parameter configuration. In wireless networks, the contention-mechanism is usually based upon a Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) mechanism with a backoff scheme, that is if the sender perceives an occupied media, then it postpones the transmission by a random backoff

until a clear channel assessment is identified. This time duration strongly varies due to the exponentially increasing backoff delay after each additional backoff until the backoff counter reaches a preconfigured maximum number. On this account, this part introduces the most significant delay jitter or the message transmission may be even aborted.
Note that we do not assume the existence of a Request To Send / Clear To Send (RTS/CTS) strategy which consequently implies the high possibility of hidden terminal problems. The *hidden terminal problem* occurs, if two nodes A and C are not in the transmission range of each other and both want to communicate at the same time to node B which is in the transmission range of both. As a result, the messages from A and C will collide at B.

3. *Serialization delay* [t_{sd}]: This delay corresponds to the time between the start of transmission or reception and the end of transmission or reception of a message at the sender or receiver, respectively, and strongly depends on the bit rate and the amount of bytes to be transmitted. We assume that this delay is the same at the sender and the receiver.
4. *Propagation delay* [t_{prop}]: This delay corresponds to the amount of time it takes for a bit to be transferred from the sender to the receiver over the wireless medium and depends on the distance between two nodes. However, since the propagation speed in a wireless medium is about $v = 200.000km/s$ and the worst case distance in WSNs is about $200 - 300m$, this part is usually neglectable.
5. *Receive time* [t_{recv}]: The time the MAC layer at the receiver requires to completely pass the received message to the next upper layer, after the last bit of the message was physically transferred. Similarly to the send time, the delay jitter of this communication part mainly results from the operating system.

Based on the informal elaboration of the communication delay, we can now formally define the *message delay* of a message to be the time duration between the computation event that sends the message and the computation event that processes the message.

We further assume that, as long as the message is eventually delivered, the message delay $d(m)$ of message m is always in the range $d(m) \in [d, d+\varepsilon]$ where d defines the constant part of the communication delay and ε the additional jitter of the communication delay. Furthermore, we assume a *uniform distribution* of the message delay for all communication channels. Note that if we assume the existence of a sophisticated time-stamping technique established below the MAC layer, then the send time t_{send}, the access time t_{acc}, and the receive time t_{recv} and their corresponding delay jitters are not included in the final message delay. In this case, we the message delay reduces to $d(m) = t_{sd}(m)$.

3.1.2 Complexity Measures

Usually, there exist two important complexity measures: Message and time complexity. However, the determination of both measures requires a notion of algorithm terminating. Therefore, we assume that the state Q_i for all processors p_i, $0 \leq i < n$, includes a subset of *terminated* states $T_i \subseteq Q_i$. We say that the algorithm (or system) has *terminated*, if all processors are in terminated state and all transmission queues $queue_{i,j}$ are empty.

Definition 21 (Message complexity)**.** *The message complexity of an algorithm is the maximum, over all possible admissible executions of the algorithm, of the total number of message broadcasts.*

Definition 22 (Time complexity)**.** *The time complexity of an algorithm is the maximum time until termination, over all possible admissible executions of the algorithm, in which the maximum message delay is at most one.*

3.2 Attacker Model

Similar to Ganeriwal *et al*. [GPČS08] we assume the existence of an omnipresent but computationally bounded adversary that controls a distinct number of communication channels in the sense that she inserts, modifies, eavesdrops, and blocks arbitrary messages.

We assume the existence of only internal attacks, *i.e.*, the faulty nodes are compromised and controlled by an adversary that knows all secret keys for an authorized communication with the other nodes within the network. We further exclude the presence of Sybil attacks. External attacks like jamming or pulse-delay attacks are also beyond the scope of this thesis.

3.3 Clock Model

This thesis primarily elaborates a distributed algorithm for Byzantine-fault tolerant self-stabilizing pulse synchronization in WSNs. On this account, we consider the already introduced concept of phase clocks as stated in Definition 7 and assume that each node p_i, $0 \leq i < n$, is equipped with a phase clock $\varphi_i(t)$ that acts according to the frequency of the underlying hardware clock HC_i. Consequently, we take on the notion of *pulse state* (Definition 8) and *pulse synchronization* (Definition 9). We further assume a bounded drift model as stated in Definition 5 with the maximum drift rate ρ among all participating nodes. Therefore, we can redefine the phase clock to a *drifting phase clock* as follows:

Definition 23 (Drifting phase clock)**.** *Let ρ be the maximum drift rate of a hardware clock among all nodes in the network and T be the nominal real-time duration of a cycle period. The phase clock $\varphi_j(t)$ of processor p_j is a phase variable that has the following properties:*

1. $\varphi_j : t \mapsto [0,1) = \varphi_j(t)$,
2. $\forall t_0, \Delta t$ *with* $0 < \Delta t \ll T \cdot (1-\rho)$ *and* $\Delta\varphi = \varphi_j(t_0+\Delta t) - \varphi_j(t_0) \neq 0$*:*
   - *if* $\Delta\varphi > 0$*:* $(1+\rho)^{-1} \leq T \cdot \frac{\Delta\varphi}{\Delta t} \leq (1-\rho)^{-1}$
   - *if* $\Delta\varphi < 0$*:* $(1+\rho)^{-1} \leq T \cdot \frac{1+\Delta\varphi}{\Delta t} \leq (1-\rho)^{-1}$
3. *and* $\varphi = 0$ *at the beginning of a cycle.*

Note that within this thesis, the above definition of the bounded drift is sometimes simply stated as $(1+\rho)^{-1} \leq T \cdot \frac{d\varphi_j(t)}{dt} \leq (1-\rho)^{-1}$ for sake of simplicity, but should be interpreted as defined in Definition 23.

Finally we assume that the the drift variation $|d\rho(t)/dt|$ is bounded by ϑ as stated in Definition 6 and that ϑ is so small for a given resynchronization interval that it can be assumed to be neglectable ($\vartheta \approx 0$), *i.e.*, we have a *constant rate model*.

3.4 Problem Statement

We assume that each node p_i is equipped with a unique identifier id_i. A node communicates via broadcasting with a bounded message delay in the range $[d, d+\varepsilon]$.We also assume a neglectable processing time of $\pi = 0$ time units. Each node is equipped with a local drifting phase clock $\varphi_i(t)$ with a nominal cycle time of T real-time time units and a bounded drift where ρ is the maximum drift rate among all nodes. This enables the definition of a non-faulty node and a non-faulty communication network as stated in Definition 24 and Definition 26. We additionally define both terms in the perfect case.

Definition 24 (Non-faulty node)**.** *A node is non-faulty at time t if for time $t' \geq t$, it behaves according to its algorithm, processes messages in no more than π time units, has a drifting*

phase clock with a bounded drift, and has a neglectable drift variation. A node that is not non-faulty is assumed to be Byzantine faulty as stated in Definition 19.

Definition 25 (Perfect node)**.** *A node is perfect if it is non-faulty and has a bounded drift of* $\rho = 0$.

Definition 26 (Non-faulty communication network)**.** *A communication network is non-faulty at time* t *if all broadcasted messages* m *at time* $t' \geq t$ *are received by the neighboring non-faulty nodes with a bounded message delay of* $d(m) \in [d, d+\varepsilon]$ *time units.*

Definition 27 (Perfect communication network)**.** *A communication network is perfect if it is non-faulty and for each message* m, $d(m) = 0$.

Definition 28 (Coherent system)**.** *A system is coherent at time* t *if the communication network is non-faulty and there are at least* $n - f$ *non-faulty nodes.*

Definition 29 (Fault-free system)**.** *A system is fault-free or non-faulty at time* t *if the communication network is non-faulty and all* n *nodes are non-faulty.*

Since we do not restrict our attention to simple single-hop networks and the fact that a multi-hop topology requires some kind of redundancy in order to tolerate Byzantine nodes, Definition 30 realizes such a measure of redundancy and is termed *k-connected network*.

Definition 30 (k-connected)**.** *A communication network is called k-connected, if there exist at least k disjoint paths between any two non-faulty nodes.*

Definition 31 formalizes the final synchronization problem. Note that the definition requires a fault-free system for converging, but then still maintains synchronicity in a coherent system. This may differ to other definitions in the literature. Informally, the goal of an algorithm solving this problem is to reach a synchronized pulse state in a fault-free system which is then maintained even in a coherent system. Note that we sometimes say that a system or network achieved *synchronicity* which means that it entered a synchronized pulse state. The second closure condition ensures that even faulty nodes do not broadcast more than one message during one complete cycle. This is feasible since we assumed that faulty nodes do not perform jamming or Sybil attacks. However, within the wireless communication we have to consider the case a single message originating from a faulty node can disrupt the message originating from another node. As already mentioned we assume that such a message loss is assumed to be an omission failure from the sender node. In other words, the communication network is always assumed to be non-faulty and the number of faulty nodes must be chosen large enough in order to have a neglectable probability of additional disregarded message losses. Clearly, this requires a high redundant communication network as they commonly exist in WSNs.

Definition 31 (Self-stabilizing pulse synchronization problem)**.** *A system solves the self-stabilizing pulse synchronization problem, if the following two conditions hold:*

- ***Convergence****: Starting from an arbitrary system state, if the set of non-faulty nodes of a fault-free system reaches a synchronized pulse state according to Definition 9 after a finite amount of time.*
- ***Closure****: If* $P(t_0)$ *is a synchronized pulse state of the system at real-time* t_0*, then* $\forall t \geq t_0$*,*
  1. *the pulse state* $P(t)$ *of a coherent system at real-time* t *is a synchronized pulse state, and*
  2. *each (non-faulty or faulty) node broadcasts at most one message in any interval* $[t, t + T \cdot (1+\rho)]$.

CHAPTER 4

Related Work

OVERVIEW

Time synchronization in wireless networks was given a lot of attention during the last decades. As a result of intensive research, manifold approaches and protocols have been proposed. This chapter covers only a few of them which are relevant with respect to this thesis. Since we discuss different aspects of time synchronization mechanisms, we partition the related work into the area of the different synchronization approaches. Therefore, Section 4.1 and Section 4.2 first present selected resilient clock synchronization protocols which are asymmetric and symmetric, respectively. Section 4.3 then discusses recent publications in the area of self-stabilizing pulse synchronization protocols that are resilient against Byzantine faults. Finally, Section 4.4 gives a short overview about related work for establishing digital clock synchronization.

4.1 Resilient Clock Synchronization

This part captures selected works during the last years in the area of network-wide clock synchronization protocols in wireless networks that are resilient to compromised nodes. Note that we do not concentrate on security aspects of the protocols (*e.g.*, eavesdropping, *etc.*). An excellent overview of several protocols can be found in [SBK05]. Other surveys on synchronization protocols in wireless networks are proposed in [SY04, RBM05, Fai07, RLK+09]. Therefore, many popular and well-established protocols are not discussed due to their absence of fault tolerance. For instance, in [MRS05], Manzo *et al*. present several attacks on the Reference Broadcast Synchronization Protocol [EGE02], the Flooding Time-Synchronization Protocol (FTSP) [MKSL04], and the Timing-Sync Protocol for Sensor Networks (TPSN) [GKS03a, GKS03b]. According to Sun *et al*. [Sun05], other interesting protocols that are not resilient against internal attacks are the protocol presented by Mock *et al*. [MFNT00b, MFNT00a] which is based on the Master/Slave synchronization scheme of the IEEE 802.11 standard, Timestamp synchronization (TSS) [Röm01], Lightweight Tree-based Synchronization (LTS) [vGR03], Tiny-Sync and Mini-Sync (TS/MS) [SV03], Hu and Servetto's protocol [HS03], Tsync [DH04], Adaptive Clock Synchronization [PSJ04], Asynchronous Diffusion (AD) [LR06], and [Liu08].

To give an overview of the huge amount of other published synchronization protocols that do not provide fault-tolerant clock synchronization in wireless networks, we list a few of them: Directed Diffusion [IGE00] does not consider compromised nodes; SPINS [PST+02] is a suite of security protocols comprising SNEP and μTESLA that provide a secure broadcast authentication protocol for local authenticated broadcast but do not handle resilience to compromised nodes; Delay Measurement Time Synchronization for Wireless Sensor Networks (DMTS) [Pin03] is based on a Master/Slave synchronization; the Timing Synchronization Function (TSF) [iee03] used in IEEE 802.11 Independent Basic Service Set (IBSS) ad-hoc networks specified in the IEEE 802.11 standards is a completely distributed synchronization algorithm but is hardly scalable [HL02] and suffers from the fastest node asynchronization problem as stated in [LZ03] which makes it prone to compromised nodes; Adaptive Timing Synchronization Procedure (ATSP) [HL02] is an improvement of the TSF algorithm which is more scalable and reduces the problem of the fastest node asynchronization, however, fault tolerance is not considered; Tiered ATSP (TATSP) [LZ03] is an enhancement of the ATSP protocol where a set of fastest nodes is classified through estimation techniques, however, synchronization attacks are again not assumed; Automatic Self-time-correcting Procedure (ASP) [SCS04] considers time synchronization in IEEE 802.11 based multi-hop networks but again does not provide fault tolerance to internal attacks; Rentel *et al.*'s protocol [RK04] is a IEEE 802.11 compatible synchronization algorithm that is not aimed at preventing internal attacks; Self-adjusting Timing Synchronization Function (SATSF) [Zho05] is compatible with IEEE 802.11 TSF, scalable, accurate, bounded, and adaptive to mobility, but is not resilient to any kind of synchronization attacks; Probabilistic Clock Synchronization [JPS03] does again not handle internal attacks; Tulone's Clock Reading Protocol (DCR/PCR) [Tul04] is not resilient to faulty nodes; Meier *et al.*'s protocol [MBT04] handles internal clock synchronization and improves the protocol from [Röm01] but is not fault-tolerant; Back-Path Interval Synchronization Algorithm (BP-ISA) [BMT04] is an improved version of the algorithm IM from [MO83] but again does not consider fault tolerance; Multi-hop Timing Synchronization Function (MTSF) [SV04] is a network-wide time synchronization algorithm which is fully distributed and self-stabilizing similar to our approach where the nodes synchronize to the fastest one, however, it does not handle internal attacks; Control Time Protocol (CTP) [GK04] from Graham *et al*. is not designed for the presence of malicious nodes; Time-Diffusion Synchronization Protocol (TDP) [SA05]

consists a set of algorithms for self-determining master nodes, creating a temporary tree-like structure via elected diffused leader nodes, and consequently for establishing a network-wide time synchronization where the Allan variance is used for determining the deviations between two clocks, but again does not consider the possible presence of compromised nodes; Routing Integrated Time Synchronization (RITS) [SKLD06] (reactive time synchronization) and Rate-Adaptive Time Synchronization (RATS) [GGH+05, GGS+05, KDL+06] (proactive time synchronization) are aimed at network-wide long-term time synchronization for a given application-specific precision bound that can adapt to changing environment conditions and clock-drift, but is prone to several kinds of attacks; The distributed asynchronous time averaging algorithm by Giridhar and Kumar [GK06] and Solis *et al*. [SBK06] exploit the existence of loops in the network and therefore belongs to a infrastructure-dependent synchronization approach which additionally do not consider internal attacks; Dynamic Continuous Clock Synchronization (DCCS) [MBRS08] is a Master/Slave synchronization approach based on 802.15.4 standard that consequently implies a single-point of failure; Adaptive Internal Clock Synchronization [JFF08] is an internal clock synchronization algorithm that handles only crash failures and does not consider the existence of Byzantine nodes; The hybrid time synchronization protocol [HDQK09] (combination of one-way dissemination and two-way exchange approach) assumes a tree-structured network which is improper in the presence of compromised nodes.

In contrast to the previous presented approaches, the following protocols are of particular interest within this thesis, since they provide resilience to at least internal attacks and additionally are based on a completely distributed synchronization approach. Whereas some of them are based on the detection of malicious nodes, others assume the persistent presence of up to f faulty nodes and exclude them from the synchronization algorithm. For instance, Generalized Extreme Studentized Deviate Many-Outlier Procedure (GESD) [SZC05] is a modified version of the Extreme Studentized Deviate (ESD) test which aims at detecting multiple outliers of malicious time offset information resulting from compromised nodes or delay attacks in the context of RBS and is based on the assumption that the time offsets among the nodes follow a similar distribution. In addition to the GESD approach, Song *et al*. also propose a threshold-based detection technique which requires less reference nodes. However, Song *et al*. only consider the synchronization of neighboring nodes and is therefore only of little interest since we require a network-wide synchronization in multi-hop networks.

4.1.1 Synchronizer Ring

This protocol is a fault-tolerant cluster-wise synchronization algorithm where the nodes in the cluster can directly communicate with each other. Message authentication is established via lightweight symmetric cryptography [ZSJ03] that exploits the broadcast medium instead of costly and complex digital signatures. The main concept behind this protocol developed by Sun *et al*. [Sun05] is that in contrast to traditional fault-tolerant clock synchronization algorithms which usually require a periodic exchange of messages among all nodes, the Synchronizer Ring protocol elects one node as the cluster-head in each round. This cluster-head then serves as a synchronizer and all other nodes adjust their clocks with respect to this node. Consequently, only one authenticated synchronization message that does not contain any time information is broadcast during each round. This dramatically reduces the probability of message collisions and therefore the uncertainty of message delay. Additionally, fault tolerance is provided through a round-robin based rotation of the synchronizers. In detail, a synchronization message is considered to be faulty if it exceeds an *a priori* defined maximum clock difference between any non-faulty node and a non-faulty synchronizer. This

approach is similar to the previously introduced threshold-based detection technique. The algorithm further guarantees an upper bound of the clock difference.

The Synchronizer Ring cannot be used for inter-cluster synchronization, only for intra-cluster synchronization. However, in the case the network is partitioned into a set of clusters (*e.g.*, through a clustering-algorithm), the proposed protocol can be used as a building block of a global clock synchronization algorithm.

However, the proposed protocol assumes that the nodes are initially synchronized and that the nodes within a cluster agree on the round-robin order in which they serve as the synchronizer. In the case, the clocks are initially synchronized, the order agreement can be achieved through fault-tolerant digital clock synchronization. The authors state that the initial synchronization can be either established by a bootstrapping phase where the nodes adjust their clocks to trusted reference nodes or by the use of the initial synchronization algorithm proposed by Lundelius and Lynch [LL84a]. This initial clock synchronization algorithm is based on the simple fault-tolerant averaging technique. However, the latter approach is only applicable in fully-connected networks and does not consider a network-wide initial synchronization. In contrast, the approach proposed in this thesis guarantees a network-wide self-stabilizing pulse synchronization even in the presence of multi-hop networks.

4.1.2 Asynchronous Diffusion

In [LR06], Li *et al*. discuss four methods for achieving *global clock synchronization* in sensor networks:

1. The *all-node-based* method,
2. the *cluster-based* method,
3. the *fully localized diffusion-based* method, and
4. the *fault-tolerant diffusion-based* method.

Global clock synchronization means that all nodes in the network have approximately the same clock value, irrespective of their relative distance. The first two methods do not consider fault tolerance neither are useful for WSNs due to their improper assumptions (see [LR06] for details). On this account, we concentrate only on the diffusion-based methods.

Generally, time-diffusion synchronization protocols [SA05] achieve global clock synchronization by spreading the local synchronization information to the entire network. Time synchronization is usually initiated by a master node. The neighboring nodes then adjust to the master node and additionally diffuse this clock adjustment to the other neighbors and so on. Contrary, the time-diffusion methods presented by Li *et al*. do not assume any specific master or diffusion nodes. Instead, the presented methods are fully localized (*i.e.*, they synchronize without a global synchronization initiator) and fault-tolerant. This means that every node can be a master node or diffusion node. In other words, the presented approach is a completely distributed algorithm and therefore of great importance within this thesis.

The authors propose two different diffusion-based algorithms: The Rate-Based Synchronous Diffusion Algorithm and the Rate-Based Asynchronous Diffusion Algorithm. The *synchronous diffusion* algorithm requires a set order for their local operations and is based on weighted averaging which ensures the flow conservation among the clock values. The *asynchronous diffusion* algorithm does not require any constraints on the order of local operations, that is, a node can synchronize with its neighbor at any time in any order and is therefore of particular interest.

The main idea of the asynchronous diffusion algorithm is presented in Algorithm 3. In detail, a node periodically gathers the clock values from its neighbors (clock reading) and then averages them. The averaged value is sent back to the neighbors (and itself), which update their clock to this value. The authors proved that, as long as the network is con-

nected, after some rounds each node eventually converges to the global average time. The only requirement is that a node executes the algorithm with nonzero probability and the sequence of operations must be atomic, that is, in the case a node is involved in several average operations, then these operations must be sequenced.

Algorithm 3: Asynchronous Averaging Algorithm: code for p_i, $0 \leq i < n$

```
1 for each node p_i with uniform probability do
2     Read value from p_i and its neighbors
3     Average the readings
4     Send the new value back to the neighbors (write values to p_i and its neighbors)
```

The authors additionally propose a modification of the above algorithm which is resilient to some fraction of Byzantine nodes. For this, they assume the existence of tamper-proof nodes together with normal nodes. A tamper-proof node can always be trusted and can only experience crash failures. Such nodes also serves as cluster heads for the normal nodes. However, since such nodes may be too costly for WSNs, this approach is improper and therefore not discussed in more detail. Furthermore, this assumption contradicts the idea of a fully localized and completely distributed synchronization algorithm where each node executes the same algorithm.

4.1.3 Chen *et al.*'s protocol

In [CL07], Chen *et al*. propose two synchronization protocols for a network-wide, serverless, and proactive time synchronization in IEEE 802.11 IBSS ad-hoc networks: Single-hop secure time synchronization procedure (SSTSP) and Multi-hop secure time synchronization procedure (MSTSP). Both consider an attacker model consisting of both external and internal attacks. The main differences of this contribution to many other published protocols are that the protocols are scalable and totally distributed, *i.e.*, the synchronization does neither require a single source node (*e.g.*, a synchronization leader) nor a synchronization hierarchy (*e.g.*, tree-structured network). Furthermore, their approach is fully localized since it is not based on information flooding.

SSTSP

SSTSP is aimed at time synchronization in single-hop ad hoc networks and makes use of a periodic exchange of time information through beacons similar to the TSF approach in IEEE 802.11 IBSS networks. The period time is defined by the Beacon Period (BP).

Synchronization is achieved through two phases: Bootstrapping phase and synchronization phase. The *bootstrapping phase* makes use of the contention mechanisms of TSF. In detail, in the beginning all nodes content to emit synchronization beacons. The winner is elected as the reference node and the nodes switch to the synchronization phase. During the synchronization phase, only the single reference node broadcasts a synchronization beacon in the beginning of every BP. The other nodes then synchronize to the reference node. If a node does not receive synchronization beacons during the last few BPs (*e.g.*, because the reference node may left the network), the nodes will initiate a new bootstrapping phase.

Similar to the Synchronizer Ring Protocol, security and integrity of the synchronization messages are achieved through symmetric cryptography by the use of μTESLA [PST$^+$02] instead of costly digital signatures. A *clock drift check* is used to detect time attacks (*e.g.*, replay attacks, pulse-delay attacks), *i.e.*, if the message delay of a beacon exceeds the threshold of the clock drift check, then the beacon is assumed to be faulty. For this, in order to

measure the message delay, each node acknowledges the reception of a packet with an ACK packet and therefore can be classified as a two-way exchange protocol.

The authors make the assumption of a bounded drift model and that each pair of nodes initially share a pairwise secret key for the local authenticated broadcast mechanism during the bootstrapping phase. Furthermore, they assume the existence of MAC-layer time-stamping.

MSTSP

Global time synchronization via MSTSP is achieved through the following two phases: First, a multi-hop network is automatically partitioned into several overlapping single-hop clusters by simply executing SSTSP which additionally enables the intra-cluster synchronization. The second phase concerns the inter-cluster synchronization, *i.e.*, all cluster reference nodes are synchronized by exchanging synchronization beacons via bridge nodes.

For this, SSTSP was extended in three ways: First, the reference node waits a random time before broadcasting the synchronization beacon in order to reduce the probability of message collisions with other reference nodes. Secondly, bridge nodes are introduced and correspond to the nodes in overlapping cluster areas which then periodically exchange the time information among the cluster reference nodes in an asynchronous way. Thirdly, a reference node collects the time information of the other reference nodes via bridge nodes and synchronizes to the fastest one, *i.e.*, the fastest cluster.

Robustness against compromised nodes is achieved by observing the time differences between the cluster reference nodes. This is done at the bridge nodes which notify the neighbors about this misbehavior, if the time difference exceeds a certain threshold. In the case multiple bridge nodes detect such a misbehavior, the synchronization process is re-initiated.

4.1.4 Secure Group Synchronization

The Secure Group Synchronization (SGS) protocol is part of the seminal work about secure time synchronization in sensor networks [GPČS08] which is a refinement of [GČHS05]. Therein, Ganeriwal *et al*. propose several protocols: Secure Pairwise Synchronization (SPS), Enhanced SPS (E-SPS), Lightweight SGS (L-SGS), SGS, Secure Simple Multi-hop Synchronization (SSM), Secure Transitive Multi-hop (STM) synchronization, and Secure Direct Multi-hop (SDM) synchronization.

However, only the SGS protocol is of particular interest with respect to this thesis, since the pairwise synchronization protocols (*i.e.*, SPS and E-SPS) do not provide countermeasures against internal attacks. Furthermore, the pairwise synchronization protocols are inefficient with respect to the message complexity in the case of large network topologies. Note that the SPS protocol is a secure version of the TPSN protocol and the E-SPS protocol is similar to the later discussed TinySeRSync protocol. Similarly, a network-wide time synchronization via SSM or STM are again of less interest, because both protocols are based on the pairwise synchronization approach without considering compromised nodes.

SGS is an enhanced version of the L-SGS protocol and additionally tolerates internal attacks resulting from compromised nodes. SGS is aimed at achieving instantaneous group consensus instead of a periodic resynchronization assuming that all nodes in the group can directly communicate with each other. Therefore, any node is able to initiate SGS. The system assumptions for SGS are that a compromised node is unable to send valid messages on behalf of some other node and that every received message can be authenticated. To overcome these assumption, the authors suggest the use of symmetric or asymmetric cryptography that is usually based on costly digital signatures. However, according to the authors,

symmetric cryptography enables only a small group cardinality of up to at most 15 nodes, because a node has to attach a message authentication code for each receiving node and the computational complexity is bounded due to time requirements. On this account, the SGS protocol is hardly scalable. Resilience against internal attacks is achieved through the Secure time synchronization OM (SOM) algorithm, which is a Byzantine agreement protocol based on the OM algorithm [LSP82] and the COM algorithm [LMS85]. SGS executes in six steps where N denotes the group cardinality:

1. Some node initiates SGS by broadcasting a *challenge packet* that contains its id and a challenge nonce. If the other nodes receive a challenge packet for the very first time, then they react in a random order by sending also a challenge packet. If a node receives more than $N_{min} \leq N-1$ such packets from different nodes, then it proceeds with step 2. Additionally, each node p_A stores the sending time $t_{A,1}$ and the receipt time $t_{B,1}$ of the challenge packet received from p_B.
2. Each node p_B broadcasts a single *response packet* that consists the *receipt time*, the *nonce*, and the *node id* from p_A of each received challenge packet and the node's *sending time* $t_{B,2}$ in combination with a message authentication code for each receiving node.
3. Based on the information of the received response packets and the receipt time $t_{A,2}$ of the response packet, each node p_A then calculates the average message delay d_{AB} to any neighbor node p_B by

$$d_{AB} = ((t_{B,1} - t_{A,1}) + (t_{A,2} - t_{B,2}))/2. \tag{4.1}$$

   A threshold verification on d_{AB} ($d_{AB} \leq d^*$) excludes possible pulse-delay attacks from an external attacker. The pairwise time offsets δ_{AB} with the other resulting nodes p_B are added to the *offset set* $O_A = O_A \cup \delta_{AB}$ where

$$\delta_{AB} = ((t_{B,1} - t_{A,1}) - (t_{A,2} - t_{B,2}))/2. \tag{4.2}$$

4. Each node p_A broadcasts the gathered offset set O_A in addition with $N-1$ message authentication codes.
5. Based on all received offset sets, each node p_A runs the $SOM(\lfloor (N-1)/3 \rfloor)$ algorithm in order to calculate the estimated clock value C_{AB} of each neighbor node p_B.
6. The median of the resulting set of clock estimates C_{AB} and the local clock C_A is then taken as the group clock C_g^A.

The SOM Algorithm

The SOM algorithm is a recursive algorithm which is executed in several rounds of computation in order to estimate the clock value of a neighboring node, based on the information gathered from all nodes in the group. SOM requires the number of faulty nodes as an argument. Since the algorithm accepts at most one third of faulty nodes f and only the group cardinality N is known by the nodes, the authors suggest the use of $f = \lfloor (N-1)/3 \rfloor$. The only disadvantage of this assumption is that the algorithm may execute in more rounds than necessary. In other words, the algorithm guarantees that each node agrees on the same (possible faulty) value of a Byzantine node. Consequently, all nodes also agree on the group clock value which is the median among all clock estimates. Algorithm 4 gives a pseudo code presentation of the SOM algorithm. Therein, C_{ij}^{kr} denotes the local clock estimate of node p_j at p_i using node p_k after r executed rounds of SOM. For a detailed explanation, the reader is referred to [GPČS08].

Algorithm 4: $SOM(f)$ to estimate C_{ij}: code for $p_i \neq p_j$, $0 \leq i, j < N$

```
return median{SOM(j,k,f) | 0 ≤ k < N, k ≠ j}
procedure SOM(j,k,r) to estimate C_ij^kr :
    if r = 1 then
        return C_i + δ_ik + δ_kj
    else
        return median{SOM(k,t,r−1) | 0 ≤ t < N, t ≠ k ≠ j} + δ_kj
```

4.1.5 TinySeRSync

TinySeRSync by Sun *et al*. [SNW06b] is an improved version from the protocol stated in [SNW06a] and is intended for a secure and resilient time synchronization. Sun *et al*. propose two protocols. The first one is aimed at secure single-hop pairwise time synchronization using hardware-assisted, authenticated MAC-layer time-stamping which can handle high data rates and is an extension to the SPS approach from [GČHS05]. The second protocol is a secure and resilient global time synchronization protocol.

Secure Single-hop Pairwise Time Synchronization

For the pairwise time synchronization, the authors assume that every pair of nodes share a secret pairwise key (*e.g.*, through TinyKeyMan[1]). Security and integrity aspects of the pairwise synchronization approach are enabled through authentication of the synchronization messages by adding a prediction-based Message Integrity Code (MIC) that is based on the shared secret key. The timeliness aspect is enabled by adding a MAC layer timestamp.

The pairwise synchronization between some node A and B is based on a two-way exchange approach in order to determine the clock difference and the estimated transmission delay. In detail, if node A initiates the synchronization by sending message M1 containing its timestamp $t_{A,1}$, then node B receives M1 at $t_{B,1}$ and responds with message M2 at time $t_{B,2}$. Since M2 contains all three timestamps, after node A receives M2 at $t_{A,2}$ it is able to calculate the clock difference with

$$\Delta_{AB} = (t_{B,1} - t_{A,1}) - (t_{A,2} - t_{B,2}) \tag{4.3}$$

and the estimated message delay

$$d_{AB} = (t_{B,1} - t_{A,1} + t_{A,2} - t_{B,2})/2. \tag{4.4}$$

Thus, node A is able to prevent external attacks like pulse-delay attacks or wormhole attacks by verifying the transmission delay according to the aforementioned GESD or threshold based detection approach [SZC05]. Internal attacks like message modifications or forged messages will be detected due to the authentication approach and consequently protects the source, content, and timeliness of the synchronization messages. Replay attacks are prevented by exploiting the uniqueness of the sender's timestamp. To obtain a precise time synchronization, this protocol is executed periodically

Secure and Resilient Global Time Synchronization

For the global time synchronization, the authors assume that there exists a single trusted source node S that is accurately synchronized to an external time source (*e.g.*, UTC) and the other nodes have to synchronize to S. To deal with ad-hoc wireless networks, Sun *et al*.

[1] http://discovery.csc.ncsu.edu/software/TinyKeyMan/

propose a protocol that consists of two phases which are executed individually and independently in a periodic manner. During the first phase, a secure pairwise synchronization is established. After the first phase finished, the second phase is initiated by the source node and enables a secure and resilient global time synchronization which is resilient against external attacks and compromised nodes. Security is achieved through local authenticated broadcasts via an adaption of μTESLA [PST+02] instead of costly digital signatures.

Let C_i be the local clock of node p_i. For global clock synchronization, each p_i obtains the direct clock difference to each neighbor node p_j, denoted by $\Delta_{i,j} = C_j - C_i$. This is done by the use of the secure pairwise time synchronization protocol. Each node p_i further maintains a source clock difference $\delta_{i,S}$ with respect to the source node S. After the source node initiates a synchronization by broadcasting a synchronization message, the direct neighbors can obtain the source clock difference directly, since all nodes know the identity of S. All other nodes p_i have to estimate $\Delta_{i,S}$ indirectly. For this, the direct neighbors broadcast their source clock difference and so on. Resilience to compromised nodes is achieved through redundancy, *i.e.*, in order to tolerate up to f compromised neighbor nodes, a node p_i that is not a direct neighbor of S has to receive the source clock difference from at least $2f+1$ different neighbors p_j. Node p_i then takes the median of the corresponding $2f+1$ candidate source clock differences $\Delta^j_{i,S} = \Delta_{j,S} + \Delta_{i,j}$. Let $\Delta_{i,S}$ denote the median. The estimated global clock at p_i then equals $\hat{C}^i_S = C_i + \Delta_{i,S}$. Afterwards, p_i rebroadcasts the resulting own source clock difference in order to flood the time information throughout whole network. High accuracy is maintained through a periodic synchronization initiation of the source node.

4.1.6 Sanchez's Protocol

In [San07], Sanchez presents a network-wide, proactive, energy-efficient time synchronization protocol for clustered multi-hop networks which is based on the combination of SPS with sample exchange (SPS-SE), RATS, and μTESLA. In detail, SPS-SE is used for the secure pairwise time synchronization, RATS maintains the accuracy throughout longer resynchronization periods, and, similar to the other protocols, μTESLA enables local authenticated broadcasts. Energy-efficiency is achieved through low duty-cycling and a low message complexity.

The presented protocol is resilient against external and internal attacks and assumes the existence of a trusted base station which is synchronized to UTC time and under human control. Security through μTESLA assumes that each pair of nodes p_A, p_B share a pairwise secret key K_{AB}. The protocol further requires a clustered network structure consisting of cluster heads (CH) which can be established through a clustering algorithm. However, it is necessary that communication network of the cluster heads is connected such that they can directly communicate. For achieving high precision, the authors also assume that timestamping is done below the MAC-layer. Since the protocol considers long-term synchronization, it is assumed that the drift rate of the clocks remain constant during a given period of time, termed as *epoch*. The relative clock model between two nodes p_A and p_B is assumed to be P-degree polynomial with $P = 1$ as used in the RATS protocol [GGS+05]. In detail, the estimated time $\hat{t}_B(t_A)$ of p_B at p_A equals

$$\hat{t}_B(t_A) = \sum_{p=0}^{P} (\beta_p \cdot t_A^p) + \upsilon \tag{4.5}$$

where υ corresponds to the error due to measurement and clock variations. Thus, given a set $\{(t_{A,i}, t_{B,i}) \mid 0 \leq i < W\}$ of W consecutive time observations where W is termed as the

window size, rate adaptive synchronization is performed through the simple calculation of β_0 and β_1 such that the residual sum of squares (RSS) is minimized:

$$RSS = \min_{\forall \beta_0, \beta_1} \sum_{i=1}^{W} (t_{B,i} - [\sum_{p=0}^{1} (\beta_p \cdot t_{A,i}^p)])^2 \tag{4.6}$$

The authors state that $W = 8$ with a sample period of $S = 60$ seconds, where S is the time duration between two consecutive observations, is optimal for indoor environments.

Based on the aforementioned assumptions, Sanchez's protocol establishes a network-wide synchronization through the periodic execution of the following two phases: *Secure CH pairwise re-synchronization* and *secure cluster re-synchronization*. Both use SPS-SE for the pairwise synchronization.

SPS-SE is very similar to SPS [GPČS08]. However, whereas SPS implements a two-message exchange protocol, SPS-SE extends SPS such that up to τ time observations are securely exchanged. In detail, the following sequence illustrates the message exchange in SPS-SE:

1. $p_A(t_{A,1}) \rightarrow (t_{B,1})p_B$: id_A, id_B, $t_{A,1}$, τ
2. $p_B(t_{B,2}) \rightarrow (t_{A,2})p_A$: id_B, id_A, $t_{B,1}$, $t_{B,2}$, $MAC_{K_{AB}}(\cdots)$
3. $p_A(t_{A,3}) \rightarrow (t_{B,3})p_B$: id_A, id_B, $t_{A,2}$, $t_{A,3}$, $MAC_{K_{AB}}(\cdots)$
4. $p_B(t_{B,4}) \rightarrow (t_{A,4})p_A$: id_B, id_A, $t_{B,3}$, $t_{B,3}$, $MAC_{K_{AB}}(\cdots)$
5. $\cdots$ ($\tau - 3$ additional steps)

Note that $MAC_{K_{AB}}(\cdots)$ denotes the appended message authentication code based on the shared secret key K_{AB}. According to Equation 4.2 and Equation 4.1 which are also used in the original SPS approach, both nodes then calculate the time difference and the message delay such that they are able to synchronize together.

Secure CH Pairwise Re-synchronization

As already mentioned, pairwise cluster head re-synchronization requires a clustered network. However, due to possible compromised CHs and changing environments, the authors suggest a periodic re-clustering with period time R.

For the pairwise synchronization between two cluster heads p_A and p_B, the authors propose the segmentation of the time into a number of variable pairwise time periods $S_{A,B}^k$, $1 \leq k \leq r_k$, with $\sum_{k=1}^{r_k} S_{A,B}^k = R$ where the first period $S_{A,B}^1$ starts right after p_A and p_B have discovered each other. This provides capabilities to establish listen/sleep schedules.

Initial pairwise synchronization is performed during the first period by the exchange of several time samples according to the default window size $W_{A,B}$ via the SPS-SE protocol. Pairwise re-synchronization is initiated at the beginning of the subsequent periods $S_{A,B}^k$, $2 \leq k \leq r_k$, by the use of SPS-SE. However, only one new time sample is exchanged in these periods.

A *time of guard* T_{guard} in dependence of the measured maximum clock drift is introduced at the beginning of each period $S_{A,B}^k$. During this time, a node is only allowed to receive messages. This ensures that a node does not miss a message due to a late wake up resulting from the drifting clocks in the case duty-cycling is enabled. The first node leaving T_{guard} initiates a re-synchronization.

The optimal pairwise window size $W_{A,B}^k$ and the corresponding optimal period time $S_{A,B}^k$ is calculated for each period $S_{A,B}^k$ at both nodes via RATS [GGS+05]. The time estimation of a neighbor node is calculated by the use of Equation 4.5 and Equation 4.6 based on the exchanged time samples.

Secure Cluster Re-synchronization

For the cluster synchronization with cluster head p_{CH}, the authors propose the segmentation of the time into a number of variable cluster time periods S_{Cl}^{j}, $1 \leq j \leq r_j$, with $\sum_{j=1}^{r_j} S_{CL}^{j} = R$ where the first period S_{CL}^{1} starts right after the cluster is formed. This provides capabilities to establish listen/sleep schedules.

In order to provide secure broadcasts via μTESLA, p_{CH} first generates a sequence of q keys K_i by hashing a random value K_{CL}, *i.e.*, $K_i = h^i(K_{CL})$ with $0 \leq i \leq q$. Initial cluster synchronization between p_{CH} and any cluster member p_u is performed during the first period by the exchange of several time samples according to the default window size W_{CL} via the SPS-SE protocol. Additionally, p_{CH} broadcasts the last key $h^q(K_{CL})$ in one of the SPS-SE messages. Cluster re-synchronization is performed through the secure pairwise re-synchronization between p_{CH} and any cluster member p_u at the beginning of the subsequent periods S_{CL}^{j}, $2 \leq j \leq r_j$, by the use of SPS-SE. However, only one new time sample is exchanged in these periods. Additionally, a cluster member p_u calculates and sends $\hat{t}_{CL}(t_u)$ to p_{CH} in each period.

A *time of guard* T_{guard} is again introduced at the beginning of each period S_{CL}^{j} in order to allocate different time slots for inter-cluster and intra-cluster synchronization. After T_{guard}, p_{CH} is the only node within the cluster which is allowed to communicate. This ensures a contention-free medium access for p_{CH}. A cluster member p_u is only allowed to communicate after it received an initial message from p_{CH}.

The optimal cluster window size $W_{CL}^{j} = \max\{W_{CH,u}^{j} \mid p_u \text{ is cluster member}\} = W_{CH,x}^{j}$ for some cluster member p_x and the corresponding optimal period time S_{CL}^{j} is calculated for each period S_{CL}^{j} at the cluster head p_{CH} via RATS [GGS+05]. p_{CH} then broadcasts the new values in a secured message. A cluster member p_u estimates $\hat{t}_{CL}(t_u)$ by the use of Equation 4.5 and Equation 4.6 based on the exchanged time samples.

Security Aspects

External attacks are prevented through authenticated synchronization messages. Furthermore, wormhole or pulse-delay attacks are detected since the corresponding measured message delays are usually beyond the expected end-to-end delay.

In the case multiple communication routes among the cluster heads exist, then a compromised cluster head during the secure pairwise CH re-synchronization can usually be detected which is then added to a blacklist of untrusted nodes in and a re-clustering is initiated. Similarly, a compromised cluster member during secure cluster re-synchronization can be detected by the cluster head and is again added to a blacklist. A compromised cluster head during secure cluster re-synchronization can be prevented, if the cluster head commits the identity of the selected cluster member p_x for calculating the maximum window size W_{CL}^{j}. In the case both the cluster head and p_x are compromised, lower and upper bounds for S_{CL}^{j} are calculated to reduce the impact of an increasing duty-cycle for the complete cluster.

4.1.7 Fast Fault-Tolerant Time Synchronization

Fast Fault-Tolerant Time Synchronization (FFTS) was proposed by Lee *et al*. [LJP08]. FFTS is a very fast, network-wide, proactive, fault-tolerant, and energy-efficient time synchronization protocol for unstructured multi-hop networks. Additionally, FFTS is completely distributed and supports duty-cycling. The authors only address internal clock synchronization.

They make the assumption of a bounded drift model and that at most f out of all nodes are faulty such that they either suffer from timer faults (Definition 18) or crash faults (Def-

inition 16). The protocol requires the assumption of a $(f+1)$-connected network [RSB90] comprising N sensor nodes, that is, the network has at least $f+1$ disjoint paths between any two non-faulty nodes. The protocol uses MAC-layer time-stamping and drift rate adjustment via linear regression. The protocol uses two delay constants: $MAXBACKOFF$ and $PERIOD$. $MAXBACKOFF$ is used to solve the medium access contention problem. In detail, when we say a node broadcasts with a random backoff delay, then the node delays the transmission by a random time between 0 and $MAXBACKOFF$. The $PERIOD$ parameter corresponds to the resynchronization interval. Based on these assumptions, FFTS proceeds in three phases:

Phase 1: INIT

This phase is executed once at the beginning of the node's lifetime. It assumes that, initially, each node resets its clock to 0. Afterwards, each node proceeds as follows:

1. Broadcast an $INIT$ packet containing the local time with a random backoff delay.
2. If an $INIT$ packet is received for the very first time, then cancel all pending transmissions and adjust the local clock according to the time in the packet.
3. If two or more $INIT$ packets are received from different nodes, then adjust the local clock according to the latest one.

Phase 2: SYNC

This phase is periodically initiated every $PERIOD$ time units after the reception of the last $INIT$ or $SYNC$ packet and proceeds according to the following steps:

1. Broadcast an $INITSYNC$ packet containing the local time with a random backoff delay.
2. If an $INITSYNC$ packet is received while waiting the backoff delay, then cancel all pending transmissions and
   a) if the packet contains $< 2f+1$ time values, then append the local time and broadcast the updated $INITSYNC$ packet.
   b) if the packet contains $\geq 2f+1$ time values, then select the median value for drift-rate adjustment and broadcast a $SYNC$ packet containing the median.
3. If a $SYNC$ packet is received for the very first time in the current period, then cancel all pending transmissions, record the time value stored in the packet for drift-rate adjustment, and broadcast the updated $SYNC$ packet.
4. If two or more $SYNC$ packets are received in the current period, then select the largest time value among all packets for drift-rate adjustment and broadcast the revised value in a $SYNC$ packet.

Phase 3: RESYNC

This phase is only initiated by a node p_i, after p_i wakes up from a suspended mode and consequently executes the following steps:

1. Broadcast a $RESYNC$ packet with a random backoff delay.
2. If an $INITSYNC$ or $SYNC$ packet is received in the current period, then cancel all pending transmissions.
3. If a $RESYNC$ packet is received in the current period, then broadcast a $SYNCREPLY$ packet containing the local time with a random backoff delay.
4. If a $SYNCREPLY$ packet is received in the current period, then cancel a possible pending $SYNCREPLY$ transmission and p_i adjusts the local clock according to the time in the packet.

	SR	AD	MSTSP	SGS	TinySeRSync	Sanchez	FFTSP	Our approach
				Classes				
Asymmetric vs. **s**ymmetric	A	S	A	S	A	A	S	S
Internal vs. **e**xternal	I	I	I	I	I+E	I+E	I	I
State vs. **R**ate	S	S	S	S	S	S+R	S	S
Continuous vs. **o**n-demand	C	C	C	O	C	C	C	C
Clock **c**orrection vs. **t**ransformation	C	C	C	C	C	C	C	C
Probabilistic vs. **d**eterministic	D	D	D	D	D	D	D	D
Sender- vs. **r**eceiver-to-receiver	S	S	S	S	S	S	S	S
One-way / **t**wo-way / **n**-way exchange	O	O	T	T	T	N	T[a]	O
Global vs. **L**ocal	G	G	G	G	G	G	G	G
				System assumptions				
Single-hop vs. **m**ulti-hop	M	M	M	S	M	M	M	M
Static vs. **d**ynamic topology	S	D	S	S	S	S	S	D
Clustered / **t**ree-like / **u**nstructured	C	U	C	T	U	C	U	U
Security (external attacks)	X		X	X	X	X		
Resilience (internal attacks[b])	B	C+O	T	B	S	C	C+T	B
Fault **d**etection vs. fault **a**cceptance	D		D	A	A	D	A	A
Clock state correction technqiue[c]	N	A	N	C	N	N	N	A
				Performance				
Message complexity[d]	1	N	1	$3N$	$O(n^2)$		$O(N)$	N
Message size	High	Low	Low	High		Low	Low	Low
Time complexity (Convergence time)	Low	High	Low	Low	Low	Low	Low	High
Average precision (Single-hop)	$< 500\mu s$ [e]		$\sim 5\mu s$ [f]	$47\mu s$ [g]		$16\mu s$ [h]	$2\mu s$ [i]	$< 50\mu s$ [j]
Duty-cycle support		X				X	X	X

Table 4.2: Overview of resilient synchronization protocols.

[a] The authors do not state which communication scheme is used. However, a node requires the knowledge about the estimated message delay which is obtained through two-way exchange.
[b] We use the following abbreviations to distinguish between the different types of faults: (**C**)rash fault, (**O**)mission fault, (**T**)imer fault, Con(**s**)istent fault, (**B**)yzantine fault
[c] We use the following abbreviations: Convergence-(**a**)veraging, convergence-(**n**)onaveraging, (**c**)onsistency technqiue.
[d] The message complexity is declared as the maximum number of broadcasts per round in a single-hop topology comprising N nodes after the initial synchronization phase.
[e] Results are worst case simulation results in the absence of attacks according to [Sun05] with a drift rate $\rho = 10^{-6}$, a delay jitter $\varepsilon = 100\mu s$, and a resynchronization period of two minutes.
[f] Results are according to [CL07] and are based on IEEE 802.11 compliant devices in the absence of attacks.
[g] Results are according to [GPČS08] and are based on Mica2 motes in the absence of attacks.
[h] Results are according to [San07] and are based on Mica2 motes in the absence of attacks.
[i] Results are according to [LJP08] and are based on Micaz motes in the absence of attacks.
[j] Results are based on the simulation of 30 IEEE 802.15.4 compliant nodes in the absence of attacks and a message delay between 2.2ms and 2.7ms.

4.1.8 Discussion

The above presented protocols are selected due to their resilience against different types of faults. To give the reader the opportunity to directly compare these protocols with respect to the introduced clock synchronization issues, we have summarized them in Table 4.2. Additionally we have tried to point out the strengths and weaknesses of the protocols by comparing several performance issues. However, it should be noted that the stated performance values should not be taken seriously, because it is nearly impossible to directly compare all protocols. For instance, the performance evaluation stated in the proposed papers are based on different assumptions and are mostly based on different hardware platforms. Furthermore, it is impossible to compare the precision degradation according to internal attacks, since the protocols assume different failure modes. On this account, a direct comparison would only be serious, if the protocols were evaluated with respect to the same underlying system assumptions. However, since this is not an objective of this thesis, we do not investigate this issue in more detail. The last column in the table corresponds to our proposed synchronization protocol. However, it should be noted that whereas all other protocols solve the clock synchronization problem, our protocol is aimed at pulse synchronization.

We further distinguish between two types of resilient clock synchronization protocols. Those that belong to the first type try to achieve resilience against internal attacks through *fault detection* (*e.g.*, if the time difference exceeds some threshold) and a consequent exclusion. Examples for fault detection mechanisms are discussed in [SZC05]. The other type of protocols achieve resilience through *fault acceptance* where the nodes usually require the knowledge of the maximum number of faulty nodes. Fault acceptance based protocols are mostly based on a Byzantine agreement or approximate Byzantine agreement algorithm.

All presented protocols except the AD protocol assume the existence of a structured network and therefore are not applicable in dynamic network topologies. Only our proposed approach and the AD protocol are fully localized and therefore do not have to maintain or periodically update the global state about the network topology. This makes them appropriate for the use in highly dynamic sensor networks. It should be noted that this is also a typical feature of self-organization.

According to the authors, there exist only three protocols that are resilient to Byzantine attacks. These are the SR protocol, the SGS protocol, and our proposed protocol. Whereas the SR protocol uses detection mechanisms for fault prevention, the other two approaches are based on some kind of Byzantine agreement. However, the SOM algorithm used in the SGS protocol is very inefficient with respect to the computational complexity and requires the exchange of large-sized messages. On this account, the authors state that the SGS protocol is appropriate for networks containing at most 15 sensor nodes. Furthermore, due to the distributed behavior, any cluster member is able to initiate the SGS protocol. Thus, if a single cluster member is compromised, it may frequently initiate a synchronization which consequently leads to a battery depletion attack.

TinySeRSync uses authenticated pairwise synchronization and a single trusted source node that initiates the synchronization for global time synchronization. However, selecting the median out of $2f+1$ clock differences does not provide resilience against Byzantine nodes. Instead, only consistent faults are tolerated since $3f+1$ nodes would be necessary for the worst case attack.

4.2 Distributed Clock Synchronization

In contrast to clock synchronization in wireless systems, research on clock synchronization in wired distributed systems was given a lot of attention during the last decades. Therefore,

there exist much more algorithms and approaches with respect to different system assumptions than in the area of WSNs. It should be noted that the fault-tolerant clock synchronization problem is very similar to the Byzantine agreement problem [LSP82]. An excellent comparison and performance evaluation of several fault-tolerant distributed clock synchronization algorithms is presented by Anceaume *et al*. in [AP98]. In this thesis, we make use of a convergence-averaging algorithm named FTA. On this account we first give a short overview of algorithms that belong to this synchronization type and then discuss the FTA approach in more detail.

In [DLP+83], Dolev *et al*. present two simple fault-tolerant convergence functions for approximate agreement in the presence of f Byzantine faults in a fully connected network. These functions are based on the calculation of the midpoint and the mean among the obtained neighbor values after removing the f highest and f lowest values. Another fault-tolerant convergence function which takes the midpoint of the range of the remaining clock values is presented in [LL84a]. Differential Fault-tolerant Midpoint (DFTM) [FC95] and Differential Fault-tolerant Averaging (DFTA) [ND00] are the improved versions of the midpoint convergence function and the mean convergence function, respectively. Other convergence functions are stated in [Sch87, AP98]. Fault Tolerant Daisy Chain (FTDC) [Lön99] clock synchronization is an alternative approach which exploits the TDMA communication in a broadcast medium such that re-synchronization is performed after each clock reading instead of applying a convergence function on a set of clock readings.

4.2.1 Fault Tolerant Averaging

In [KO87], Kopetz *et al*. analyze the precision of a fault-tolerant clock synchronization algorithm named FTA which is based on the synchronous approximate agreement protocol using the mean convergence function. The algorithm assumes a fully connected network comprising N nodes where at most $f < \lceil N/3 \rceil$ nodes are Byzantine. Additionally, the communication system is assumed to be reliable and the clocks must be initially synchronized. That is, the FTA algorithm is not self-stabilizing. Since FTA is a proactive clock synchronization algorithm, each node performs the following three steps at the end of each round in a synchronous system.

1. Each node p_i collects the clock differences to all neighboring nodes p_j, including itself.
2. If the clock difference to some neighboring node is not obtained, then p_j is faulty and p_i takes some arbitrary default value for p_j. As a result, every node has a set V containing exactly N clock differences.
3. Each node removes the f lowest and f highest clock differences in V and calculates the average among the remaining $N-2f$ values. The result is applied to the local clock.

The worst case precision Π of the FTA algorithm depends on the delay jitter ε, the number of faulty nodes f, and the drift offset Γ as defined in Section 2.2.2:

$$\Pi(N, f, \varepsilon, \Gamma) = \frac{N-2f}{N-3f} \cdot (\varepsilon + \Gamma) \tag{4.7}$$

In [DLP+86], Dolev *et al*. present an improved version of the mean convergence function as used in the FTA algorithm which provides a faster convergence time. Furthermore, they state that no convergence function provides a uniformly faster convergence. In detail, in the synchronous case, after removing the f highest and f lowest deviations, only every f-th value in increasing order is taken for the average calculation.

4.3 Self-stabilizing Pulse Synchronization

Many aforementioned clock synchronization protocols for WSNs assume initially synchronized nodes. However, this is inappropriate in the case of multi-hop networks, since there hardly exist distributed algorithms that guarantee self-stabilization such that all nodes are eventually synchronization, independent of their initial states. This is especially a problem in the case of the pulse synchronization problem. Therefore, the objective of this thesis is to present a self-stabilizing pulse synchronization algorithm for WSNs in the presence of Byzantine faults. For this, a lot of work was done by Dolev *et al*. in the area of wired distributed system. For instance, in [DD05], Dolev *et al*. state how to enhance an existing Byzantine algorithm to become self-stabilizing by the use of pulse synchronization.

In [DW04], Dolev *et al*. present two algorithms that solve the self-stabilizing pulse synchronization problem in the presence of Byzantine faults. The first algorithm assumes a synchronous model where all nodes periodically receive the same common pulse. The second algorithm assumes a bounded-delay model. However, the algorithms assume a fully connected network and both are based on randomization and, consequently, are non-deterministic in the convergence time. The authors state that the algorithms converge in exponential time.

In [DD08], Dolev *et al*. developed the `AB-PULSE-SYNCH` algorithm which is based on the Byzantine agreement protocol `SS-BYZ-AGREE` [DD06] and assumes a bounded-delay model in a fully connected network. It does not require a broadcast primitive, scales very well, and provides an excellent linear convergence time of $O(1)$. The `BALANCED_PULSER` algorithm [DH07a] is a much simpler protocol also based on the `SS-BYZ-AGREE` algorithm and assumes the same bounded-delay model. However, both algorithms have a very high message complexity per cycle due to the use of the `SS-BYZ-AGREE` algorithm and, therefore, is inappropriate for the use in WSNs.

The Byzantine self-stabilizing clock synchronization protocol by Malekpour [Mal06] is deterministic and provides a linear convergence time. The algorithm achieves only a coarse synchronization precision which, however, is precise enough such that other fine clock synchronization algorithms that require initially synchronized clocks can be built on top of the proposed protocol. The only disadvantage is the high message complexity and that the protocol is restricted to fully connected network.

In [DDP08], Dolev *et al*. propose a biologically inspired pulse synchronization protocol named `BIO-PULSE-SYNCH` which is based on the model of a PCO [MS90]. This algorithm is the most appropriate protocol for WSNs, because it is very simple, provides a low message complexity of at most n messages per cycle with small sized messaged, a tight precision of at most the message delay, and assumes a bounded-delay model in a broadcast environment. However, the only disadvantage of this algorithm for the use in WSNs is that the broadcast medium usually does not allow the nodes to broadcast a message at the same time. Consequently, messages will be strongly delayed and even lost in the case of a large number of participating nodes. Additionally, a node is usually deaf during the transmission.

Independently, Werner-Allen *et al*. developed the RFA approach which takes the aforementioned broadcast and deafness problem in WSNs into account [WATP+05]. The algorithm is also based on the PCO model [MS90]. They evaluated the algorithm by simulation with TOSSIM in contrast to several parameter choices in single-hop and multi-hop topologies. In [TAB07], Tyrell *et al*. introduce a time advance strategy based on the PCO model, which takes the delays in wireless systems into account. Similarly to [WATP+05], they incorporate the fact that a node cannot transmit and receive at the same time. Other papers regarding clock or pulse synchronization in wireless networks that are based on the PCO model are [MM96], [TAB06], [BL05], [LW04], [HS05], and [DBR08]. However, none of them consider the presence of Byzantine faults.

In [DRPN07, PDN07, DN08], Degesys *et al*. present a novel alternative approach regarding self-stabilizing pulse desynchronization as formally stated in Definition 12 and is called the `DESYNC` algorithm. The algorithm assumes a single-hop network and works as follows: Each node records the received pulses until it knows the events immediately before and after its own pulse invocation. Afterwards, the node adjusts its phase to the average of both neighboring events. The authors have proven that the nodes eventually enter a pulse desynchronized state.

4.4 Digital Clock Synchronization

Digital clock synchronization can be built upon pulse synchronization in order to establish clock synchronization. In principle, this approach corresponds to pulse synchronization in a synchronous system and was first treated by Dolev *et al*. in [DW04]. Since this thesis does not cover the approach of digital clock synchronization, related work about this type of synchronization are not discussed herein. In [BODH08], Ben-Or, Dolev, and Hoch present a good overview of self-stabilizing digital clock synchronization algorithms in the presence of Byzantine faults.

CHAPTER 5

Design Approach

OVERVIEW

The objective of this thesis is to build an effective synchronization algorithm for global clock synchronization in WSNs. Since pulse synchronization is a very powerful and fundamental primitive that can be used as a building block in other algorithms, we concentrate on the development of a robust and effective pulse synchronization that additionally tolerates Byzantine faults in the case the network already converged to a synchronized pulse state (Definition 31).

Since the RFA is one of the most simple self-stabilizing pulse synchronization protocols that is appropriate for the use in WSNs, we first devise an improved version of the RFA and then modify the algorithm such that it is more robust to erroneous nodes. The main advantage of the resulting algorithm is that, in the absence of rate calibration, it is anonymous. That is, the nodes need not be equipped with unique identifies.

We further propose an algorithm for fault-tolerant rate calibration which can be executed in parallel to the pulse synchronization approach. The rate calibration algorithm is non-anonymous and ensures that each node approximately agrees on the same cycle period even in the presence of Byzantine faulty nodes.

Since the main contribution of this thesis is based on the RFA approach from Werner-Allen *et al*. [WATP+05], Section 5.1 first presents the theory and backgrounds of RFA. Afterwards, Section 5.2 extends the algorithm such that it provides a shorter convergence time and a better precision. This section also introduces a model for clock rate calibration. It should be noted that several parts from our previous work in [LE09] are reused and refined in this section. In Section 5.3, we discuss how to make the extended RFA approach more robust against erroneous nodes which do not collide or perform attacks in an adversary manner, but are still able to send different wrong messages to each distinct neighbor. Therein, we also enhance the RFA approach in order to improve the achievable precision by dynamically switching to the FTA synchronization algorithm. A technique that reduces the convergence time in fully connected network topologies is discussed in Section 5.4.1. Further improvements in single-hop networks are presented in Section 5.4.

5.1 Reachback Firefly Algorithm

The RFA was introduced in [WATP+05] and supports scalability, graceful degradation, and a low computation complexity. The algorithm can be classified as a self-stabilizing distributed proactive pulse synchronization algorithm. The concept is based on the PCO phase advance synchronization model [MS90], but with the difference that it is more appropriate for the practical implementation in wireless networks. For instance, the following assumptions from the original PCO model make a practical application very difficult: 1. The oscillators have identical dynamics, 2. Nodes can instantaneously invoke pulses, 3. Every pulse is observed immediately, 4. All computations are performed perfectly and instantaneously.

To understand the principle behind the main concept of the PCO model, consider the following simple example: Assume two persons A and B want to synchronize their wrist watches but can only inform the others if the own watch indicates twelve o'clock. Let c_A and c_B denote the time of the persons' clocks. Every time a person is notified, it advances the own watch by a factor (in our example 1.25) to at most twelve o'clock. Consequently, the higher the multiplication factor, the faster the clocks converge, but the system becomes less robust to faulty notifications. This algorithm describes the simplified phase advance synchronization model of the fireflies, which is described in more detail in the next section. Based on the initial configuration c_A =12:00 and c_B=08:00, Table 5.1 shows that after 5 periods the clocks are synchronized.

Table 5.1: A demonstration of the PCO model. The columns correspond to the ongoing time sequence.

c_A	12:00	02:00 → 02:30	12:00	00:08 → 00:10	12:00
c_B	08:00 → 10:00	12:00	09:30 → 11:52	12:00	11:50 → 12:00

However, in the case all clocks are synchronized, they all indicate the synchronization event at the same time. Using wireless broadcast communication, this implies a high probability of message collisions and the problem that transmitters cannot receive messages while being in transmission mode[1].

The problem can be bypassed by sending the synchronization messages with a random offset, while packing the chosen offset into the message. The receiver can then reconstruct the intended synchronization instant and perform a clock adjustment with respect to the received offset values. Obviously, this random offset results in an out-of-order reception of synchronization messages which causes a problem in the case of the simple synchronization approach as demonstrated above. A solution to this problem is to gather all synchronization events until reaching the period end and then react to the received time information from the last period. This idea was introduced in [WATP+05] and is called *reachback response*.

The formal description is based on the fact that each node p_i is equipped with a pulse clock that has a cycle period of T time units. Let $\phi_i \in [0,1]$ be the corresponding phase variable. We further denote the event where a node invokes a pulse to be a *firing event*. Let $x = f(\phi)$ denote the *state variable* which corresponds to the charge of a firefly [Buc88]. In [MS90], Mirollo and Strogatz have proven that the *state function* $f : [0,1] \rightarrow [0,1]$ must be a smooth, monotonically increasing, and concave down function in order to achieve synchronicity. Therefore, the authors have stated a general state function as shown in Equation 5.1 where the form of the curve depends on a parameter named *dissipation factor*, de-

[1] In [TAB08], the authors discuss this deafness problem in the context of slot synchronization.

noted by b, and determines the extent to which $f(\phi)$ is concave down. Figure 5.1 visualizes the state function for different dissipation factors.

$$f(\phi) = \frac{1}{b} \cdot ln(1 + [e^b - 1] \cdot \phi) \text{ with } b > 0 \tag{5.1}$$

The coupling between the oscillators is defined by the *firing function* $g(\phi)$ and depends on

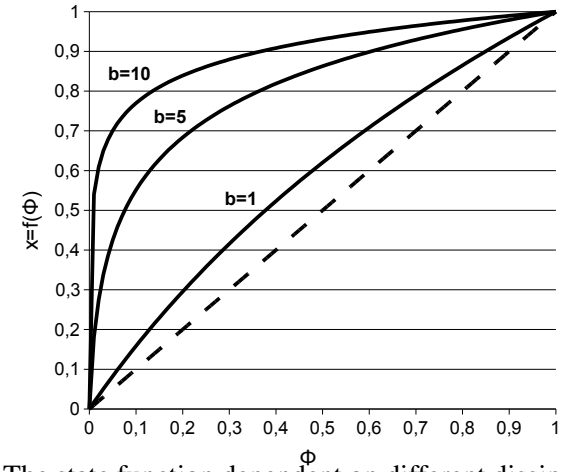


Figure 5.1: The state function dependent on different dissipation factors.

the state function and the *pulse strength* ε, where f^{-1} denotes the inverse state function:

$$\phi_{new} = g(\phi) = \min(1, f^{-1}(f(\phi) + \varepsilon)) \tag{5.2}$$

The firing function is calculated immediately after an oscillator receives a pulse (or flash in case of a firefly). We further use the term of *phase advance* to define the increase in the phase-domain, denoted by $\Delta(\phi) = g(\phi) - \phi$. Due to the concave down state function, a constant addition in the state-domain results in a variable increase in the phase-domain such that a phase advance in the beginning of a cycle is smaller than later in the cycle.

To combat the assumption problems of the PCO model in wireless networks, the RFA makes use of a *reachback response* and *pre-emptive message staggering* technique.

Reachback response. In the original PCO model, an oscillator immediately reacts to each firing event. In contrast, the *reachback response* records the timestamps of all received firing events and calculates an overall phase jump once at the end of each period which is then applied at the beginning of the next cycle. Thus, if a node reaches the period end, it "reaches back in time" and reacts to the firing events of the past period. This principle is visualized in Figure 5.2.

Pre-emptive message staggering. In the case of a synchronized network, all nodes in the PCO model will trigger the transmission event for the synchronization message at the same time. As a result, the messages will collide and the collision avoidance mechanism of the CSMA/CA scheme takes effect resulting in a high delay jitter and even message loss. This delay jitter can be reduced by using MAC time-stamping. However, in order to relax the problem of message collisions, the RFA approach introduces an additional random time offset at the application layer such that a node broadcasts a synchronization message some random time before reaching the period end.

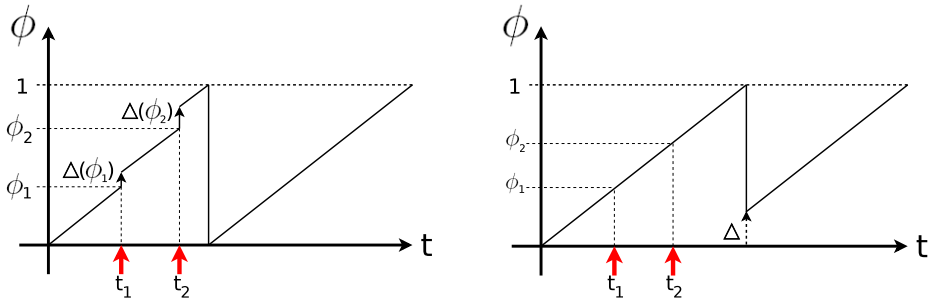


Figure 5.2: Comparison of the original PCO model (a) and the RFA (b). In the PCO-model, an oscillator immediately reacts to a firing event. In contrast, The RFA applies the overall phase jump at the beginning of the next cycle: $\Delta = \Delta(\Delta(\phi_1) + \phi_2)$.

5.2 Improved Pulse Synchronization using RFA

5.2.1 Pulse Synchronization

Similarly to the original RFA approach, we reuse the definition of the smooth, monotonically increasing, and concave down state function of Equation 5.1 to calculate the overall phase advance Δ. Consider that the dissipation factor $b > 1$ and the pulse strength is within $0 < \varepsilon < 1$, then the phase advance equals

$$\Delta(\phi) = \min(1, f^{-1}(f(\phi) + \varepsilon)) - \phi. \tag{5.3}$$

The direct implementation of all these functions would result in a time-consuming calculation process. Therefore, we simplified the equation by inserting the inverse function $f^{-1}(x) = \frac{e^{bx}-1}{e^b-1}$ in Equation 5.3. Let $\alpha = e^{\varepsilon b}$ and $\beta = \frac{\alpha-1}{e^b-1}$, then Equation 5.3 can be transformed to

$$\Delta(\phi) = \min(1, \alpha \cdot \phi + \beta) - \phi. \tag{5.4}$$

Assuming a strong dissipation factor $b \gg 1$ and a small pulse strength such that $0 < \varepsilon \ll b^{-1}$, then we can replace $e^{\varepsilon b}$ by the first order approximation of the Taylor expansion $1 + \varepsilon b$ and thus β is negligible. The phase advance then can be reduced to

$$\Delta(\phi) = \min(1, \alpha \cdot \phi) - \phi. \tag{5.5}$$

As a result, we have a linear Phase Response Curve (PRC) where the *coupling factor* α specifies the strength of coupling between 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].

In contrast to the original RFA algorithm, our approach achieves a better synchronization precision and a faster convergence time by indirectly performing a clustering of the received firing events. This is done by ignoring all events which are within the phase advance of the last event to which a node would react. In fact, this corresponds to the introduction of a short *refractory period*. Additionally, we do not allow a node to react to firing events which originally (*i.e.*, if the synchronization message would be sent without a random time offset) would occur after the node reaches the period end. This ensures that in the case of synchronized nodes, the fastest node does not advance its phase anymore, resulting in a better precision. The algorithm is formally analyzed in more detail and guarantees network synchronization as long as the bounds for several parameters are maintained. Algorithm 5

explains the behavior of this extended RFA (E-RFA) algorithm by the use of pseudocode. The refractory period is implemented by the condition in Line 9. The variable *eventset* contains the corrected phase of all received firing messages and offset_i denotes the random time offset for the preponed transmission with at most the maximum message staggering delay r_{msd}^{max} and at least the minimum message staggering delay r_{msd}^{min}. Note that we always assume a normalized threshold of 1. In reality, a pulse clock is usually implemented by the use of a hardware clock that periodically increments a counter up to some threshold Φ_{th}. If necessary, we use $\Phi_{msd}^{max} = r_{max} \cdot \Phi_{th}$ and $\Phi_{msd}^{min} = r_{min} \cdot \Phi_{th}$ to denote the absolute maximum and minimum message staggering delay with respect to Φ_{th}, respectively.

Algorithm 5: E-RFA: code for p_i, $0 \leq i < n$

```
1  Init: eventset := ∅, Δ_i := 0, φ_i := 0, offset_i := 1 − random(r_msd)
2  upon event φ_i(t) = 1 − offset_i do                    // preponed transmission
3      trigger broadcast_i(φ_i(t))  // broadcast current phase to all neighbors
4  upon event recv_i(φ_j) from p_j do                     // received sync-message
5      if φ_i(t) − φ_j < 0 then                                // check timeliness
6          add (φ_i(t) + 1 − φ_j) to eventset
7  upon event φ_i(t) = 1 do                                  // threshold reached
8      φ_last := δ_last := Δ_i := 0                                    // clean up
9      for each event φ_j ∈ eventset in increasing order do
10         if Δ_i + φ_j < 1 and φ_last + δ_last < φ_j then
11             δ_last := min(1, (φ_j + Δ_i) · α) − (φ_j + Δ_i)
12             Δ_i := Δ_i + δ_last
13             φ_last := φ_j
14     φ_i(t) := Δ_i                                  // Apply reachback response
15     offset_i := random(r_msd^max − r_msd^min) + r_msd^min   // Calculate firing offset
16     eventset := ∅
```

Lower bound for the coupling factor α. We now show that in the case of two non-faulty nodes that are initially synchronized, Algorithm 5 maintains the synchronized pulse state with a worst case precision as stated in Definition 11. That is, the precision is defined with respect to real-time (*i.e.*, Newtonian time). For the following proofs, we define $R = \frac{1+\rho}{1-\rho}$ and $\Gamma = 2\rho T$ be the drift offset as stated in Section 2.2.2.

Lemma 1. *In the case of a fault-free fully connected system comprising two nodes, if* $\alpha > \left(1 - r_{max} \cdot (R-1) - \frac{\Pi^U - d}{T \cdot (1-\rho)}\right)^{-1}$ *and* $\rho < \frac{1}{7}$*, then for* $\frac{\Pi^U + d + \varepsilon}{T \cdot (1-\rho)} < r_{min} \leq r_{max} < \frac{1}{2}$*, Algorithm 5 keeps the network synchronized with a worst case precision of*

$$\Pi^U = (1 + r_{max})\Gamma + \varepsilon \cdot R + \max(\Gamma \cdot r_{max}, d \cdot R). \tag{5.6}$$

Note that in the case of a fully connected network comprising more than two nodes, all nodes synchronize to the fastest one due to Line 4 and Line 9 of Algorithm 5. Especially the condition $\varphi_{last} + \delta_{last} < \varphi_j$ in Line 9 ensures that if a node advances its phase due to some received firing event φ_j, all events immediately follow some short time after φ_j are ignored. This condition is necessary. Otherwise, assume n nodes are perfectly synchronized. Consequently, a node would perform n times a phase advance, which results in a mutual excitation in the case n is very large.

Theorem 1. *In the case of a non-faulty communication network comprising $n \geq 2$ non-fault nodes, if $\alpha > \left(1 - r_{max} \cdot (R-1) - \frac{\Pi^U - d}{T \cdot (1-\rho)}\right)^{-1}$ and $\rho < \frac{1}{7}$, then for $\frac{\Pi^U + d + \varepsilon}{T \cdot (1-\rho)} < r_{min} \leq r_{max} < \frac{1}{2}$, Algorithm 5 keeps the network synchronized with a worst case precision of*

$$\Pi^U = (1 + r_{max})\Gamma + \varepsilon \cdot R + \max(\Gamma \cdot r_{max}, d \cdot R). \tag{5.7}$$

Upper bound for the coupling factor α. One may ask why not setting $\alpha = \infty$ such that a node immediately adjusts its phase to a neighboring clock every time receiving a firing message from this clock. However, the following lemma shows that there exist a basic upper bound which holds for every network. For this we first define the notion of a *firing state*.

Definition 32. *A firing state $C(N,k,m) = (\varphi_{0,m}, \varphi_{1,m} \ldots \Delta_{k,m} \ldots \varphi_{n-1,m}) = P(t)$ of a fully connected network N comprising n nodes with $\varphi_{i,m} = \varphi_i(t)$ is defined to be the pulse state $P(t)$ of the system at time t when p_k just reached the threshold for the m-th time and consequently applied the phase advance $\Delta_{k,m}$.*

Lemma 2. *In a perfect fully connected communication network N comprising $n = 2$ perfect nodes, if the coupling factor $\alpha \geq \frac{3}{2}$, then the nodes may never become pulse-synchronized.*

Since the algorithm ignores all firing events immediately following some short time after a previous firing event due to Line 9, a node may realize a set of nodes as a single node and therefore Lemma 2 also applies to networks comprising more than two nodes. We now exploit the intuition behind Lemma 2 and extend this problem to a general network comprising $n \geq 2$ nodes.

Definition 33. *$C(N,k,m)$ is called to be an infeasible firing state, if there exists a positive integer $i > 0$ such that $C(N,k,m) = C(N,k,m+i)$ and the network is not synchronized.*

Lemma 3. *The maximum phase advance a node can perform in a perfect fully connected communication network N comprising n perfect nodes equals $\Delta = \frac{(2\alpha-1)^{n-1}-1}{(2\alpha-1)^{n-1}+1}$.*

A weak upper bound results from the fact that we do not want a node to perform a phase advance which is greater than $1/2$ and directly follows from Lemma 3.

Corollary 1. *In a perfect fully connected communication network comprising $n \geq 2$ perfect clocks, if the coupling factor $\alpha < \frac{\sqrt[n-1]{3}+1}{2}$, then in every admissible execution a node will never perform a phase advance which is greater than $1/2$.*

Note that if the weak bound is maintained, it can be still shown that there exist infeasible firing states. However, due to imprecisions in calculations, the varying short-term drift, the delay jitter, and due to several other indeterministic environmental effects, this bound is generally applicable in fully connected networks. By contrast, multi-hop networks especially ring topologies have to maintain the stronger bound as discussed next.

A stronger bound was devised from empirical studies in fully connected networks which have shown that infeasible firing states highly likely do not exist, if the maximum phase advance $\Delta_{max} < \frac{1}{n+1}$. The resulting bound for α can be deduced from Lemma 3.

Theorem 2 (unproven)**.** *In a perfect fully connected communication network comprising $n \geq 2$ perfect clocks, if $\alpha \geq \frac{1}{2}\left(1 + \sqrt[n-1]{1 + \frac{2}{n}}\right)$, then the system may enter a stable infeasible firing state.*

Rate of synchronization. Theorem 3 analyzes the time to sync for the case of two perfect nodes. The authors of [MS90] have also analyzed the case of $n > 2$ nodes. However, considering a multi-hop topology requires a more sophisticated solution and is treated in [LW04]. For the following proofs, we consider a perfect communication network N comprising only two perfect nodes p_A and p_B. Let $\gamma = \alpha - 1$ and $\Phi_0 = \varphi_A - \varphi_B$ denote the initial phase difference between the two nodes with $\varphi_B \leq \varphi_A$.

Lemma 4. *The infeasible firing state $C(N,A) = (\Delta_A^*; \varphi_B^*)$ with $\Delta_A^* = \frac{\alpha-1}{3-\alpha}$ and $\varphi_B^* = \frac{1}{3-\alpha}$ is a unique fixpoint and has a phase difference of $\delta^* = \frac{2-\alpha}{3-\alpha}$.*

Proof. If we set $C(N,A,k+1) = C(N,A,k)$, we get $\Delta_{A,k+1} = \Delta_{A,k} = (\Delta_{A,k} + 1 - \varphi_{B,k}) \cdot (\alpha - 1)$ and $\varphi_{B,k+1} = \varphi_{B,k} = \varphi_{B,k} \cdot \alpha - \Delta_{A,k}$. Thus we can deduce that $\Delta_A^* = \frac{\alpha-1}{3-\alpha}$ and $\varphi_B^* = \frac{1}{3-\alpha}$. □

Although this fixpoint is a repeller, the roundoff error in the calculation may cause a node to enter the fixpoint. This is especially a concern if the granularity of the hardware clock is very low. The rate of sync with respect to different initial phase differences is visualized in Figure 5.3. It is obvious that there exist a special initial configuration Φ_0^* which causes the

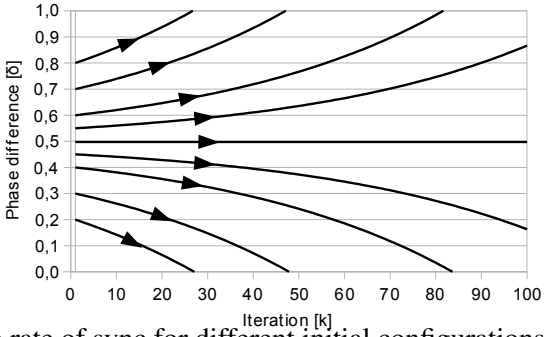


Figure 5.3: The rate of sync for different initial configurations with $\alpha = 1.01$.

network to enter this fixpoint. To analyze this initial configuration, we first transform the recursion of the dynamic system into a closed term.

Lemma 5. *The phase difference δ_k of $C(N,A,k)$ for $k \geq 1$ equals*

$$\delta_k = \delta^* + ((A_2 \cdot \gamma(1-\gamma) - A_1) \cdot z_1^{-k} + (B_2 \cdot \gamma(1-\gamma) - B_1) \cdot z_2^{-k})/\gamma^2 \tag{5.8}$$

where $z_1 = \frac{1+2\gamma+\sqrt{1+4\gamma}}{2\gamma^2}$, $z_2 = \frac{1+2\gamma-\sqrt{1+4\gamma}}{2\gamma^2}$, $A_1 = \frac{1-\Phi_0-z_1\gamma}{z_1-z_2}$, $B_1 = \frac{z_2\gamma-1+\Phi_0}{z_1-z_2}$, $A_2 = \frac{z_1^2}{(1-z_1)\cdot(z_1-z_2)}$, $B_2 = \frac{-z_2^2}{(1-z_2)\cdot(z_1-z_2)}$, and δ^* from Lemma 4.

Proof. Let $C(N,A,1) = (\Delta_{A,1}, \varphi_{B,1})$ be the initial firing state with $\Delta_{A,1} < \varphi_{B,1}$ where $\Delta_{A,1} = 0$ and $\varphi_{B,1} = 1 - \Phi_0$. The phase difference when p_A reached the threshold for the k-th time is $\delta_k = \varphi_{B,k} - \Delta_{A,k}$. From Lemma 2 we know that $C(N,A,k+1) = (\Delta_{A,k+1}; \varphi_{B,k+1})$ with $\Delta_{A,k+1} = (\Delta_{A,k} + 1 - \varphi_{B,k}) \cdot (\alpha - 1)$ and $\varphi_{B,k+1} = \varphi_{B,k} \cdot \alpha - \Delta_{A,k}$. If we substitute γ for $\alpha - 1$ and consider the phase difference δ_k of $C(N,A,k)$, we get $\Delta_{A,k+1} = \gamma(1-\delta_k)$ and $\varphi_{B,k+1} = \delta_k + \gamma \cdot \varphi_{B,k}$ which yields $\delta_{k+2} = \delta_{k+1} \cdot (1+2\gamma) - \delta_k \cdot \gamma^2 - \gamma(1-\gamma)$ for $k \geq 1$. The dissolving of the recursion is left to the reader and leads to the solution as stated. □

Lemma 6. *There exist a unique initial phase difference $\Phi_0^* \in (0,1)$ where the network eventually enters the fixpoint of Lemma 4 and equals $\Phi_0^* = 1 - z_2\gamma \cdot \frac{1-z_2\gamma}{1-z_2}$ with z_2 from Lemma 5.*

Proof. If the network enters the fixpoint in $C(N,A,m)$ for some $m > 1$, then we have a phase difference of $\delta_k = \delta^*$ for $k \geq m$ with δ^* from Lemma 4. Using Equation 5.8 then yields

$\left(\frac{z_2}{z_1}\right)^{k+1} = -\frac{B_2\cdot\gamma(1-\gamma)-B_1}{A_2\cdot\gamma(1-\gamma)-A_1}$. Since $z_1 > z_2$ we get $\lim_{k\to\infty}\left(\frac{z_2}{z_1}\right)^{k+1} = 0$ and thus $B_2\cdot\gamma(1-\gamma) = B_1$. Using B_1 and B_2 from Lemma 5 results in $\delta_1 = z_2\gamma\cdot\frac{1-z_2\gamma}{1-z_2}$. The initial phase difference then has to be $\Phi_0^* = 1-\delta_1$ as stated. □

Theorem 3. *The number of iterations k until synchrony is at most $k \leq \log_{z_2}\frac{B_2\cdot\gamma(1-\gamma)-B_1}{(\delta^*-\delta)\cdot\gamma^2}$ with B_1, B_2, and z_2 from Lemma 5 and*

$$\delta = \begin{cases} 1 & \text{if } \Phi_0 \leq \Phi_0^* \\ 0 & \text{if } \Phi_0 > \Phi_0^* \end{cases}. \tag{5.9}$$

Proof. Note that $\lim_{k\to\infty} C(N,A,k) = (\Delta_{A,k}, \varphi_{B,k})$ either converges to $(0,0)$ or $(1,1)$ as visualized in Figure 5.3. Therefore, we simply equate Equation 5.8 with 1 if $\Phi_0 \leq \Phi_0^*$ or with 0 if $\Phi_0 < \Phi_0^*$. Since $z_1 > 7$ for $\alpha < \frac{3}{2}$ and the multiplicative factor is smaller than 1, the term with respect to z_1^{-k-1} does not influence the rate of sync for larger k and hence can be neglected. This leads to the equation as stated. □

Note that Theorem 1 validates the closure condition and Theorem 3 the convergence condition of the self-stabilizing pulse synchronization problem for a fault-free system comprising $n = 2$ nodes.

So far, we have only considered the convergence time for the case of two nodes where the pulse state repels from a single fixpoint and converges to a stable fixpoint. Fortunately, we can assume that a system highly likely exits an unstable fixpoint due to the inaccuracies in calculation, delay jitter, and other imprecisions. Therefore, for the rest of this thesis we assume that a system always exits an unstable fixpoint configuration which allows the definition of eventual convergence.

However, a generalization of Theorem 3 for $n > 2$ requires much more mathematical insights and is beyond the scope of this thesis, because beside the stable fixpoint of the synchronized pulse state, a fully connected network with $n > 2$ contains much more fixpoints which are very complex to identify. Fortunately, due to our assumption that the system never keeps in an unstable fixpoint configuration, we can assume that the system eventually enters the synchronized pulse state as long as no other infeasible firing states and consequently no other stable fixpoint configurations do exist. For that reason, we have to introduce lower and upper bounds for all parameters which ensure that no stable fixpoint configuration except the synchronized pulse state do exist.

Note that the persistence of the system in one of the aforementioned unstable fixpoint configurations can be also treated as a symmetry breaking problem. As we have stated, the exact identification of such configurations in a fully connected network is too complex for the case of $n > 2$. However, in the case of a simple ring topology, such unstable fixpoint configuration can be simply identified. In detail, assume a ring topology containing $n \geq 2$ processors $p_0, p_1, \cdots, p_{n-1}$. Without loss of generality assume that at $t = 0$, p_0 just reached the period end and started a new round with a phase advance of $\Delta_0 = \min(1, \alpha\cdot(n-1)/n) - (n-1)/n$. Further assume that, afterwards, $\varphi_0 < \varphi_1 < \cdots < \varphi_{n-1}$ with $\varphi_k = k/n$ for $1 \leq k < n$. The next node which reaches the period end then is p_{n-1} and consequently applies a phase advance of $\Delta_{n-1} = \min(1, \alpha\cdot(n-1)/n) - (n-1)/n$. As a result, every node applies the same phase advance irrespective of the coupling factor α which leads to the fact that the system forever keeps in an infeasible firing configuration. However, the aforementioned unstable fixpoint configurations may be also stable, if the coupling factor is too great. Therefore, we devised Theorem 4 that applies to ring topologies and other multi-hop topologies that contain ring-like subnetworks.

Theorem 4. *Let $d_{max} = d + \varepsilon$ be the maximum possible message delay in a non-faulty communication network as stated in Definition 26 and $c = \frac{1}{n} - \frac{d_{max}}{T} - \frac{2\rho}{1-\rho^2}$. In a fault-free ring system comprising $n \geq 2$ nodes, if $\alpha \geq \frac{1}{1-c}$, then the system may enter a stable infeasible firing state.*

Proof. Assume that the phase of the nodes are distributed in an equidistant way on the ring. Then the distance between two neighboring nodes equals $1/n$. In the case the two neighboring nodes p_i and p_j with $\varphi_i(0) < \varphi_j(0) = \varphi_i(0) + 1/n$ are situated at the beginning of the pulse period, then in the worst case the phase difference between both nodes at the period end after T real time units reduces to $1/n - 2\rho/(1-\rho^2)$ due to the clock drift. We now have to ensure that a node does not perform a phase advance which is greater than $1/n - 2\rho/(1-\rho^2) - d_{max} = c$. Otherwise, each node p_i will only change its position to the phase of the next node p_{i+1} minus the maximum message delay d_{max} and the synchronized pulse state will never be entered. Therefore, we require that $(1-c) \cdot \gamma < c$ and consequently $\alpha < 1/(1-c)$. □

Note that in the case of a ring topology, the tight upper bound for the coupling factor as stated in Theorem 2 is no more valid and since the aforementioned identified upper bound of Theorem 4 is more tight than the weak bound identified in Corollary 1, Theorem 4 is the only upper bound for the coupling factor for ring topologies.

5.2.2 Rate Calibration

The concept of clock rate calibration combats the problem of frequency deviations due to the high clock drift of the RC-oscillators usually used in low-cost devices. This approach should allow a longer resynchronization interval with the same synchronization precision. Note that the rate correction can be performed completely independent from the clock state correction scheme.

The core concept of our rate calibration algorithm is that a processor p_j implements a pulse clock PC_j with the threshold value Φ_{th} as defined in Algorithm 2 such that $PC_j(t) \in [0, \Phi_{th} - 1)$. Let HC_j be the underlying hardware clock. By adjusting $T_j^{th} = T_{nom} + H_j$, the period time of the pulse clock can be increased or decreased. Note that T_{nom} is a constant that determines the nominal number of microticks and H_j is the adjustable variable which can be set by an algorithm. Let $h_j(t) = H_j(t)/T_{nom}$ be the corresponding *relative adjustment value*.

In order to perform the rate calibration, every node p_j periodically broadcasts a synchronization message m_j. Let $PC_j(m_j)$ and $HC_j(m_j)$ denote the timestamps of the pulse clock and hardware clock at p_j, respectively, at the time when p_j broadcast m_j. Let $PC_r(m_j)$ and $HC_r(m_j)$ denote the timestamps of the pulse clock and hardware clock at p_r, respectively, at the time when p_r received m_j from p_j. Let $m^r_{j,k}$ be the k-th message that p_r received from p_j and $m_{j,k}$ be the k-th message p_j broadcast. We further assume that $m_{j,k+1}$ is not received at some p_r before $m_{j,k}$ is received for $k \geq 1$ and that each message contains all necessary timestamps.

The rate correction algorithm works as follows: Based on the timestamp $HC_j(m^r_{j,k})$ stored in $m^r_{j,k}$, the receiving processor p_r calculates p_j's relative adjustment value in its own granularity, denoted by $h^r_{j,k+1}$, with $h^r_{j,k+1} = \frac{HC_r(m^r_{j,k+1}) - HC_r(m^r_{j,k})}{(HC_j(m_{j,k+1}) - HC_j(m_{j,k}))/(1+h_{j,N})} - 1$. Therein, the term $h_{j,N}$ denotes the latest received adjustment value from p_j, *i.e.*, the relative adjustment value contained in the latest received message $m_{j,N}$ from p_j. In order to reduce the impact of the delay jitter, we should choose the time interval between the two received messages as large

as possible[2]. However, the optimal time interval also depends on the underlying oscillator type. In our case, we store the last N received messages from each node and calculate the relative deviation with respect to the buffer size N. To visualize the impact of the delay jitter, we replace $HC_r(m^r_{j,k})$ by $HC_r(m^r_{j,k}) + d_k$, where d_k corresponds to the message delay of message $m^r_{j,k}$ in p_r's clock granularity. From this it follows

$$h^r_j = \frac{HC_r(m^r_{j,N}) - HC_r(m^r_{j,1})}{(HC_j(m^r_{j,N}) - HC_j(m^r_{j,1}))/(1+h_{j,N})} - 1 + \frac{(d_N - d_1)\cdot(1+h_{j,N})}{HC_j(m^r_{j,N}) - HC_j(m^r_{j,1})}. \tag{5.10}$$

In our implementation we set $N = 8$, reducing the impact of the jitter with respect to the resynchronization period to at about $\sim \varepsilon/8$.

Let P_r be a set of processors that are within the broadcast domain of p_r such that for all $p_j \in P_r$, p_r is in the broadcast domain of p_r and $p_r \notin P_r$. The next step of the algorithm is to calculate the average relative phase adjustment value of all nodes $p_j \in P_r$, *i.e.*,

$$\bar{h}^r = \frac{h_r + \sum_{p_j \in P_r} h^r_j}{|P_r| + 1}. \tag{5.11}$$

The main challenge, however, concerns the adjustment of $\bar{h}^r$ to the new relative adjustment value h^r_{new} of p_r. Due to natural imprecisions and the influence of the delay jitter, a direct adjustment of $h^r_{new} = \bar{h}^r$ usually leads to a continuous increase or decrease of the real overall average relative adjustment value. This effect is also known as the *common-mode drift*. In general, the common-mode drift cannot be avoided, but the effect can be reduced by carefully choosing a large enough N with respect to the delay jitter. A further approach for the reduction of the common-mode drift depends on the parametrized adjustment of $\bar{h}^r$ by the use of a smoothing factor σ as shown in Equation 5.12. This ensures that the pulse clocks smoothly converge to the same overall average cycle time.

$$h^r_{new} = h^r_{old} + (\bar{h}^r - h^r_{old}) \cdot \sigma. \tag{5.12}$$

Note that there is a tradeoff between convergence time and rate stability with respect to the smoothing factor. For instance, a smaller smoothing factor results in smaller rate variations, but increases the convergence time. In contrast, a greater smoothing factor decreases the convergence time, but results in wider rate variation. Empirical tests and simulations have shown that a value of $\sigma = \frac{1}{4}$ is a good compromise between the extended convergence time and a better rate stability.

In order to overcome the common-mode drift, we developed a drift stabilization approach which makes use of the calculated *slope* $s_r(t, N_s)$ of the adjustment value $h_r(t)$. In detail, each node p_r calculates $s_r(t, N_s)$ at time t, based on the gathered adjustment values of $h_r(t)$ during the last N_s synchronization periods. Let $h_{r,k}$, $1 \leq k \leq N_s$, be the gathered values at some p_r where $h_{r,1}$ and h_{r,N_s} terms the oldest and latest adjustment value of the current synchronization round, respectively. The assignment of the new adjustment value h^r_{new} is then done with respect to the calculated slope as shown in Equation 5.13.

$$h^r_{new} = \bar{h}^r - s_r(t, N_s). \tag{5.13}$$

The slope is calculated by assuming a linear regression model and using the linear least square computation technique for minimizing the sum of squared residuals. This leads to the following slope calculation equation:

$$s_r(t, N_s) = \frac{\sum_{k=1}^{N_s} (k - (N_s+1)/2) \cdot (h_{r,k} - h_{avg})}{\sum_{k=1}^{N_s} (k - (N_s+1)/2)^2} \quad \text{with} \quad h_{avg} = \frac{1}{N_s} \sum_{k=1}^{N_s} h_{r,k} \tag{5.14}$$

[2]Note that there still exist an upper limit due to the long-term stability of an oscillator, which is usually in the order of minutes.

One may assume that this recalibration will stop the common-mode drift at some time. However, the fact that the nodes have different time basis and the existence of imprecisions and jitter again invalidates the aforementioned argument. Nevertheless, the common-mode drift is dramatically reduced with the advantage of an unaffected convergence speed in contrast to the smoothing approach.

One way to completely overcome the common-mode drift is that the nodes incorporate the drift deviation with respect to some globally fixed reference value. Note that a solution solely based on locally fixed reference values (*e.g.*, the local drift adjustment value at some fixed time t) requires that all nodes fix their reference value at nearly the same time which would result in complex algorithms. Therefore, the agreement on the globally fixed reference value is based on continuously averaging the averaged locally fixed reference values of all neighbors and the own node. For this, the locally fixed reference value is based on the integrated slope over time since a node entered the sync-state.[3] Let $\Sigma_{s_r}(t)$ denote the *summarized slope* at time t. In detail, at the end of each synchronization period, the actual calculated slope $s_r(t,N_s)$ is added, *i.e.*, $\Sigma'_{s_r}(t) = \Sigma_{s_r}(t) + s_r(t,N_s)$. The basic idea then is to additionally incorporate this "integrated" slope into the calculation of the new drift adjustment value as follows:

$$h^r_{new} = \bar{h}^r - s_r(t,N_s) - \Sigma'_{s_r}(t)/N_s \tag{5.15}$$

The only problem in Equation 5.15 results from the fact that there may exist some node p_x which started the integration process for $\Sigma_{s_r}(t)$ much more earlier than the other nodes. As a consequence, the summarized slope value of this node can become exploding high according to the other nodes. In the presence of a multi-hop topology and the case that all nodes except p_x have a very small summarized slope value, then p_x will always perform a too excessive rate adjustment. Indeed, the common-mode drift is stopped, but at the price of a possible strong rate instability.

A remedy to this problem is based on keeping all summarized slope values at nearly the same global level. The simplest approach to do this is by implementing a distributed averaging mechanism which incorporates all neighboring summarized slope values. According to the set of processors P_r, p_r then calculates the new summarized slope as follows, where s_j denotes the received summarized slope of p_j of the last synchronization period:

$$\Sigma'_{s_r}(t) = \frac{\Sigma_{s_r}(t) + \sum_{p_j \in P_r} s_j}{1 + |P_r|} + s_r(t,N_s) \tag{5.16}$$

5.3 Introducing Robustness and Fault Tolerance

In this section, we modify the proposed E-RFA algorithm and the rate calibration scheme such that both are robust or even tolerant to f Byzantine faults. For this, we assume that the system is fault-free in the case of E-RFA and coherent in the case of the rate calibration scheme. In the coherent case, in Definition 31 we stated that a non-faulty node receives at most one message from a faulty node in each period. This simplification is necessary in order to maintain the determinism of the algorithm. Furthermore, this constraint is feasible since we do not assume the existence of radio jamming attacks.

The robust version of the E-RFA algorithm works in single-hop topologies in the presence of at most f permanent erroneous nodes, if the number of nodes $n \geq 5f+1$. The algorithm is also admissible in constraint multi-hop networks as long as the network is $(5f+1)$-connected. Note that the aforementioned assumption of a single-hop network is a special case of a $(5f+1)$-connected network. We will also show that there exist executions where

[3] Note that choosing a reference value which is directly based on the local drift adjustment value is inappropriate, because all nodes may have strongly deviating initial adjustment values.

the algorithm will never converge in a coherent system. Hence, the R-RFA algorithm is not resilient to Byzantine faults, but robust against erroneous nodes which do not behave in an adversary manner.

5.3.1 Robust RFA

The first idea is to incorporate the same concept used in the FTA algorithm. In detail, before a node calculates the phase advance, it first removes the f lowest and f highest phase deviations from the multiset variable *eventset*. This behavior is illustrated in Algorithm 6. Note that in the case of a wired distributed system, the assumption of $n \geq 3f+1$ is adequate. Unfortunately, in wireless networks our assumption of a Byzantine node allows it to jam the medium in order to destroy the message transmission of at most one non-faulty node. We assume the same for an erroneous node. Consequently, in the worst case the f erroneous nodes may always prevent f non-faulty nodes from broadcasting their messages. However, if a node receives only $2f$ messages in the case of $n = 3f+1$, then we cannot assume that all messages origin from non-faulty nodes due to the unreliability of the wireless communication channels. On this account, a node always removes the f lowest and f highest phase deviations independent of the number of received messages. In order to provide robustness with respect to the previous mentioned attack, this fact then requires the assumption of $n \geq 5f+1$ or according to general network topologies a $(5f+1)$-connected network.

Note that the phase deviations are symmetric with respect to $\pm\frac{1}{2}$. On this account we define the set S of *symmetric phase deviations* according to the stored events in the *eventset* variable as defined below. For this we use the *symmetrization function* $s(\varphi_j)$.

$$S(eventset) = \{s(\varphi_j) \mid \varphi_j \in eventset\}, \text{ with } s(\varphi_j) = \begin{cases} \varphi_j & \text{if } \varphi_j < \frac{1}{2} \\ \varphi_j - 2 & \text{if } \varphi_j \geq \frac{3}{2} \\ \varphi_j - 1 & \text{else} \end{cases}. \tag{5.17}$$

Let $j_{min} = \min_{\forall \varphi_j}\{j \mid s(\varphi_j) = \min(S)\}$ and $j_{max} = \max_{\forall \varphi_j}\{j \mid s(\varphi_j) = \max(S)\}$ be the smallest and greatest array indices of the subset of phase deviations that have the same minimum or maximum symmetric phase deviation, respectively. Note that the algorithm uses the array indices to refer to the distinct received phases. This does not mean that each node requires a unique identifier. Consequently, we can define the function *reduce* as follows:

$$reduce(eventset) = eventset \setminus \{\varphi_{j_{min}}, \varphi_{j_{max}}\}.$$

Since we want to remove a set of lowest and highest deviations, we use $reduce^k$ to denote the k-fold iteration of the function *reduce*.

A lot of experimental studies have shown that this approach nearly always converges in the presence of simple erroneous nodes (*e.g.*, omission failure) as long as the parameters are correctly chosen according to Theorem 1 and Theorem 2. However, if the erroneous nodes act in an adversary manner (*i.e.*, they are Byzantine faulty), then they are always possible to prevent the system from converging. For instance, consider the configuration as visualized in Figure 5.4. Therein, a group of nodes is already synchronized and the node p_i is outside the group. Assume that p_i has a higher drift (*i.e.*, p_i is much more faster) compared to the other nodes in the group and consequently diverges from the group. Let p_f be a Byzantine faulty node which transmits a message to each node in the group in each round with the information as it would be situated exactly c phase units in front the group. Further let Δ_i be the phase advance performed by node p_i in some round. Consequently, if $f = 1$, then p_f can chose a different $c > 1/L$ for each node of the group such that the phase advance Δ_g of all these nodes is the same and equals Δ_i. This leads to the fact that the phase difference between each group node and p_i never changes over time and convergence is never achieved.

Algorithm 6: R-RFA: code for p_i, $0 \leq i < n$, $n \geq 5f+1$

```
1  Init: eventset := ∅, Δ_i := 0, φ_i := 0, offset_i := random(r_msd^max − r_msd^min) + r_msd^min
2  upon event φ_i(t) = 1 − offset_i do                          // preponed transmission
3      trigger broadcast_i(φ_i(t))  // broadcast current phase to all neighbors
4  upon event recv_i(φ_j) from p_j do                           // received sync-message
5      add (φ_i(t) + 1 − φ_j) to eventset
6  upon event φ_i(t) = 1 do                                     // threshold reached
7      φ_last := δ_last := Δ_i := 0                                          // clean up
8      reduce^f(eventset)          // remove f highest and lowest deviations
9      for each φ_j ∈ eventset in increasing order do
10         if Δ_i + φ_j < 1 and φ_last + δ_last < φ_j then
11             δ_last := min(1, (φ_j + Δ_i) · α) − (φ_j + Δ_i)
12             Δ_i := Δ_i + δ_last
13             φ_last := φ_j
14     φ_i(t) := Δ_i                                          // Apply reachback response
15     offset_i := random(r_msd^max − r_msd^min) + r_msd^min   // Calculate firing offset
16     eventset := ∅
```

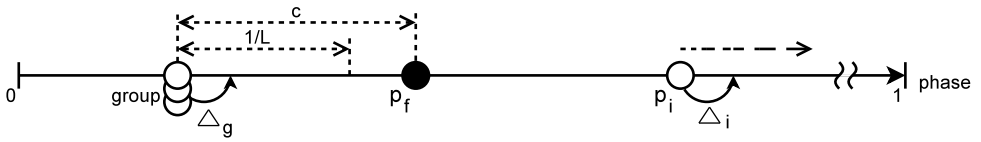


Figure 5.4: Demonstration of a configuration where R-RFA will never converge in a coherent system for all admissible executions.

Beside the problem of convergence, we further show that the closure condition does not hold in the presence of drift.

Theorem 5. *In any coherent system with a fully connected network comprising $n \geq 5f+1$ nodes for $f > 0$, Algorithm 6 does not satisfy the closure condition of the self-stabilizing pulse synchronization problem.*

Proof. Assume that initially all nodes are perfectly synchronized and that all nodes (even the erroneous nodes) initially behave non-faulty. Let p_i be the fastest non-faulty non-erroneous node. Consequently, due to Line 8 the other nodes will exclude p_i from their phase advance calculation. Since a node never advances more than the difference to the fastest node included in the phase advance calculation, and p_i never sets its phase back, the phase difference between p_i and any other node increases after each iteration until the pulse state is no more synchronized. □

As a consequence of Theorem 5 and the fact that the algorithm provides robust convergence in a fault-free system, the nodes periodically enter a synchronized pulse state, keep therein for some time, and then become unsynchronized until the fastest node again comes close to the other nodes.

In order to maintain the synchronized pulse state, we extended the algorithm by the FTA approach. The main advantage of the combination with the R-RFA algorithm is that both can calculate the phase advance solely on the messages stored in the *eventset* variable. Furthermore, we can reuse the formal results of FTA to determine the worst case precision. In other words, we use the R-RFA to provide convergence with a coarse synchronization precision in a fault-free system and if the precision is small enough, the nodes then switch to the FTA

approach which provides a fine synchronization precision in a coherent system. Clearly, the coarse precision must be small enough to validate the assumption of initially synchronized nodes for the FTA algorithm, but must be great enough to have enough time until all nodes switched to the FTA approach. A second reason for the switching to the FTA approach is that FTA provides a much better precision, especially in the fault-free case. In more detail, the worst case precision of FTA improves with an increasing number of nodes participating in the synchronization process. In contrast, the worst case precision of the E-RFA approach is about the maximum message delay and independent of the number of nodes as stated in Theorem 1 and therefore worse compared to the FTA approach.

Algorithm 7 illustrates the cooperation of both approaches. Therein, we make use of the symmetrization function $s(\varphi_j)$ and the symmetrized set $S(eventset)$ as defined in Equation 5.17. The switching condition is defined in Line 13 and depends on the maximum deviation a node identified. In detail, if the maximum deviation exceeds $1/L$, then the R-RFA approach is chosen. Otherwise, if the deviation is less than $1/L$, then the FTA approach is chosen. The parameter L is named the *FTA convergence threshold*, because it depends on the worst case scenario where the FTA approach may never converge. In order to get the worst case scenario, we first define the term of a Basic Rest Circle (BRC) in an undirected communication graph $N = (V,E)$ which represents the communication network topology for a given system. In detail, the vertex set V contains all nodes of the system and the edge set E represents the topology of the system, that is, an edge $(u,v) \in E$ means that both p_u and p_v are in the transmission range of each other.

Definition 34 (Path set)**.** *For a given undirected communication graph $G = (V,E)$ and any two nodes $p_i, p_j \in V$ with $p_i \neq p_j$, $P(p_i, p_j)$ denotes the set of all possible paths from p_i to p_j within G. If there exists no such path, then $P(p_i, p_j)$ is empty.*

Definition 35 (Length of a path/circle)**.** *For a given path $p = \langle p_0, p_1, \ldots, p_k \rangle$, $l(p) = k$ denotes the length of path p. Similarly, for a given circle $c = \langle p_0, p_1, \ldots, p_k, p_0 \rangle$, $l(c) = k+1$ denotes the length of circle c.*

Definition 36 (Basic Rest Circle)**.** *A BRC of an undirected communication graph $G = (V,E)$ is a closed simple path $C = \langle p_0, p_2, \ldots, p_k, p_0 \rangle$ which starts and ends at the same node, but has no other repeated nodes and satisfies the following condition:*

1. $\forall 0 \leq i < j \leq k : \nexists p \in P(p_j, p_i) : l(p) < \min(j-i, k+1-(j-i))$

Condition 1 in Definition 36 ensures that the basic rest circle C passes only different broadcast domains. For instance, if two nodes $p_i, p_j \in C$ with $i+1 < j$ are in the transmission range of each other (*i.e.*, $(p_i, p_j) \in E$), then C is no basic rest circle. More informally, any two nodes $p_i, p_j \in C$ with $i+1 < j$ are not in the transmission range of each other. Note that from Definition 36 it follows that the smallest BRC C_{min} with $l(C_{min}) \neq 0$ equals a ring comprising three nodes[4] such that $l(C_{min}) = 3$. Smaller BRCs do not exist.

Definition 37 (Maximum Basic Rest Circle)**.** *The Maximum BRC C_{max} of an undirected communication graph $G = (V,E)$, is a BRC such that there exists no other BRC C_i in G with $l(C_i) > l(C_{max})$.*

Theorem 6. *Let C_{max} be the maximum BRC of the system. In the case at most f nodes are erroneous and $n \geq 5f+1$, Algorithm 7 may never converge if the convergence threshold $L \leq \max(l(C_{max}), 4)/2$.*

[4]Note that such a ring also corresponds to an all-to-all topology.

Proof. The proof consists of two parts. The first part shows that there exist configurations such that the FTA approach will never converge if $L \leq \max(l(C_{max}),2)/2$. The second part proofs that $L \leq \max(l(C_{max}),4)/2$ must hold to cover the general case.

The simplest network topology comprising n nodes with the maximum possible BRC of $l(C_{max}) = n$ is a simple bidirectional ring topology. In other words, C_{max} equals the complete communication topology such that $l(C_{max}) = \langle p_0, p_1, \ldots, p_{n-1}, p_0 \rangle$. Let the initial pulse state at $t = 0$ be $P_0 = \langle \varphi_0, \varphi_1, \ldots, \varphi_{n-1} \rangle = \langle 0, \frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n} \rangle$. Clearly, the successive pulse state P_1 after each node applied the FTA approach then equals the initial pulse state. Consequently, the pulse state will never change over time and the algorithm will never converge. This proof can be applied on any topology that contains a ring and thus has $l(C_{max}) \neq 0$.

The second part is based on the fact that the FTA approach only converges if the nodes are initially synchronized to some coarse precision. Therefore, the worst case precision where the nodes in an all-to-all topology converge equals exactly $1/2$. That is, any two nodes are no more than $1/2$ apart. This leads to the fact that $L > 4$ must hold in order to provide convergence of the FTA approach in any all-to-all topology. Consequently, if $L \leq \max(l(C_{max}),4)/2$, then the FTA approach may not converge in a connected communication topology. □

In order to incorporate the effect of inaccuracies and drift, we finally set the FTA convergence threshold to $L = \max(l(C_{max}),4)$. However, in the next section we will see that there exists an even more tight lower bound for L which was identified for ring topologies but also applies to all topologies which contain a maximum BRC C_{max} with $l(C_{max}) > 0$.

Algorithm 7: FTA-RFA: code for p_i, $0 \leq i < n$, $n \geq 5f+1$

1 **Init:** eventset $:= \emptyset$, $\Delta_i := 0$, $\varphi_i := 0$, $\text{offset}_i := \texttt{random}(r^{max}_{msd} - r^{min}_{msd}) + r^{min}_{msd}$
2 **upon event** $\varphi_i(t) = 1 - \text{offset}_i$ **do** `// preponed transmission`
3 **trigger** $\text{broadcast}_i(\varphi_i(t))$ `// broadcast current phase to all neighbors`
4 **upon event** $recv_i(\varphi_j)$ *from* p_j **do** `// received new sync-message`
5 eventset $:=$ eventset $\cup \{\varphi_i(t) + 1 - \varphi_j\}$
6 **upon event** $\varphi_i(t) = 1$ **do** `// threshold reached`
7 $\varphi_{last} := \delta_{last} := \Delta_i := 0$ `// clean up`
8 ftaset $:=$ eventset $\cup \{1\}$ `// copy set for FTA approach`
9 $\texttt{reduce}^f(\text{eventset})$ `// remove` f `highest and` f `lowest deviations`
10 $\text{dev}_{max} := \max(S(\text{eventset}))$
11 $\text{dev}_{min} := \min(S(\text{eventset}))$
12 $\text{dev} := \max(\text{dev}_{max} - \text{dev}_{min}, |\text{dev}_{max}|, |\text{dev}_{min}|)$
13 **if** $\text{dev} \geq 1/L$ **then** `// Execute E-RFA`
14 **for each** $\varphi_j \in$ eventset *in increasing order* **do**
15 **if** $\Delta_i + \varphi_j < 1$ **and** $\varphi_{last} + \delta_{last} < \varphi_j$ **then**
16 $\delta_{last} := \min(1, (\varphi_j + \Delta_i) \cdot \alpha) - (\varphi_j + \Delta_i)$
17 $\Delta_i := \Delta_i + \delta_{last}$
18 $\varphi_{last} := \varphi_j$
19 **else** `// Execute FTA`
20 $\Delta_i := -\texttt{avg}(S(\texttt{reduce}^f(\text{ftaset})))$
21 $\varphi_i(t) := \Delta_i$ `// Apply reachback response`
22 $\text{offset}_i := \texttt{random}(r^{max}_{msd} - r^{min}_{msd}) + r^{min}_{msd}$ `// Calculate firing offset`
23 eventset $:= \emptyset$

To sum up, whereas the R-RFA part of Algorithm 7 provides a robust convergence to a coarse synchronization precision in a fault-free system, the FTA part provides a fault-tolerant synchronization with a fine precision even in the presence of Byzantine faults.

Further Bounds for L

So far we have only devised a lower bound for L to guarantee that the FTA part in the FTA-RFA algorithm converges. However, this does not ensure that every fault-free system executing FTA-RFA converges to a synchronized pulse state. If anything, it is apparently impossible to prove convergence in general connected fault-free systems. On this account, we try to identify all possible stable fixpoint configurations which lead to an infeasible firing state and further try to find bounds to eliminate the existence of such system states. Therefore, based on the simulation results of ring topologies, we identified further bounds which are formally defined in Theorem 7 and Theorem 8. Note that both theorems of course also hold true in general connected network topologies that contain a maximum BRC C_{max} with $l(C_{max}) > 0$. In this case n must be replaced by $l(C_{max})$. Note that Theorem 8 was devised from empirical studies with ring topologies which have shown that the system sometimes entered an infeasible firing state.

Theorem 7. *Assume a fault-free ring system comprising n nodes and let $d_{max} = d + \varepsilon$ be the maximum possible message delay in a non-faulty communication network as stated in Definition 26. Algorithm 7 may never converge, if $L \geq \frac{T}{n \cdot d_{max}}$.*

Proof. The proof is based on the fact that the best real-time precision between two neighboring nodes achieved solely through the RFA part of Algorithm 7 is at least d_{max}. As a consequence, the best achievable real-time precision according to the E-RFA approach among all nodes is at least $(n-1) \cdot d_{max}$. Since in this case the FTA part of Algorithm 7 should be activated at all nodes in order to improve the synchronization precision, the inequation $1/L > (n-1) \cdot d_{max}/T$ must hold. □

Theorem 8. *Assume a fault-free ring network N comprising n nodes that suffers from no drift ($\rho = 0$). Let d_{max} be the maximum possible message delay of the system. Algorithm 7 may never converge if the following inequality is invalid:*

$$L \cdot (6 \cdot \gamma/\alpha - 2d_{max} \cdot (2\alpha^n \cdot (n+1) + 5\gamma)) > 3 \cdot (n-2) \cdot (\alpha^n - 1) + 3\gamma$$

Proof. The proof is based on the assumption that the initial configuration corresponds to a stable fixpoint which leads to an infeasible firing state. In detail, we assume that the nodes are placed exactly $1/(2L)$ apart such that they just execute the FTA part of Algorithm 7. However, the first and the last node still execute the RFA part due to their increased phase distance. Let $C(N, p_0, 1)$ be the corresponding firing state when the first node p_0 reaches the period end for the first time. In this case, we have $C(N, p_0, 1) = (\Delta_{p_0,1}, \varphi_{p_1,1}, \cdots, , \varphi_{p_{n-1},1})$ with $\varphi_{p_k,1} = 1 - k/(2L)$ for $1 \leq k < n-1$. Since the last node p_{n-1} always executes the RFA part and the fact that the RFA algorithm cannot synchronize two nodes better than the maximum message delay d_{max}, we have $\varphi_{n-1} = \varphi_{n-2} - d_{max}/T$. Based on this initial firing state, we now devise bounds for the parameters such that the phase difference between p_0 and p_{n-1} is getting smaller over time. Therefore, let $\Phi_\Pi = (n-2) \cdot 1/(2L) + d_{max}/T$ be the initial maximum phase difference.

Due to the existence of the message delay d_{max} and the fact that the nodes execute the FTA algorithm, all nodes p_k, $1 \leq k < n-1$, adjust their pulse clock backward by a phase of $2d_{max}/(3T)$. Note that this applies only once for p_1, because afterwards p_1 will recognize a phase difference greater than $1/L$ and consequently executes the RFA part. In contrast, p_{n-3} adjusts their pulse clock at most $(n-3)$ times backward by a phase of $2d_{max}/(3T)$. On this account, after p_0 reaches the period end the $(n-1)$th time, in the worst case we then have $C(N, p_0, n-2) = (\Delta_{p_0,n-1}, \varphi_{p_1,n-1}, \cdots, \varphi_{p_{n-1},n-1})$ with $\varphi_{p_{n-2},n-1} = \varphi_{p_{n-2},1} - (n-1) \cdot 2d_{max}/(3T))$ and again $\varphi_{p_{n-1},n-1} = \varphi_{p_{n-2},n-1} - d_{max}/T$. Note that in the same round, p_{n-2}

executes the RFA part and consequently, p_{n-2} recognizes a phase deviation between p_{n-1} and p_{n-3} that is greater than $1/L$.

As a result, the next time when p_0 reaches the period end, p_{n-2} executes the RFA part and we get $C(N, p_0, n-1) = (\Delta_{p_0,n}, \varphi_{p_1,n}, \cdots, \varphi_{p_{n-1},n})$ with $\varphi_{p_{n-2},n} = \varphi_{p_{n-2},n-1} + \Delta_{p_{n-2},n}$ where in the worst we get a smallest $\Delta_{p_{n-2},n}$ of $\Delta_{p_{n-2},n} \geq (1 - 1/(2L) - 2d_{max}/(3T) - \Delta_{p_{n-3},n-1} - d_{max}/T) \cdot \gamma$ and $\Delta_{p_{n-3},n} = \gamma/(1+\gamma)$. In order to ensure convergence, the inequality $1 - \varphi_{p_{n-1},n} + \Delta_{p_0,n} < 1 - \varphi_{p_{n-1},1} + \Delta_{p_0,1}$ must hold. This inequality is valid, if $\Delta_{p_{n-2},n} > (n-2) \cdot 2d_{max}/(3T) + \sum_{k=1}^{n} \Delta_{p_0,k}$ is true. Note that in the worst case we can assume that $\sum_{k=1}^{n} \Delta_{p_0,k} \leq (\Phi_\Pi + (n-2) \cdot 2d_{max}/(3T) + d_{max}/T) \cdot (\alpha^n - 1)$. Replacing the variables finally leads to the inequality as stated. □

5.3.2 Fault-tolerant Drift Compensation

The fault-tolerant variant of the previously presented drift calibration technique is based on the same idea like the FTA approach. That is, we exclude a set of highest and lowest values. However, since this algorithm requires the distinction of the different messages with respect to the different sender nodes, each synchronization message used for the drift calibration scheme must contain the unique identifier of the origin. In other words, this algorithm is *non-anonymous*. Unfortunately, this fact makes it impossible to tolerate f Byzantine nodes, if $n = 3f + 1$, because the calculation of the drift compensation is based on the individual history of the last N received messages for each neighboring node. In detail, the calculation as stated in Equation 5.10 and realized in Line 25 of Algorithm 8 depends on the content of two received messages from the same node. The fact that a faulty node may transmit with different ids in each round and the assumption that a faulty node cannot transmit more than one message per round (Definition 31) requires at least $n \geq 5f + 1$ nodes. For instance, assume the case that the oldest message of f different nodes are forged and additionally the latest message of f other different nodes are forged. Consequently, in the worst case we would have at most $2f$ incorrect drift calculations of neighboring nodes. By excluding the $2f$ highest and $2f$ lowest values, we can be sure that the average of the remaining drift values agrees to some extent with the average at the neighboring nodes. This proves that at least $n \geq 5f + 1$ nodes are required.

If the real world would behave exactly according to our system assumption, then everything would be alright. For instance, if a node receives less than $n - 2f$ messages, then we can assume that the faulty nodes performed a jamming attack such that the messages are transmitted exactly at the same time when a non-faulty node started the message transmission. In other words, the f faulty nodes corrupted the transmission of at most f non-faulty nodes resulting in the fact that a non-faulty node receives at least $n - 2f - 1$ messages. In this case, all received messages can be assumed to be correct.

In practice, the aforementioned situation is inappropriate, because messages may be temporarily lost due to environmental influences, or non-faulty nodes may run out of energy. Consequently, a node has to remove the $2f$ lowest and highest values irrespective of the overall number of received messages. In the case there are no remaining values after the removal process, a node does not change its drift rate. However, as already explained in the previous section, consider the case the f faulty nodes always jam f other non-faulty nodes which is valid due to our definition of a Byzantine node. Hence, a non-faulty node always receives at most $n - 2f$ messages and removes the $2f$ highest and $2f$ lowest values from $n - 2f$ corresponding drift values. In the case $n = 5f + 1$, every node will never have remaining values after the removal process and consequently will never adjust its drift rate. To overcome this situation, we require that $n \geq 7f + 1$ for the robust version of the drift calibration scheme.

Algorithm 8 illustrates this fault-tolerant version named Robust Fault-tolerant Drift Calibration (FT-DC)[5]. All variables used in the algorithm are described in short in Table 5.2. Similar to Algorithm 7, Algorithm 8 makes use of the $reduce(M)$ function for removing the lowest and highest value contained in the multiset M.

Table 5.2: Variables of the FT-DC algorithm.

$offset$	the actual phase offset of p_i for broadcasting the synchronization message
$h^i(t)$	the actual relative rate adjustment value of p_i at time t
$HC_i(t)$	the actual microtick of hardware clock HC_i of p_i at time t
$\Sigma_{s_i}(t)$	the periodically summarized slope of p_i at time t
$s_i(t,N_s)$	actual slope of p_i, calculated by linear regression over the last N_s periods
h_{tmp}	stores the relative rate adjustment value of the previous round
$\bar{h}$	the average relative rate adjustment value among all nodes
$eventset$	is a set that contains all event messagesall received during a complete round
$idset$	is a multiset that contains all received neighboring identifiers
$tmpidset$	is used to select a single message if more messages with same id exist
$rateset$	is a multiset that contains all received relative rate adjustment values
$slopeset$	is a multiset that contains all received averaged summarized slope values
$history[id]$	contains the timestamped history of received adjustment values for each id
N	the history length to be stored (same for all nodes)
$r_{msd}^{max}, r_{msd}^{min}$	the maximum and minimum message staggering delay (same for all nodes)

The proof of convergence of Algorithm 8 is skipped for sake of clarity. However, intuitively it is clear that the drift of all nodes smoothly converge to a common average drift, as long as the network is $(7f+1)$-connected. In short, this argument results from the fact that the lowest and highest existing drift adjustment value cannot become smaller and higher, respectively. The simulation results presented in Chapter 6 additionally emphasize the correctness of this approach. Note that the algorithm also implements the compensation of the common-mode drift in Line 29-31. The calculation of the slope variable $s_i(t,N_s)$ directly follows Equation 5.14 and is omitted in the algorithm in order to keep the algorithm small.

Note that Algorithm 7 and Algorithm 8 use the same messages and can be easily combined to provide both a drift correction and a state correction in order to precisely solve the pulse synchronization problem. For sake of simplification we assume that after a long enough time, the drift correction algorithm hardly affects the precision among the nodes with respect to their pulse clocks. In detail, the precision degradation resulting from the drift correction algorithm is assumed to be orders of magnitudes smaller than the achievable precision of the state correction algorithm. This is an important assumption, since otherwise, we would have to incorporate the precision degradation of the drift calibration algorithm into the worst case precision of the FTA algorithm at the pulse clock level.

5.4 Improvements in Single-hop Networks

So far, we have considered general network topologies that provide redundancy to some degree. If we restrict our attention to single-hop networks, then several additional concepts can be implemented. This results from the fact that the assumption of a single-hop topology is a powerful constraint which allows the development of efficient and deterministic algorithms even in the presence of Byzantine faults.

[5] Note that the index i at some variables is often omitted when it is clear from context that the variable corresponds to node p_i.

Algorithm 8: Robust FT-DC: code for p_i, $0 \leq i < n$, $n \geq 7f+1$

1 **Init:** eventset $=$ idset $= \emptyset$, $\varphi_i = h_{tmp} = 0$, offset $=$ `random`$(r^{max}_{msd} - r^{min}_{msd}) + r^{min}_{msd}$
2 **upon event** $\varphi_i(t) = 1 - \text{offset}$ **do** `// preponed transmission`
3 **trigger** broadcast$_i(id_i, HC_i(t), h^i(t), \Sigma_{s_i}(t))$ `// broadcast local timestamp`
4 **upon event** $recv_i(id_j, HC_j, h_j, \Sigma_{s_j})$ *from* p_j **do** `// received sync-message`
5 idset := idset $\cup \{id_j\}$
6 add $(id_j, HC_j, h_j, HC_i(t), \Sigma_{s_j})$ to eventset `//` $HC_i(t)$ `is the local timestamp`
7 **upon event** $\varphi_i(t) = 1$ **do** `// threshold reached`
8 offset := `random`$(r^{max}_{msd} - r^{min}_{msd}) + r^{min}_{msd}$ `// calculate new random offset`
9 slopeset := $\emptyset$ `// slopeset is a multiset`
`/* Initialize local variables */`
10 $\bar{h} := h^i(t)$
11 tmpidset := $\emptyset$
12 rateset := $\{h^i(t)\}$
`/* Update event history */`
13 **for each** $(id_j, HC_j, h_j, HC_i, \Sigma_{s_j}) \in$ eventset **do**
14 slopeset := slopeset $\cup \{\Sigma_{s_j}\}$
15 **if** $id_j \notin$ tmpidset **then** `// ensure a single event for each id`
16 tmpidset := tmpidset $\cup \{id_j\}$
17 history$[id_j]$:= history$[id_j] \cup (HC_i, HC_j, h_j)$
18 **for each** $id_j \in$ idset $\setminus$ tmpidset **do** `// indicate omissions`
19 history$[id_j]$:= history$[id_j] \cup (HC_i(t), \perp, \perp)$
20 eventset := $\emptyset$
`/* Calculate new rate adjustment value */`
21 **for each** $id_j \in$ idset *with* |history$[id_j]| = N$ **do**
22 Let $(HC_{i,N}, HC_{j,N}, h_{j,N}) \in$ history$[id_j]$ be the latest stored message
23 Let $(HC_{i,1}, HC_{j,1}, h_{j,1}) \in$ history$[id_j]$ be the N-th oldest stored message
24 **if** $HC_{j,1} \neq \perp$ **and** $HC_{j,N} \neq \perp$ **then**
25 $h_j := (HC_{i,N} - HC_{i,1}) \cdot (1 + h_{j,N}) / (HC_{j,N} - HC_{j,1}) - 1$
26 rateset := rateset $\cup \{h_j\}$ `// note that rateset is a multiset`
27 `reduce`^{2f}(rateset)
28 **if** |rateset| > 0 **then** $\bar{h} := (\sum_{\forall h_j \in \text{rateset}} h_j) / |\text{rateset}|$
29 slopeset := $\{\Sigma_{s_i}(t)\} \cup$ `reduce`^{f}(slopeset)
30 $\Sigma_{s_i}(t) := avg(\text{slopeset}) + s_i(t, N_s)$ `// common-mode drift compensation`
31 $h^i(t) := \bar{h} - s_i(t, N_s) - \Sigma_{s_i}/N_s$ `// adjust drift rate of pulse clock`
32 **continuously do** `// Ongoing cleanup`
33 if p_i remains in the unsync-state, then $\Sigma_{s_i}(t) := 0$ and $s_i(t, N_s) := 0$
34 **for each** $id_j \in$ idset **do**
35 delete history$[id_j]$, if no message was received during the last N periods
36 keep only the latest N entries in history$[id_j]$ and delete the rest

5.4.1 Clique Discovery

The main drawback of the FTA-RFA approach (Algorithm 7) is the long convergence time in single-hop topologies in the case all but one are initially synchronized. For instance, consider the fault-free case of $n > 2$ nodes, where all nodes except some p_i are initially synchronized. Algorithm 7 ensures that the nodes keep synchronized and that p_i is attracted to the other nodes. The convergence time then equals the case of two nodes and is already analyzed in Theorem 3.

Based on the aforementioned example, we devise a simple extension for the FTA-RFA algorithm such that a node first performs a clique discovery. If a clique is detected, then a node directly adjusts its phase to the clique by executing the FTA algorithm solely on the information received from the nodes within this clique. The synchronization information of the other nodes is discarded. Consequently, the convergence time of the mentioned example is reduced to exactly 1 round, independently of the initial phase difference between p_i and the other synchronized nodes. In practice, the additional clique discovery approach effectively speeds up the convergence time, because completely equidistant distributed initial phases hardly occur in reality and even randomly initiated networks often contain cliques.

Another advantage of the clique discovery approach is the fact that it can be used for the *bootstrapping phase*. For instance, in the case a new node wants to join an already synchronized network, then the bootstrapping phase ensures that the other nodes are not desynchronized. This is done by not actively participating in the synchronization process. In detail, the new node only listens to the medium for one complete period without transmitting any messages and then adjusts its phase directly to the discovered clique by executing the FTA algorithm based on the information of the nodes within this clique.

Algorithm 9 extends the basic FTA-RFA Algorithm 7 by the use of a clique discovery approach. Therein, Line 8 executes the $CD(eventset)$ function which is described in more detail in Algorithm 10. An important detail within the clique discovery approach is that the passed multiset for the clique discovery function must contain the nodes' own phase, that is, it includes always the phase $\varphi = 1$. In short, the function returns a subset $clique \subseteq eventset$ that corresponds to a discovered clique. In the case no clique was identified, then the function simply returns the original passed multiset $eventset \cup \{1\}$. This fact is used in Line 9 in order to set the boolean variable *discovered*. In the case a clique was discovered, then the algorithm directly performs the FTA approach on the identified clique. However, if no clique was discovered and all nodes are very close together, then FTA is again executed instead of the original RFA approach. Fortunately, the FTA approach can be directly applied on the returned clique as shown in Line 21, because this set already contains the nodes' own phase as required by the FTA approach. For this, the $avg(M)$ function applied on a multiset M is defined as $avg(M) := \sum_{\forall \varphi \in M} \varphi / |M|$ and returns 0 if M is empty.

Algorithm 10 presents the pseudocode of the clique discovery approach for $n > 3f$ in the presence of at most f Byzantine nodes. The code makes use of a trivial distance function $d(\varphi_A, \varphi_B)$ which calculates the normalized absolute phase difference between both φ_A and φ_B. The formal definition is given in Equation 5.18.

$$d(\varphi_A, \varphi_B) = \begin{cases} |\varphi_A - \varphi_B| & \text{if } |\varphi_A - \varphi_B| < \frac{1}{2} \\ 1 - |\varphi_A - \varphi_B| & \text{else} \end{cases} \tag{5.18}$$

The $vic(S, \varphi_r, k)$ function, formally defined in Equation 5.19, returns a subset $C \subseteq S$ of phase values which are in the k-vicinity of a reference phase φ_r. In Algorithm 10, the parameter φ_r sometimes corresponds to an element or a multiset of phase values. However, in the case of a multiset, all elements in φ_r have the same phase value and, therefore, this does not matter. In the case the multiset S or φ_r is empty, the vicinity function returns an empty set too.

$$vic(S, \varphi_r, k) = \{\varphi \in S \mid d(\varphi_r, \varphi) \leq k\} \tag{5.19}$$

In each round, the algorithm simply tries to find the two greatest cliques which are at least $1/k$ apart. However, it must be ensured that all nodes consistently discover the same two cliques. On this account, a node only stores the greatest clique, if it contains at least $2f$ more phase values than the second greatest clique. This ensures that Byzantine nodes cannot introduce inconsistencies by sending different values to the distinct nodes. Furthermore, if

Algorithm 9: FTA-RFA with CD: code for p_i, $0 \leq i < n$, $n \geq 5f+1$

Init: eventset $:= \emptyset$, $\Delta_i := 0$, $\varphi_i := 0$, offset$_i$:= random$(r_{msd}^{max} - r_{msd}^{min}) + r_{msd}^{min}$

upon event $\varphi_i(t) = 1 - \text{offset}_i$ **do** // preponed transmission
  trigger broadcast$_i(\varphi_i(t))$ // broadcast current phase to all neighbors

upon event $recv_i(\varphi_j)$ *from* p_j **do** // received new sync-message
  eventset := eventset $\cup \{\varphi_i(t) + 1 - \varphi_j\}$

upon event $\varphi_i(t) = 1$ **do** // threshold reached
  $\varphi_{last} := \delta_{last} := \Delta_i := 0$ // clean up
  clique := CD(eventset $\cup \{1\}$) // perform clique discovery
  if |clique| = |eventset| + 1 **then** discovered := false **else** discovered := true
  reduce^f(eventset) // remove f highest and f lowest deviations
  $\text{dev}_{max} := \max(S(\text{eventset}))$
  $\text{dev}_{min} := \min(S(\text{eventset}))$
  $\text{dev} := \max(\text{dev}_{max} - \text{dev}_{min}, |\text{dev}_{max}|, |\text{dev}_{min}|)$
  if discovered = false **and** dev $\geq \frac{1}{L}$ **then** // Execute E-RFA
    for each $\varphi_j \in$ eventset *in increasing order* **do**
      if $\Delta_i + \varphi_j < 1$ **and** $\varphi_{last} + \delta_{last} < \varphi_j$ **then**
        $\delta_{last} := \min(1, (\varphi_j + \Delta_i) \cdot \alpha) - (\varphi_j + \Delta_i)$
        $\Delta_i := \Delta_i + \delta_{last}$
        $\varphi_{last} := \varphi_j$
  else // Execute FTA
    $\Delta_i := -\text{avg}(S(\text{reduce}^f(\text{clique})))$
  $\varphi_i(t) := \Delta_i$ // Apply reachback response
  offset$_i$:= random$(r_{msd}^{max} - r_{msd}^{min}) + r_{msd}^{min}$ // Calculate firing offset
  eventset := $\emptyset$

the greatest clique contains more than $2f$ phase values, then the algorithm refines the clique discovery by executing a further round with a smaller vicinity size which is the half of the previous round. This is continuously done until the smallest vicinity size of $1/w$ is reached or the greatest clique contains $\leq 2f$ phase values. For this, the *precision window* w contains the estimation of the achievable worst case precision. In other words, the precision window defines the lower bound of the clique discovery resolution such that a highly probable consistent clique discovery among all nodes can just be guaranteed. The algorithm finally returns the subset $C \subseteq S$ which contains the unique greatest discovered clique with the smallest possible clique discovery resolution. In the case no unique clique was discovered, the algorithm simply returns the complete original set S.

5.4.2 Message Delay Estimation

Message Delay Estimation (MDE) is usually done by implementing a two-way message exchange mechanism as used in MSTSP (Section 4.1.3), SGS (Section 4.1.4), or TinySeR-Sync (Section 4.1.5). However, a two-way message exchange approach would result in an increased message complexity and further ruins the simplicity and advantages of one-way dissemination. In contrast, our approach preserves one-way dissemination by exploiting the homogeneity within a WSN. That is, all nodes are based on the same hardware and usually execute the same software. Note that this assumption only applies to WSNs and is inappropriate in the case of general wireless or wired distributed systems.

The basics behind the algorithm is that the message delay estimation does not rely on the information of a single node. Contrary, the calculation is distributed over the complete

Algorithm 10: Clique Discovery (CD)

```
1  procedure CD(S)
2      k := 4, C := S
3      repeat
4          k := min(⌊1/w⌋, 2k)
           /* Find cliques with greatest size */
5          S_A := {φ_A ∈ S | |vic(S, φ_A, 1/k)| = max_{∀φ∈S} |vic(S, φ, 1/k)|}
6          if |S_A| = 1 then
               /* Find cliques with second greatest size */
7              S_B := {φ_B ∈ S | |vic(S, φ_A, 1/k)| = max_{∀φ∈S} |vic(S, φ, 1/k)| ∧ d(S_A, φ_B) > 1/k}
               /* Check if greatest clique is unique */
8              if |vic(S, S_A, 1/k)| − |vic(S, S_B, 1/k)| > 2f then
9                  C := vic(S, S_A, 1/k)          // Store discovered unique clique
10     until k = ⌊1/w⌋ or |vic(S, S_A, 1/k)| ≤ 2f
11     return C
```

network such that a node estimates the delay by averaging the estimates over all neighboring nodes.

Algorithm 11 presents the extended version of Algorithm 7 for single-hop networks. The new code lines are highlighted within the code and realize the message delay estimation. The main advantage of our distributed message delay estimation is that it is based on one-way dissemination and adds only one new value to the synchronization message. Basically, in Line 28 the algorithm calculates the average deviation over all received synchronization messages after removing the f highest and f lowest values. This deviation then is broadcasted within the synchronization message as shown in Line 3. During each period, a node gathers all received deviations in the *devset* variable. This multiset is cleared at the end of each period in Line 29. Since the calculation of the overall average message delay among all average deviations of the neighboring nodes also includes the calculated average deviation of the own node of the previous period, the *devset* variable initially contains the calculated average deviation of the last period.

The final message delay estimation is done in Line 16 by simply averaging all received neighboring message delay estimates. However, it should be noted that a fraction of the resulting message delay estimate may not belong to the real message delay and thus must be removed. Otherwise, especially within multi-hop topologies, simulation results have shown that the phase deviation among the nodes will periodically increase and decrease over time. To avoid this swinging behavior, we decided to remove half the maximum deviation among the neighboring nodes (excluding the own node), because this deviation is a good measure for the real actual precision that does not include the imprecision due to the message delay.

The estimation of the average message delay is only performed, if the corresponding node remains in the sync-state, *i.e.*, the deviation to all neighboring nodes is smaller than the predefined synchronization window w. Otherwise, the calculation would make no sense. Therefore, Line 15 clears the *devset* variable, if the node recognizes a deviation which is greater than w. Thus, if a node recognizes that the maximum deviation with respect to all neighboring nodes is less than w for several consecutive synchronization rounds, then it determines itself to be synchronized. In order to minimize the influence of outliers and keep the average message delay *mde* nearly constant and stable, Line 17 smoothly adapts the new mde' value by the use of the smoothing factor σ_{mde}. During all our simulations, we set σ_{mde} to the value of $1/2$. Line 9 finally makes use of the actual estimated average message delay.

In detail, the clique discovery algorithm is executed on the gathered events, shifted by the estimated average message delay.

Algorithm 11: FTA-RFA with CD and MDE: code for p_i, $0 \leq i < n$, $n \geq 5f+1$

1 **Init:** eventset $:=$ devset $:= \emptyset$, $\Delta_i := \varphi_i := \text{dev}_{avg} := \text{mde} := 0$,
  $\text{offset}_i = \texttt{random}(r_{msd}^{max} - r_{msd}^{min}) + r_{msd}^{min}$
2 **upon event** $\varphi_i(t) = 1 - \text{offset}_i$ **do** // preponed transmission
3 **trigger** broadcast$_i(\varphi_i(t), \text{dev}_{avg})$
4 **upon event** $recv_i(\varphi_j, \text{dev}_j)$ *from* p_j **do** // received new sync-message
5 eventset $:=$ eventset $\cup \{\varphi_i(t) + 1 - \varphi_j\}$
6 devset $:=$ devset $\cup \{\text{dev}_j\}$
7 **upon event** $\varphi_i(t) = 1$ **do** // threshold reached
8 $\varphi_{last} := \delta_{last} := \Delta_i := 0$ // clean up
9 clique $:= \texttt{CD}(\{\varphi - \text{mde} \mid \varphi \in \text{eventset}\} \cup \{1\})$ // clique discovery
10 **if** $|\text{clique}| = |\text{eventset}| + 1$ **then** discovered $:=$ false **else** discovered $:=$ true
11 $\texttt{reduce}^f(\text{eventset})$ // remove f highest and f lowest deviations
12 $\text{dev}_{max} := \max(S(\text{eventset}))$
13 $\text{dev}_{min} := \min(S(\text{eventset}))$
14 $\text{dev} := \max(\text{dev}_{max} - \text{dev}_{min}, |\text{dev}_{max}|, |\text{dev}_{min}|)$
15 **if** dev $> w$ **then** devset $:= \emptyset$
16 $\text{mde}' := \max(0, \texttt{avg}(\texttt{reduce}^f(\text{devset})) - (\text{dev}_{max} - \text{dev}_{min})/2)$
17 $\text{mde} := \text{mde} + (\text{mde}' - \text{mde}) \cdot \sigma_{mde}$
18 **if** discovered $=$ false **and** dev $\geq \frac{1}{L}$ **then** // Execute E-RFA
19 **for each** $\varphi_j \in$ eventset *in increasing order* **do**
20 **if** $\Delta_i + \varphi_j < 1$ **and** $\varphi_{last} + \delta_{last} < \varphi_j$ **then**
21 $\delta_{last} := \min(1, (\varphi_j + \Delta_i) \cdot \alpha) - (\varphi_j + \Delta_i)$
22 $\Delta_i := \Delta_i + \delta_{last}$
23 $\varphi_{last} := \varphi_j$
24 **else** // Execute FTA
25 $\Delta_i := -\texttt{avg}(S(\texttt{reduce}^f(\text{clique})))$
26 $\varphi_i(t) := \Delta_i$ // Apply reachback response
27 $\text{offset}_i := \texttt{random}(r_{msd}^{max} - r_{msd}^{min}) + r_{msd}^{min}$ // Calculate firing offset
28 $\text{dev}_{avg} := \texttt{avg}(S(\text{eventset}))$
29 devset $:= \{\text{dev}_{avg}\}$
30 eventset $:= \emptyset$

5.4.3 Automatic Passive Mode for Single-hop Networks

The efficient use of energy is an important keystone in WSNs but also in nearly all battery-powered devices. Since message transmission is one of the major energy consumers in wireless nodes, the design of an energy-efficient communication protocol often focuses on the transmission of small-sized messages. In addition to that, we investigated the fact that many nodes unnecessarily participate in the synchronization process. For instance, assume a single-hop topology comprising hundreds of nodes where at most f nodes are permanently Byzantine faulty. Instead that a node always receives a synchronization message from each neighboring node in each round, it is sufficient enough that a node receives messages from at least $\geq 3f+1$ distinct nodes (including itself). In other words, the main approach is to let all

but at least $3f+1$ nodes enter a passive state. The remaining nodes keep in an active state. If a node is in the active state, then it behaves like before and broadcasts a synchronization message in each round. Contrary, if a node is in the passive state, then it only listens to the medium for broadcasted synchronization messages but never broadcasts a message itself. Consequently, the active nodes will be the first that may run out of energy at some time t. However, in the simple approach without containing passive nodes, all nodes may run out of energy very frequently. A further advantage of the passive nodes approach is that if active nodes run out of energy, the passive nodes recognize this dilemma and can re-enter the active state such that again at least $3f+1$ nodes are active. Depending on the energy ratio $r = E_t/E_l$ between the energy consumed for transmitting and only listening on the medium, then the network lifetime can be extended by a factor of at most $\min(n/(3f+1), 1/r)$. In other words, we exploit the redundancy of the number of nodes in order to increase robustness and network lifetime.

The Automatic Passive Mode (APM) approach for single-hop networks is a simple algorithm and consists of two parts. The first part considers the recognition if too many nodes are active. If so, then the second part determines if a node has to enter the passive state or can remain in the active state. Algorithm 12 demonstrates the approach as an extension to Algorithm 11 and makes use of the APM-procedure which is coded in Algorithm 13. The new code lines are again highlighted.

In more detail, we introduce a new multiset variable named *recvset*. This variable gathers the local reception phase at the receiving node for each received synchronization message during one synchronization period. This procedure is implemented in Line 6 in Algorithm 12. A further new boolean variable is termed *active* and represents one of the two states a node persists in, namely active state ($active = true$) or passive state ($active = false$). In order to use this boolean variable within mathematical calculations, we use the binary value of 1 to represent the $true$ state and 0 to represent the $false$ state. Initially, *active* is set to 1. However, if *active* was set to 0 in the previous round, then some calculations must be modified, because then the own node must not be incorporated in the calculation process. Examples of the modified lines are Line 11, and Line 12 of Algorithm 12. For instance, if *active* equals 0, then the clique detection procedure must not incorporate the own node (Line 11). Consequently, the state of the *active* variable must be incorporated for setting the *discovery* variable in Line 12. Based on the content of the *recvphase* set, the actual state of the *active* variable and the last offset-value, Line 29 then sets the *active* variable for the next round. However, in the case a node recognizes that at least some other node deviates more than the allowed worst case precision as defined via the precision window w, then the node always actively participates in the synchronization process. This fact is implemented in Line 30.

Algorithm 13 presents the code for determining the new state of the *active* variable. Therein, Line 5 returns 1, if a node recognizes that too many nodes are active. Similarly, Line 6 ensures that at least $\max(5f, 3)$ nodes are always active. In other words, the procedure implements some kind of hysteresis such that after some time, the nodes stably stay within the active or passive state, irrespective of the behavior of the Byzantine faulty nodes, until several active nodes run out of energy or other environmental effects change the structure of the network. The *diff* variable in Line 4 contains the absolute difference between the number of synchronization messages that were received before the own message broadcast and the number of synchronization messages that were received after the own message broadcast.

Since the lower threshold in Line 6 directly follows from the condition that at least $\max(5f, 3)$ nodes must be active for Algorithm 12 in order to overcome the Byzantine nodes, we concentrate on the upper threshold in Line 5. Note that in the case $f = 0$, we require that

at least 4 nodes participate in the synchronization process in order to provide some synchronization precision. If a higher synchronization precision is required, then one may increase both threshold requirements by some $k > 0$. Clearly, if we would simply implement the condition $|recvphase| \geq \max(8f,5)+2$ in Line 5 without the *diff* variable, then all nodes simultaneously may enter the passive state and in the next round re-enter the active state in a periodic manner. Note that the reception phases are stored in the *recvphase* variable and is the most consistent information a node gathers from the other nodes. Consequently, this information is used for electing a limited number of nodes to become passive. On this account, we decided that a node is only allowed to enter the passive state, if the phase offset of the last broadcast was in the middle with respect to all other nodes according to the reception phases of the received synchronization messages. In the perfect case, a node recognizes itself to be in the middle, if *diff* equals 0. Unfortunately, this leads to the fact that in each round at least 1 node or at most $f+2$ nodes become passive and consequently requires a long time until the upper threshold is underrun in the case n is very high. Therefore, we relaxed the condition of $diff = 0$ to $diff \leq k$ for some $k > 0$. For this, Lemma 7 proofs the worst case upper and lower bound of the number of nodes that may become passive at the end of the current round.

Lemma 7. *For a given single-hop network comprising $n \geq 8f+3$ active nodes at the beginning of some synchronization round where at most f nodes are permanently Byzantine faulty, Algorithm 13 with the condition $diff \leq k$ for some $k > 0$ instead of $|recvphase| \geq diff + \max(8f,5)+2$ in Line 5 ensures that at least $\max(0,k-f)$ nodes and at most $k+f+2$ nodes can become passive at the end of the current round.*

Lemma 7 applied on the original Algorithm with condition $diff \leq |recvphase| - 8f - 2$ for $f > 0$ in Line 5 then directly leads to Corollary 2.

Corollary 2. *For a given single-hop network comprising $n \geq 8f+3$ active nodes at the beginning of some synchronization round where at most f nodes are permanently Byzantine faulty, Algorithm 12 ensures that at least $\max(0,n-9f-3)$ nodes and at most $n-7f-1$ nodes become passive at the end of the current round.*

Proof. Follows directly from Lemma 7 with $k = (n-1)-8f-2 = n-8f-3$. □

From Corollary 2 we can deduce that, if the nodes are synchronized, then the Byzantine nodes can keep the number of active nodes n' within the interval $7f+1 \leq n' \leq 9f+3$. This fact is finally summarized in Theorem 9.

Theorem 9. *If the network of a coherent system comprises $n \geq 8f+3$ synchronized active nodes at the beginning of some round, then Algorithm 12 ensures that at least $7f+2$ and at most $9f+4$ nodes are active in the following rounds.*

It should be noted that the APM approach as stated in Algorithm 13 is designed for the FTA-RFA and not for the drift compensation approach of Algorithm 8. Fortunately, if Algorithm 8 is used in combination with Algorithm 12, then the only difference according to Algorithm 8 is that Line 11 must be replaced by $rateset := \emptyset$. Additionally, Algorithm 13 must be modified such that Line 5 contains the condition $|recvphase| \geq diff + \max(10f,5)+2$ and Line 6 contains the condition $|recvphase| < \max(7f,3)$. That is, both the upper and lower threshold are increased by $2f$. This comes from the fact that, in contrast to Algorithm 12 which requires $n \geq 5f+1$ nodes, Algorithm 8 requires $n \geq 7f+1$ nodes. In this case, Theorem 9 must be modified such that at least $9f+2$ and at most $11f+1$ nodes are active for $n \geq 10f+3$.

Algorithm 12: FTA-RFA with CD, MDE, and APM : code for p_i, $0 \leq i < n$, $n \geq 5f+1$

1 **Init:** eventset := devset := recvphase := $\emptyset$, $\Delta_i := \varphi_i := \mathsf{dev}_{avg} := \mathsf{mde} := 0$, active := 1, $\mathsf{offset}_i := \mathtt{random}(r_{msd}^{max} - r_{msd}^{min}) + r_{msd}^{min}$

2 **upon event** $\varphi_i(t) = 1 - \mathsf{offset}_i$ **and** active $= 1$ **do** // preponed transmission

3 **trigger** $\mathsf{broadcast}_i(\varphi_i(t), \mathsf{dev}_{avg})$

4 **upon event** $recv_i(\varphi_j, \mathsf{dev}_j)$ *from* p_j **do** // received new sync-message

5 $\mathsf{eventset} := \mathsf{eventset} \cup \{\varphi_i(t) + 1 - \varphi_j\}$

6 $\mathsf{recvphase} := \mathsf{recvphase} \cup \{\varphi_i(t)\}$

7 $\mathsf{devset} := \mathsf{devset} \cup \{\mathsf{dev}_j\}$

8 **upon event** $\varphi_i(t) = 1$ **do** // threshold reached

9 $\varphi_{last} := \delta_{last} := \Delta_i := 0$ // clean up

10 $\mathsf{clique} := \mathtt{CD}(\{\varphi - \mathsf{mde} \mid \varphi \in \mathsf{eventset}\} \cup \{1\})$

11 **if** active $= 0$ **then** $\mathsf{clique} := \mathtt{CD}(\{\varphi - \mathsf{mde} \mid \varphi \in \mathsf{eventset}\})$

12 **if** $|\mathsf{clique}| = |\mathsf{eventset}| +$ active **then** discovered := false **else** discovered := true

13 $\mathtt{reduce}^f(\mathsf{eventset})$ // remove f highest and f lowest deviations

14 $\mathsf{dev}_{max} := \max(S(\mathsf{eventset}))$

15 $\mathsf{dev}_{min} := \min(S(\mathsf{eventset}))$

16 $\mathsf{dev} := \max(\mathsf{dev}_{max} - \mathsf{dev}_{min}, |\mathsf{dev}_{max}|, |\mathsf{dev}_{min}|)$

17 **if** $\mathsf{dev} > w$ **then** devset := $\emptyset$

18 $\mathsf{mde}' := \max(0, \mathtt{avg}(\mathtt{reduce}^f(\mathsf{devset})) - (\mathsf{dev}_{max} - \mathsf{dev}_{min})/2)$

19 $\mathsf{mde} := \mathsf{mde} + (\mathsf{mde}' - \mathsf{mde}) \cdot \sigma_{mde}$

20 **if** discovered = false **and** $\mathsf{dev} \geq \frac{1}{L}$ **then** // Execute E-RFA

21 **for each** $\varphi_j \in \mathsf{eventset}$ *in increasing order* **do**

22 **if** $\Delta_i + \varphi_j < 1$ **and** $\varphi_{last} + \delta_{last} < \varphi_j$ **then**

23 $\delta_{last} := \min(1, (\varphi_j + \Delta_i) \cdot \alpha) - (\varphi_j + \Delta_i)$

24 $\Delta_i := \Delta_i + \delta_{last}$

25 $\varphi_{last} := \varphi_j$

26 **else** // Execute FTA

27 $\Delta_i := -\mathtt{avg}(S(\mathtt{reduce}^f(\mathsf{clique})))$

28 $\varphi_i(t) := \Delta_i$ // Apply reachback response

29 $\mathsf{active} := \mathtt{APM}(\mathsf{active}, \mathsf{recvphase}, \mathsf{offset}_i)$ // Check for passive mode

30 **if** $\mathsf{dev} > w$ **then** active := 1

31 $\mathsf{offset}_i := \mathtt{random}(r_{msd}^{max} - r_{msd}^{min}) + r_{msd}^{min}$ // Calculate firing offset

$\mathsf{dev}_{avg} := \mathtt{avg}(S(\mathsf{eventset}))$

32 $\mathsf{devset} := \{\mathsf{dev}_{avg}\}$

33 eventset := $\mathsf{recvphase} := \emptyset$

APM in Multi-hop Networks

Unfortunately, multi-hop networks do not offer the same elegance like single-hop networks. For instance, in single-hop networks the single variable n which represents the number of nodes in the system is enough to determine the complete network topology. In contrast, in multi-hop networks this variable useless without additional information about the connectivity in order to make points about the network topology. However, the process of gathering information which can be used to make assumptions about the connectivity is not a trivial task and usually goes along with an increased message complexity and maybe invalidates the advantage of our one-way dissemination approach. Furthermore, the presence of up to f Byzantine faulty nodes makes it worse, because it has to be ensured that the network keeps $(5f+1)$-connected or even $(7f+1)$-connected if the drift compensation approach is used.

Algorithm 13: Automatic Passive Mode (APM)

```
1 procedure APM(active, recvphase, offset)
2     prior := {φ ∈ recvphase | φ + offset < 1}
3     posterior := {φ ∈ recvphase | φ + offset ≥ 1}
4     diff := ||prior| − |posterior||
5     if |recvphase| ≥ diff + max(8f, 5) + 2 then return 0
6     if |recvphase| < max(5f, 3) then return 1 else return active
```

In [JL09], Jang *et al*. state that this is a NP-hard problem. On this account, we do not make attempts to implement APM in multi-hop networks.

There exist a lot of work in the scientific literature which concerns this problem. In detail, nodes have to determine itself to become passive in order to reduce the network complexity to be minimal k-connected. Jang *et al*. give a short overview of related work in [JL09] and also present the Decreasing Coverage (DECC) and the Distributed DECC (DECCdist) algorithm for an efficient reduction of the network connectivity to become approximately minimal k-connected. Unfortunately, all proposed algorithms make use of unique identifiers and therefore do not fit into our concept of our anonymous FTA-RFA algorithm. However, in the case the non-anonymous drift compensation algorithm is used in parallel to the FTA-RFA approach, then such a k-coverage reduction algorithm makes sense. However, the implementation of such an approach goes beyond the scope of this thesis and therefore is no treated herein.

5.4.4 Internal Desynchronization

The principle of clock desynchronization was already discussed in Section 2.2.4. With respect to our FTA-RFA algorithm, we adapted this approach for the message staggering delay in order to provide a desynchronized preponed message transmission. This reduces the number of message collisions compared to the simple random offset approach.

Algorithm 14 presents the code of the extended FTA-RFA approach from Algorithm 12 in combination with the desynchronization scheme. The new or modified code lines are again highlighted. Since the offset desynchronization concerns only the adjustment of the *offset* variable, Algorithm 14 is mostly similar to Algorithm 12 except that the new offset is now set by the DESYNC procedure (Algorithm 15) in Line 31. A negative return value of the DESYNC procedure means that there is no more bandwidth available for the exchange of synchronization messages, because the bandwidth between the two predefined offset bounds r_{msd}^{min} and r_{msd}^{max} is completely occupied by the other active nodes. If so, then the only opportunity is to become passive and try to synchronize to the active nodes (Line 32). It should be noted that a careful parametrization of all variables should avoid or even minimize the probability of the occurrence of this case. For instance, the available bandwidth between r_{msd}^{min} and r_{msd}^{max} has to be large enough to overcome this problem and further has to be small enough to provide a small duty-cycle of the listen/sleep schedule in order to reduce the consumption of energy.

Algorithm 15 presents the insights of the desynchronization scheme. In order to reduce the number of code lines, we make use of two new functions, namely the $prev(S, \varphi_r))$ and the $next(S, \varphi))$ function. As the name implies, the former function returns the nearest phase value $\varphi_{prev} \in S$ with respect to φ_r with $\varphi_{prev} < \varphi_r$. Similarly, the latter function returns the nearest phase value $\varphi_{next} \in S$ with respect to φ_r with $\varphi_{next} > \varphi_r$. Equation 5.20 and Equation 5.21 define both functions in a formal way and make use of the distance function as previously presented in Equation 5.18. Since both functions may return a multiset that

contains the same value several times, we assume that in this case the multiset corresponds to a single element of the set. In the case the function returns an emptyset, we assume that the prev-function and the next-function return $-\infty$ and $+\infty$, respectively. This behavior is finally expressed in Equation 5.22 and Equation 5.23.

$$f_{prev}(S,\varphi_r) = \{\varphi_{prev} \in S \mid \varphi_{prev} < \varphi_r \ \wedge \ \min(d(\varphi_{prev},\varphi_r))\} \tag{5.20}$$

$$f_{next}(S,\varphi_r) = \{\varphi_{next} \in S \mid \varphi_{next} > \varphi_r \ \wedge \ \min(d(\varphi_{next},\varphi_r))\} \tag{5.21}$$

$$prev(S,\varphi_r) = \begin{cases} \varphi \in f_{prev}(S,\varphi_r) & \text{if } f_{prev}(S,\varphi_r) \neq \emptyset \\ -\infty & \text{else} \end{cases} \tag{5.22}$$

$$next(S,\varphi_r) = \begin{cases} \varphi \in f_{next}(S,\varphi_r) & \text{if } f_{next}(S,\varphi_r) \neq \emptyset \\ +\infty & \text{else} \end{cases} \tag{5.23}$$

The DESYNC procedure requires four parameters. The first one is a multiset variable termed *recvphase*. It contains the measured phase of all received synchronization messages during one period at the time the message was received. The *offset* parameter is the offset a node used for the preponed broadcast of its synchronization message during the previous round. The *mde* parameter is the estimated message delay. In the case the average message delay is high, it is very important to incorporate this delay in the desynchronization process. Otherwise the nodes will never correctly desynchronize. The last two parameters correspond to the allowed lower and upper bound for the offset variable.

The DESYNC procedure works in a simple manner: Initially, every node calculates and transmits with a random offset. This fact is implemented in the code that invokes the DESYNC procedure. A node is required to gather the offset values of all neighboring nodes by measuring the actual phase at the time a synchronization message was received. Therefore, the *recvphase* variable contains these values for one round. The procedure then simply looks in the *recvphase* variable for the nearest previous and next phase value in accordance with the own offset and the actual estimated message delay. Next, the own offset is adapted to the middle of both values. However, if the own transmission of the synchronization message was omitted due to a failed clear channel assessment, then the actual selected bandwidth is highly probably occupied by another node and thus the procedure tries to jump to a different offset time where enough bandwidth for the message transmission is available. Note that in the absence of faulty nodes, Line 5 of Algorithm 15 could be replaced by a check if the time difference between the previous and the next phase value with respect to the own offset underruns the *bandwidth limit* of $2B$. However, a single Byzantine node may send different synchronization messages to all other nodes such that each correct nodes receives an offset event which is in the vicinity of the own offset event and consequently may underrun the aforementioned bandwidth limit. On this account, a bandwidth limit check cannot be used in the presence of Byzantine faults. Instead, the result of the Clear Channel Assessment (CCA) of the last message transmission must be checked.

The bandwidth limit B defines the minimum distance between two transmissions which is required in order to avoid message collisions and transmission overlapping. The parameter B depends on the transmission delay $d(m)$ of a message m and the worst case synchronization precision Π. Assuming a bounded transmission delay of $d(m) = [d, d+\varepsilon]$ (if m is eventually delivered), then B must be at least

$$B \geq d + 2\varepsilon + \Pi.$$

If several possible offset values were found (*i.e.*, $fallowset \neq \emptyset$), then the procedure randomly elects one out of all values by executing the *rand* function on the set (Line 11). Otherwise, the procedure returns a negative value which has to be verified by the caller.

Algorithm 14: FTA-RFA with CD, MDE, APM, and DESYNC : code for p_i, $0 \leq i < n$

1 **Init:** eventset := devset := recvphase := $\emptyset$, $\Delta_i := \varphi_i := \mathsf{dev}_{avg} := \mathsf{mde} := 0$, active := 1, $\mathsf{offset}_i := \mathtt{random}(r^{max}_{msd} - r^{min}_{msd}) + r^{min}_{msd}$

2 **upon event** $\varphi_i(t) = 1 - \mathsf{offset}_i$ **and** active $= 1$ **do** // preponed transmission

3 **trigger** $\mathsf{broadcast}_i(\varphi_i(t), \mathsf{dev}_{avg})$

4 **upon event** $recv_i(\varphi_j, \mathsf{dev}_j)$ *from* p_j **do** // received new sync-message

5 eventset := eventset $\cup \{\varphi_i(t) + 1 - \varphi_j\}$

6 recvphase := recvphase $\cup \{\varphi_i(t)\}$

7 devset := devset $\cup \{\mathsf{dev}_j\}$

8 **upon event** $\varphi_i(t) = 1$ **do** // threshold reached

9 $\varphi_{last} := \delta_{last} := \Delta_i := 0$ // clean up

10 clique := $\mathtt{CD}(\{\varphi - \mathsf{mde} \mid \varphi \in \mathsf{eventset}\} \cup \{1\})$

11 **if** active $= 0$ **then** clique := $\mathtt{CD}(\{\varphi - \mathsf{mde} \mid \varphi \in \mathsf{eventset}\})$

12 **if** $|\mathsf{clique}| = |\mathsf{eventset}| + \mathsf{active}$ **then** discovered := false **else** discovered := true

13 $\mathtt{reduce}^f(\mathsf{eventset})$ // remove f highest and f lowest deviations

14 $\mathsf{dev}_{max} := \max(S(\mathsf{eventset}))$

15 $\mathsf{dev}_{min} := \min(S(\mathsf{eventset}))$

16 $\mathsf{dev} := \max(\mathsf{dev}_{max} - \mathsf{dev}_{min}, |\mathsf{dev}_{max}|, |\mathsf{dev}_{min}|)$

17 **if** $\mathsf{dev} > w$ **then** devset := $\emptyset$

18 $\mathsf{mde}' := \max(0, \mathsf{avg}(\mathtt{reduce}^f(\mathsf{devset})) - (\mathsf{dev}_{max} - \mathsf{dev}_{min})/2)$

19 $\mathsf{mde} := \mathsf{mde} + (\mathsf{mde}' - \mathsf{mde}) \cdot \sigma_{mde}$

20 **if** discovered = false **and** $\mathsf{dev} \geq \frac{1}{L}$ **then** // Execute E-RFA

21 **for each** $\varphi_j \in$ eventset *in increasing order* **do**

22 **if** $\Delta_i + \varphi_j < 1$ **and** $\varphi_{last} + \delta_{last} < \varphi_j$ **then**

23 $\delta_{last} := \min(1, (\varphi_j + \Delta_i) \cdot \alpha) - (\varphi_j + \Delta_i)$

24 $\Delta_i := \Delta_i + \delta_{last}$

25 $\varphi_{last} := \varphi_j$

26 **else** // Execute FTA

27 $\Delta_i := -\mathsf{avg}(S(\mathtt{reduce}^f(\mathsf{clique})))$

28 $\varphi_i(t) := \Delta_i$ // Apply reachback response

29 active := $\mathtt{APM}(\mathsf{active}, \mathsf{recvphase}, \mathsf{offset}_i)$ // Check for passive mode

30 **if** $\mathsf{dev} > w$ **then** active := 1

31 $\mathsf{offset}_i := \mathtt{DESYNC}(\mathsf{recvphase}, \mathsf{offset}_i, \mathsf{mde}, r^{min}_{msd}, r^{max}_{msd})$

32 **if** $\mathsf{offset}_i < 0$ **then** active := 0 // Check on successful offset desync

33 $\mathsf{dev}_{avg} := \mathsf{avg}(S(\mathsf{eventset}))$

34 devset := $\{\mathsf{dev}_{avg}\}$

35 eventset := recvphase := $\emptyset$

Clearly, the developer must again ensure that the difference between the two predefined offset bounds r^{min}_{msd} and r^{max}_{msd} is large enough. For instance, for some given bandwidth limit B and the assumption that the system never contains more than n_{max} nodes configured in single-hop topology, the inequality $n_{max} < \left\lfloor \frac{r^{max}_{msd} - r^{min}_{msd}}{B} \right\rfloor$ must always hold. Interestingly, several simulation results have shown that the bound has to be even tighter, that is

$$n_{max} < \left\lfloor \frac{r^{max}_{msd} - r^{min}_{msd}}{2B} \right\rfloor .$$

In [DRPN07, PDN07, DN08], the authors analyzed and proved the correctness of the desynchronization approach in the case of a single-hop network. They also analyzed multi-hop networks. However, due to several indeterministic properties of multi-hop networks (*e.g.*,

hidden terminal problem), no guarantees can be made for the correctness and convergence time in multi-hop networks.

In spite of everything, we decided to use the desynchronization approach also in multi-hop networks and compare the results with the standard approach. This comparison is presented and discussed in the next chapter.

Algorithm 15: Offset desynchronization

```
1  procedure DESYNC(recvphase, offset, mde, r_msd^min, r_msd^max)
2      φ_prev := max(1 − r_msd^max, prev(recvphase, 1 − offset + mde) − mde)
3      φ_next := min(1 − r_msd^min, next(recvphase, 1 − offset + mde) − mde)
4      offset := 1 − (φ_next + φ_prev)/2
5      if CCA failed during last transmission then
6          fallowset := ∅
7          for each φ ∈ recvphase ∪ {1 − r_msd^max + mde, 1 − r_msd^min + mde} do
8              φ_next := next(recvphase, φ)
9              if φ_next ≠ ∞ and φ_next − φ > 2B then
10                 fallowset := fallowset ∪ {1 − (φ_next + φ)/2 + mde}
11         if fallowset ≠ ∅ then offset := rand(fallowset) else offset := −1
12     return offset
```

5.5 Discussion

Within this chapter, we developed an efficient and robust pulse synchronization algorithm for single-hop networks and general multi-hop topologies. It must be noted that our approach mainly differs from other synchronization protocols in wireless networks from the fact that we establish pulse synchronization. The main advantage is that, in most cases, this primitive provides a faster convergence time compared with other clock synchronization algorithms (*e.g.*, Asynchronous Diffusion [LR06]). Furthermore, in contrast to the implementation of general clocks, pulse clocks make use of a simple cycle concept. The simplicity and elegance of this concept allows it to be used as an optimal building block for complex communication protocols. In addition to that, a pulse clock can be simply implemented either in hardware or in software. Unfortunately, there may exist initial configurations which are in the vicinity of an unstable fixpoint and consequently affect the convergence time. However, in complex networks the probability that the nodes initialize exactly in such a fixpoint is neglectable. In addition to that, even if the network entered such a fixpoint, it highly likely exits this configuration due to natural imprecisions resulting from the delay jitter and the calculation.

We have shown that the original RFA cannot be used for self-stabilizing pulse synchronization in the presence of Byzantine faulty nodes. Another weakness of this approach is the bad synchronization precision which results from its asymmetric behavior. This problem was compensated by switching to the FTA scheme. The switching is based on the actual precision and the FTA convergence threshold L as shown in Algorithm 7. In Theorem 6, we analyzed the parametrization of this constant and found out that this parameter heavily depends on the network topology. Unfortunately, the nodes in a WSN are usually randomly scattered. This does not allow an *a priori* analysis of the network structure. However, with respect to this problem, an interesting elaboration of random networks was done by Wang *et al*. in [WC03]. In short, networks can be characterized according to some basic concepts. These are the network size N, the average path length L, the clustering coefficient C, and the degree distribution $P(k)$.

As we have shown, single-hop networks are powerful topologies which allow the establishment of several improvement strategies. For instance, beside the precision improvement and the distributed message delay estimation, the lifetime of highly redundant networks can be increased by establishing a listen/sleep schedule and additionally an APM approach. Interestingly, the presented message delay estimation, originally developed for single-hop networks, also works well in multi-hop networks as shown via simulations in the next chapter. This is of major importance in cases where MAC-layer time-stamping[6] is not provided by the MAC layer as it is the case for IEEE MAC 802.15.4. However, multi-hop networks implicate a problem. For instance, in the case of f Byzantine faulty nodes, it has to be ensured that the communication topology is at least $(5f+1)$-connected or in the additional use of the drift compensation approach at least $(7f+1)$-connected. This constraint cannot be guaranteed, if the network is randomly created. On this account, straight randomly scattered WSNs are of less interest, if faulty nodes have to be considered. Therefore, it is essential to interfere in the network creation process such that the connection constraint can be guaranteed with a higher probability. One technique of such a creation strategy can be a grouped multi-hop topology and is analyzed in more detail in Chapter 6. Therein, nodes are assumed to be scattered in a clustered way which allows a better exploitation for fault tolerance.

By the use of offset desynchronization, single-hop networks additionally offer the possibility to implement a self-organizing resource management strategy. This means that the nodes have knowledge of the occupied and available bandwidth for exchanging synchronization messages. This reduces the probability of message collisions and comes along with an improved synchronization precision.

[6] MAC-layer time-stamping was first proposed in [GKS03b].

CHAPTER 6

Evaluation by Simulation

OVERVIEW

The main objective of this chapter is threefold. First, simulation results are used to validate correctness and theoretical aspects of the devised algorithms presented in Chapter 5. Furthermore, this kind of evaluation allows us to easily gain more insights about the behavior of the different approaches. A third important reason for simulation is that several properties can be measured much easier and faster. In the beginning of this chapter, different evaluation metrics are presented in order to reasonably compare the algorithms according to their flexibility and performance aspects with respect to different network topologies and parameter choices. Since many communication protocols used in WSNs are based on a non-beacon enabled IEEE 802.15.4 MAC [Soc03] layer, we analyzed this communication standard in order to get some important communication specific parameters. Section 6.1 gives an overview of the used simulator and the modifications we have done in order to simulate the presence of clock drift and the behavior of a non-beacon enabled IEEE 802.15.4 communication stack. The corresponding parameters are illustrated in detail in Section 6.5. Several evaluation metrics that are necessary for a solid comparison of the simulation results are presented and discussed in Section 6.3. In contrast to single-hop networks, randomly created multi-hop topologies cannot be characterized solely on the number of comprised nodes. On this account, Section 6.4 discusses different basic concepts that can be used for characterizing general complex network topologies. These concepts are later used for comparing the simulation outcomes with respect to different network characteristics and topologies. A lot of simulation results regarding single-hop networks are illustrated and discussed in Section 6.6. Similarly, Section 6.7 presents and discusses the results regarding different types of multi-hop topologies reaching from simple chain networks to complex grid and ring topologies. Therein, emphasis is also put on a special topology type named grouped multi-hop topology. Such networks are very robust with respect to Byzantine faulty nodes and therefore give rise to establish new concepts in WSNs.

6.1 Simulating the MAC Layer

We used a probabilistic wireless sensor network simulator called JProwler[1] which is basically configured to simulate the behavior of the Berkeley Mica Motes running TinyOS with the B-MAC protocol. It is a Java version of the Matlab-based Prowler[SVML03] network simulator. Both Prowler and JProwler can be used for verifying and analyzing communication protocols of ad-hoc wireless sensor networks. Since the B-MAC communication protocol is very similar to the IEEE 802.15.4 standard (*e.g.*, both implement the same CSMA/CA mechanism), we simply modified several MAC layer specific attributes in order to simulate the behavior of an IEEE 802.15.4 MAC layer.

According to the identified parts of the transmission delay (Section 3.1.1), the *serialization delay* t_{sd} is based on the amount of transmitted data. In our case, we assume that the synchronization frame contains at least a frame-identifier (8 bit), the actual phase $\varphi_i(t)$ (24 bit), the average phase deviation dev_{avg} (24 bit), and a checksum (8 bit). We further assume that the frame also contains the parameters required by the drift calibration algorithm. These are the unique identifier of the sender id_i (16 bit), the phase adjustment value $h_i(t)$ (16 bit), the summarized slope $\Sigma_{s_i}(t)$ (16 bit), and the timestamp of the sender's hardware clock $HC_i(t)$ (32 bit). In sum, the application needs about 18 bytes for a single synchronization message. The real amount of transmitted data is greater due to the payload of the MAC and the physical layer. According to the IEEE 802.15.4 MAC standard, 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 ISM band, the bit rate is 250kbps. As a result the serialization delay equals $t_{sd} = [33 \cdot 8/250000]s = 1056\mu s$ and is hereinafter assumed to be about $1ms$. To calculate the *propagation delay* t_{prop}, we need the maximum radio range which is typically at most 100 meters in the case of WSNs. If we assume a typical propagation velocity of $v = 200.000km/s$, then the propagation delay is about $t_{prop} = 0.5\mu s$.

The estimation of the *send time* t_{send} and the *receive time* t_{recv} is more difficult, because both mainly depend on the implementation layer where the transmission is invoked and the reception event is finally triggered. This delay can be influenced by interrupts and buffer handling. In our case, we assume that the synchronization algorithm is directly based upon the MAC layer an consequently introduces a small average delay of about $1ms$ in combination with a relatively small delay jitter of about $500\mu s$.

The *access time* t_{acc} solely depends on the configuration of the backoff scheme in the CSMA/CA mechanism and can introduce a high average delay of at least $8ms$ with a delay jitter of $4ms$ or higher. For instance, in [Mah06, p. 44], Mahalik states that if the IEEE 802.15.4 MAC layer is configured with the default parameters and if we set the MAC attribute *macMinBE* to 0, then the average delay does not exceed $8ms$.

However, this high delay jitter is unacceptable for our synchronization algorithms, if we want to achieve a worst case synchronization precision which is lower than $1ms$. There exist two alternatives for eliminating the highly variable access delay. The first one may be the additional implementation of MAC layer time-stamping. That is, at the sender side, the MAC layer measures the time between the broadcast invocation and the time when the first bit is transmitted. This time duration is added to the corresponding message. Similarly, at the receiver side, the MAC layer measures the time between the reception of the first bit and the time when the reception event is triggered. Since the serialization delay is deterministic, the receiver is able to calculate a highly accurate estimation of the complete transmission delay. The remaining average uncompensated delay and the corresponding delay jitter may

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

be in the order of microseconds. Unfortunately, the IEEE 802.15.4 standard does not provide mechanisms which can be used for such a low level time-stamping.

A second approach which reduces the amount of delay jitter is based on a reconfiguration of some MAC specific attributes that control the CSMA/CA algorithm. These parameters are described in the MAC PAN information base (PIB) [Soc03]. In detail, we assume that both MAC attributes *macMinBE* and *macMaxCSMABackoffs* are set to 0. As a consequence, the backoff scheme and the collision avoidance mechanism of the CSMA/CA algorithm are disabled. In other words, if a transmission is invoked, the MAC layer directly performs CCA which requires 20 symbol periods[2]. Afterwards, if the channel is assessed to be free, the message will be transmitted. Otherwise, a channel access failure is declared. Clearly, this may dramatically decrease the message throughput. However, several simulation results have shown that with respect to time determinism, in many cases it is better to omit messages than to transmit strongly delayed messages that additionally observe a high delay jitter. To sum up, the second approach reduces the access time to exactly the time duration used for the CCA. That is, $t_{acc} = 128\mu s$.

It should be mentioned that the DESYNC approach in Algorithm 14 requires knowledge about the result of the last CCA. Therefore, a real world implementation requires a MAC layer that provides such a service. A further requirement of the underlying MAC layer is that it must not postpone the transmission, if a current transmission of another message or a reception process is active. If so, then we associate this operation with a failed clear channel assessment such that the DESYNC algorithm (Algorithm 15) tries to allocate a new offset where enough bandwidth is available. JProwler's virtual MAC layer class was modified to overcome this preconditions.

Finally, when using the disabled backoff approach and the requirement that a transmission is discarded instead of a suspension if a transmission or reception process is currently going on, then we can assume that, as long as some message m is eventually delivered, the message delay $d(m)$ of message m is always in the range $d(m) \in [d, d+\varepsilon]$ where $d = 2.2ms$ and $\varepsilon = 500\mu s$. These parameters are reflected in JProwler's specific MAC constants *sendMinWaitingTime*, *sendRandomWaitingTime*, *sendMinBackOffTime*, *sendRandomBackOffTime*, and *sendTransmissionTime*. We additionally implemented a new MAC constant named *csmaMaxBackOffCnt* which represents the *macMaxCSMABackoffs* attribute of the IEEE 802.15.4 standard. Table 6.1 lists the configured parameters of JProwlers virtual MAC layer. Note that these parameters are used for all simulations. Further note that the simulated jitter is uniformly distributed and therefore affects the communication more than in reality where the delay jitter usually follows a Gaussian distribution.

Table 6.1: Parameter configuration of JProwler's virtual MAC layer.

sendMinWaitingTime	1.2 ms
sendRandomWaitingTime	0.5 ms
sendMinBackOffTime	1 ms
sendRandomBackOffTime	2 ms
csmaMaxBackOffCnt	0
sendTransmissionTime	1 ms

The configuration of the transmission strength and noise is the same as originally implemented for the Mica2 motes and corresponds to a transmission range of about 25 meters. JProwler originally provides two methods for simulating noise. Since our simulations are based on statically placed nodes, we used the Gaussian radio model. Thus, if a node starts a

[2] According to [Soc03], if the $2.4GHz$ ISM band is used, then one symbol equals $16\mu s$.

transmission, all other nodes perceive an additional Gaussian noise which strongly depends on the geographical distance of the origin.

Beside the modified MAC layer of JProwler, the graphical user interface was enhanced by several new dialogs which enables the user to modify various parameters during the simulation. We further extended the simulator by an oscillator model. Thus, every virtual node is based on an adjustable oscillator (*e.g.*, RC-oscillator or several crystal cuts). This allows the simulation of clock drift and its influence on the clock synchronization. Due to the fact that the frequency of an oscillator heavily depends on the supply voltage and the ambient temperature, the enhanced JProwler also contains the simulation of the ambient temperature. Other new features consider the adjustment of the simulation speed and enabling/disabling nodes during the simulation. The supply voltage and its influence on the drift was not incorporated in the modification, because this was of less importance with respect to this thesis.

6.1.1 Simulating Erroneous Nodes

Simulating the behavior of an erroneous fault is not a simple task, especially if a set of nodes should be Byzantine faulty in a way that they behave in an adversary manner and consequently are able to collaborate in order to perform the worst case attack against the system. Since such a worst case attack usually assumes that the attackers have unbounded processing resources and the fact that the adversary is only a theoretical model, it is nearly impossible to practically implement such a faulty behavior. On this account, we implemented erroneous nodes instead of Byzantines nodes which are still able to send different incorrect messages to the distinct neighbors, but do not collaborate or perform special worst case attacks. In other words, erroneous nodes have a limited attack coverage compared to Byzantine nodes, but provide realistic settings in a way that they have a strong impact on the convergence time and the precision.

Fault tolerance mechanisms are usually aimed at removing outliers. In our case, we do not only detect and remove the outliers. Instead, we assume a fixed upper bound f of erroneous nodes that may simultaneously exist in the network and always remove the f highest and smallest values of the gathered information of all nodes that participate in the synchronization process. If the system is highly redundant, then this strategy is very effective, especially if the parameters received from the faulty nodes are out of range according to an *a priori* defined value domain. A typical worst case attack with respect to the FTA algorithm occurs when a subset of correct nodes receive the same incorrect values from the faulty nodes which are situated at the left end of the deviation set, and the remaining correct nodes receive incorrect values which are all situated at the right end of the deviation set from the same faulty nodes (two-faced malicious behavior). However, with respect to the RFA approach, such extreme faulty values have only a little impact on the convergence behavior, because in this case a node reacts stronger to a node that is in the vicinity of the own node.

A combination of both attacks affecting the FTA-RFA approach is realized by randomly selecting one out of three predefined attacker models as listed below. All three attacks only modify the content of the transmitted actual phase φ_i of the faulty node p_i. For sake of simplicity, all three attacks chose a random value within the interval $[0, w]$ for the average phase deviation dev_{avg}, because this parameter hardly affects the synchronization precision.

1. *Out of range attack*: The values contained in a faulty message are randomly chosen and are usually beyond the reasonable value domain. Therefore, the faulty messages can be also easily detected as outliers. In detail, the actual transmitted phase φ_i is a random value between 0 and 1. This attack should degrade the precision of the FTA synchronization approach.

2. *Strong vicinity attack*: This type of attacks is aimed at affecting the RFA part of the presented synchronization approach. Therefore, the actual phase contained in the faulty synchronization message is randomly set either to $\varphi_i = \varphi_j(t_r) - w$ or to $\varphi_i = \varphi_j(t_r) + w$ such that the receiving node just reacts to the information. For this, $\varphi_j(t_r)$ denotes the time when the receiver node p_j receives the message at real-time t_r.
3. *Weak vicinity attack*: This attack is similar to the second attacker model, but selects a random value between $\varphi_j(t_r) - w$ and $\varphi_j(t_r) + w$ for φ_i.

In the case the drift compensation approach of Algorithm 8 is implemented in addition to the FTA-RFA approach, an erroneous node is also able to forge several more parameters. In detail, an erroneous node p_i can transmit different incorrect unique identifiers id_i, phase adjustment values h_i, summarized slope values Σ_{s_i}, and different timestamps of the sender's hardware clock $HC_i(t)$ to all distinct neighbors. Similarly to the aforementioned FTA-RFA attacks, an erroneous node may also transmit different random values for all three parameters to all distinct neighbors.

Since the vicinity attack require some knowledge of the receiver node, all attacks are implemented and simulated at the receiver side of each node. In detail, an erroneous node correctly synchronizes to the rest of the network, broadcasts the correct message, but the receiver then modifies the received information to simulate the two-faced malicious behavior of the erroneous nodes.

Additionally, an erroneous node does not pay attention on the actual transmission of another node and ignores a failed clear channel assessment. In other words, an erroneous node may intentionally start a transmission immediately after a neighboring node started its transmission and consequently causes a message collision. To simulate this behavior, several parts of the virtual MAC layer were modified in order to support the opportunity of a *forced message transmission*. In order to increase the number of message collision, a Byzantine node always calculates a random offset irrespective of an activated desynchronization approach and then always performs a forced transmission based on this offset.

6.2 Evaluation Types

We mainly distinguish between two evaluation approaches:

1. *Prepared evaluation*: This scheme is based on an *a priori* defined initial system configuration including the number of nodes, network topology, and initial phase and drift settings. Different variants of the synchronization algorithm are then executed with the same initial configuration. This allows a reasonable comparison as it would be in the case of a randomized initial system configuration.
2. *Randomized evaluation*: This comparison scheme is based on a randomized initial configuration whereas the network topology and the number of nodes is assumed to be the same. Only the phase and drift settings are initialized in a random way. Different variants of the synchronization algorithm are then executed hundred times, each based on such a random initial configuration. Worst case and average results among these simulations are then used for sake of comparison. This scheme is especially used for comparing the average time required until achieving network synchronicity with respect to different algorithm variants and parameter choices.

Whereas the prepared evaluation is mainly used for the performance comparison with respect to different synchronization variants and parameter choices, the randomized evaluation is used for covering other configurations and provide statistical statements about the performance aspects (*e.g.*, average time to sync).

6.3 Evaluation Metrics

The comparison of the simulation results according to different parameter choices is done with respect to several evaluation metrics that together represent the performance aspect of an implemented approach.

Percentage of synchronized nodes $p_{sync}(t)$**:** As the name implies, this evaluation parameter declares the percentage of nodes that entered the synchronization state. Note that a node only enters this state, if the measured maximum absolute deviation with respect to the neighboring nodes is within the synchronization window w for 10 out of the last 11 periods. This ensures that, in the border case, the nodes do not frequently alternate between the synchronized and the unsynchronized state. This is of major importance in the case the time until all nodes persist in a stable synchronized state has to be measured. Empirical studies turned out that a number of 10 periods is enough to ensure a highly probable stable synchronization state.

Percentage of active nodes $p_{active}(t)$**:** This parameter is only interesting, if the APM approach is activated. If so, then $p_{active}(t)$ declares the percentage of nodes that are active and thus participate in the synchronization process. Otherwise, this parameter always equals 100%.

Time to sync t_{sync}**:** This evaluation parameter defines the duration of real-time until all nodes have entered the synchronization state (*i.e.*, $p_{sync}(t_{sync}) = 100\%$) and keep therein until the simulation ends at time t_e (*i.e.*, $p_{sync}(t') = 100\%$ for all $t_{sync} \leq t' \leq t_e$).

Average interval duration $T_{avg}(t)$**:** This evaluation parameter declares the average over the actual real-time duration of a cycle of all nodes in the system. This parameter is of major importance if the drift compensation approach (Algorithm 8) is activated in parallel to the FTA-RFA algorithm. This parameter should be kept almost constant over time.

Precision $\Pi(t)$**:** The precision parameter $\Pi(t)$ declares the maximum deviation between any two nodes in *real-time* at time t. According to Definition 11, $\Pi(t)$ must not be greater than the worst case precision Π, if the pulse synchronization problem should be solved. If $T_{avg}(t)$ is getting very small compared to the nominal cycle time duration T, then the nodes may strongly deviate with respect to their phase, however, the precision may still be small. Thus, the precision parameter is useless without the declaration of $T_{avg}(t)$. On this account, we use the definition of group spread as described next instead of $\Pi(t)$ in order to characterize the quality of synchronization precision.

Relative Group spread $\varphi_\Pi(t)$**:** The *relative group spread* $\varphi_\Pi(t)$ is the actual maximum phase deviation between any two nodes in the system over the time and cannot be greater than $1/2$. The corresponding *absolute group spread* is simply scaled by the cycle granularity Φ and is defined as $\Phi_\Pi(t) = \varphi_\Pi(t) \cdot \Phi$. According to Definition 9, $\Phi_\Pi(t)$ should never exceed a predefined maximum phase deviation Φ_Π in order to solve the pulse synchronization problem. For sake of illustration, we hereinafter use the definition of the group spread in the time domain. That is, $T_\Pi(t) = \varphi_\Pi(t) \cdot T$ where T is the nominal time duration of one cycle. Note that if the average interval T_{avg} equals the nominal time interval T, then the precision $\Pi(t)$ also equals $T_\Pi(t)$. However, if $T_{avg} > T$, then $\Pi(t) > T_\Pi(t)$ and reversely.

Maximum phase adjustment $\varphi_{adj}(t)$**:** This parameter declares the maximum phase adjustment among the latest phase adjustments of all nodes in the system at time t. If $\varphi_{adj}(t)$ is very small, then the nodes hardly deviate at the beginning of the phase. However,

a large $\varphi_{adj}(t)$ means that the nodes strongly adjust their phase clock after a synchronization period but may still maintain a very small worst group spread throughout the rest of the phase (ignoring the part where not all nodes have already performed their phase adjustment). A large $\varphi_{adj}(t)$ is also very problematic, if the coordination of a nodes' application depends on a small phase value which is smaller than the phase adjustment. For sake of illustration, we use the maximum phase adjustment in the time domain, denoted by $T_{adj}(t) = \varphi_{adj}(t) \cdot T$.

Number of message omissions per round $N_{om}(t)$**:** $N_{om}(t)$ denotes the number of message omissions at all sending nodes per period over time t. A message omission at node p_i occurs if the broadcast transmission of a message was discarded due to a failed clear channel assessment. That is, the wireless medium was occupied at the time p_i wanted to transmit a message. A perfect communication network should never contain message omissions. The presence of Byzantine nodes and a small message staggering delay increases the number of message omissions per period.

Number of message collisions per round $N_{col}(t)$**:** $N_{col}(t)$ denotes the number of message collisions at all receiving nodes per period over time t. A message collision at node p_i occurs if the reception of a message was corrupted due to high noise or interference resulting from a forced transmission of a faulty node or the hidden terminal problem. A perfect communication network should never contain message collisions. The presence of Byzantine nodes and a small message staggering delay increases the number of message collisions per period.

Since the comparison of the aforementioned evaluation metrics over time with respect to different simulations in one single plot is very confusing, we decided to compare only some characteristic distribution parameters. In particular, we use the concept of box plots as visualized in Figure 6.1. A boxplot is typically constructed by the use of a box and an error bar in both directions which are called whiskers. The bottom and top of the box represent the 25th and 75th percentile, respectively. The box is also divided by a line which indicates the 50th percentile (*i.e.*, the median). The range of the box (*i.e.*, the range between the 25th and 75th percentile) is called the Interquartile Range (IQR). Data values which are situated more than $1.5 \cdot \text{IQR}$ lower than the 25th percentile or $1.5 \cdot \text{IQR}$ higher than the 75th percentile are considered as outliers. The end of a whisker is indicated with a small horizontal line and represents the smallest or highest data value which is not an outlier. The end of the whiskers are connected with the corresponding minimum and maximum existing data value by a dashed line.

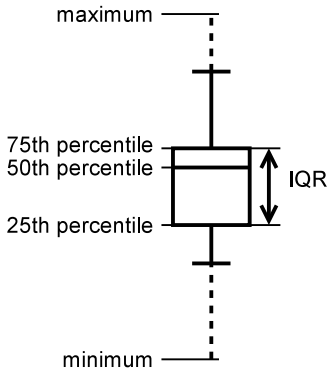


Figure 6.1: Demonstration of a simple boxplot.

In order to avoid incorrect results due to settling effects during the startup phase, we decided to calculate the distribution parameters only in the time interval $[(t_s+t_e)/2, t_e]$, where t_s is the first time when all nodes achieved synchronicity for a sufficient long time period and t_e is the time when the simulation ends.

6.4 Network Characterization

In the following, different network topologies are presented, each typically used for comparing the effectiveness and quality of the applied algorithms. Note that JProwler allows only an indirect definition of the network topology by a geographical placement of the nodes in a virtual environment. As a consequence, a change in the signal power will lead to a change in the communication structure and thus can be used to control the topology. Note that JProwler does not consider fading and shadowing effects.

The All-to-all Topology This topology is the most simple one and corresponds to a fully connected communication network. The only network parameter used in this case is the number of contained nodes n. The all-to-all topology is of major importance with respect to scalability tests regarding different network sizes.

The Regular Grid Topology This evaluation topology distributes the nodes on a two dimensional regular grid such that the inner nodes have exactly four neighboring nodes, one in each direction. Therein, a node can only communicate with its direct neighbors. This topology can be compared with the regular distribution of radio cells in a cellular network. As later discussed, simulation results in such a network are promising regarding the achievable precision and time to sync. Within this thesis, we use only squared grid topologies. Therefore, the grid size g defines the number of nodes in both directions (*e.g.*, $g = 4x4$, $g = 6x6$, *etc.*) and is the only important network parameter in this case. In order to reduce interference noise, two neighboring nodes are geographically situated exactly 15 meters apart. This definitely ensures connectivity and less message collisions.

The Chain Topology This topology equals a simple multi-hop network where the nodes are arranged along a chain. Thus, whereas the two border nodes can only communicate with one neighbor, the inner nodes have two neighbors. The only network parameter in this case is the network size n, where $n-1$ indicates the number of hops in the network. Similarly to the grid topology, two neighboring nodes are again geographically situated 15 meters apart.

The Grouped Multi-hop Topology The grouped multi-hop topology is a special communication topology, we suggest for a more robust synchronization in large multi-hop networks, especially if Byzantine faulty nodes are present. This topology type results from the assumption that it is possible to interfere in the network creation process such that the nodes form several groups where nodes within the same group are configured in an all-to-all topology and the groups itself are distributed in a chain-like topology. As a consequence, all nodes in one group can communicate with all nodes in the direct neighboring groups. An example of such a scattering technique may be the periodic drop of a group of nodes out of an air plain while flying over an area which has to be observed. The reason of the more robust nature in this case is the fact that, in the case of a group size of k nodes, this increases the probability of a k-connected network. In contrast of a uniform dropping sequence, this probability will shrink. This leads to the fact that a maximum number of faulty nodes can be tolerated to exist in parallel in each group. The topology parameters used to describe the

network characteristics in this case are the group size g, and the number of groups G. In order to avoid a high rate of message collisions due to noise and interference, two neighboring groups are geographically situated 15 meters apart. However, nodes within the same group are geographically situated at most 1 meter apart.

The Ring Topology The ring topology is of special theoretical interest with respect to the possibility of self-stabilization in the case the FTA convergence threshold value L is set too low. In contrast to the bidirectional communication pattern, we sometimes simulate ring topologies with a unidirectional communication pattern where the nodes can only communicate clockwise. In this case, the pattern is explicitly mentioned. The only topology parameter is the number of nodes n. In order to avoid interference and noise, the nodes are distributed in an equidistant order around a circle which has a diameter of $d = 10 \cdot n/\pi$ meters. Consequently, the geographical distance on the circle between two neighboring nodes equals 10 meters.

Random Geometric Topology The random geometric topology type is of more practical interest. Therein, we assume that the nodes are scattered randomly on a two dimensional square field of $100x100$ meters. Simulation results are then performed with a network size of $n = 100$ nodes and with respect to different parameter choices. The network size n is also the only important topology parameter in this case.

6.5 General Simulation Parameters

Beside the fixed configuration of the MAC layer, several other parameters required by the implemented algorithms are almost the same and listed in Table 6.2. In order to illustrate the effectiveness of the drift compensation approach, we assume that every node suffers from a different clock drift. Therefore, JProwler was extended to support virtual oscillators. In detail, the nodes local clock is based on the drift rate of the underlying virtual oscillator relative to the global simulation time. Since RC-oscillators typically suffer from the greatest and most sensitive drift rate, we parametrized the virtual oscillator to have a random initial drift rate between -10^5ppm and $+10^5$ppm. Typical existing wireless nodes used in WSNs (*e.g.*, Mica2 motes) have a nominal clock frequency of $f_{clk} = 8MHz$. Thus, the best achievable granularity would be about $125ns$ and is better than the granularity of $1\mu s$ as used in our simulation environment. Since we expect a worst case synchronization precision in the order of a few milliseconds, we set the synchronization window w to 10ms. The virtual time duration of a simulation is set to be at least 10000 seconds or longer where necessary. We further assume that the ambient temperature is almost stable or at least varies very slowly (*e.g.*, 1°C per minute). The meaning of the other parameters is already described in the previous chapter. Most of the parameters are assumed to be the same in all simulations unless mentioned otherwise.

In the case of a prepared simulation evaluation, all nodes p_i have a pre-defined initial phase $\varphi_i(0)$ and drift-rate ρ_i. These parameters for up to 100 nodes are stated in Table 6.3.

6.6 Simulating Single-hop Topologies

This section presents and discusses the quality improvements that can be achieved by using different combinations of the presented approaches with respect to the network size and other parameters in an all-to-all topology. Therefore, we first consider the fault-free case and then compare the results with the existence of f erroneous nodes.

Table 6.2: The general parameter choice used in all simulator experiments.

Parameter	Symbol	Value
Oscillator technology		RC-oscillator
Maximum drift rate	ρ	10^5 ppm
Period time	T	1 s
Granularity	Φ_{th}	10^6 ticks/period
Minimum relative message staggering delay	r_{msd}^{min}	0.01
Maximum relative message staggering delay	r_{msd}^{max}	0.3
Coupling factor	α	1.01
Synchronization window	w	10 ms
Evaluation end	t_e	10000 periods
Robust FT-DC approach (Algorithm 8)		enabled
FTA-RFA approach (Algorithm 7)		enabled
FTA convergence threshold	L	4
Clique discovery approach (Algorithm 10)		enabled
MDE approach (Part of Algorithm 11)		enabled
MDE smoothing factor	σ_{mde}	$1/2$
APM approach (Algorithm 13)		enabled
Offset desynchronization approach (Algorithm 15)		enabled
Bandwidth limit	B	4 ms

6.6.1 The Fault-free Case

An extensive study about the results with respect to different coupling factors was already done in [WATP$^+$05] and [LE09]. Therein, the main outcome is that the group spread does not depend on α, if α maintains the lower bound. A higher coupling factor mostly reduces the time to sync as long as it maintains the upper bound. On this account, we do not concentrate on the simulation with respect to different coupling factors. Instead, we adjust α as denoted in Table 6.5 in dependence of the network size and compare the results with respect to the different variants of our FTA-RFA approach.

Table 6.5 summarizes the upper bound for different values of α with respect to Corollary 1. The stronger bound of Theorem 2 was neglected due to the fact that most network simulations using the weak upper bound with a random initial configuration have achieved synchronicity. For sake of simplicity, all simulations are done with a coupling factor of $\alpha = 1.01$ as presented in Table 6.2. In our case, this is valid for systems containing at most 100 nodes. However, even in the case of $n = 100$, all simulations with $\alpha = 1.01$ achieved synchronicity. The time to sync calculated in Table 6.5 are based on Theorem 3 for an initial maximum phase difference of $\Phi_0 = 0.4$.

In the case of an enabled drift calibration approach, we have to get an estimation of the achievable drift among the nodes in order to calculate the worst case precision. On this account, a simulation based on a network comprising 8 nodes was performed. Since the presence of Byzantine nodes usually degrade the achievable drift deviation, we configured one node to simulate a Byzantine-like behavior. The message staggering delay is further assumed to be large enough to keep the amount of message omissions and collisions very small. Therefore, the maximum relative message staggering delay r_{msd}^{max} was set to 0.3. The real-time deviation of the pulse clock duration between the slowest and the fastest node over time of this simulation is visualized in Figure 6.2b. This histogram compares two approaches, one with disabled and the other with enabled drift stabilization. The maximum interval deviation never exceeded $70\mu s$ whereas the simulation with a disabled drift stabilization contained some outliers of up to $130\mu s$. Figure 6.2a further visualizes the average real-time duration T_{avg} of the pulse clock periods over the time and demonstrates the effectiveness of the drift stabilization approach. In detail, whereas the drift stabilization approach

Table 6.3: Initial node configuration for a prepared simulation evaluation.

id_i	phase $\varphi_i(0)$	drift rate ρ_i [ppm]	id_i	phase $\varphi_i(0)$	drift rate ρ_i [ppm]
0	0/1000	−100000	1	500/1000	100000
2	50/1000	−100000	3	550/1000	50000
4	100/1000	−50000	5	600/1000	33333
6	600/1000	−25000	7	600/1000	25000
8	920/1000	42000	9	910/1000	41000
10	900/1000	40000	11	890/1000	39000
12	880/1000	38000	13	870/1000	37000
14	860/1000	36000	15	850/1000	35000
16	840/1000	34000	17	830/1000	33000
18	820/1000	32000	29	810/1000	31000
20	800/1000	30000	21	790/1000	29000
22	780/1000	28000	23	770/1000	27000
24	760/1000	26000	25	750/1000	25000
26	740/1000	24000	27	730/1000	23000
28	720/1000	22000	29	710/1000	21000
30	700/1000	20000	31	690/1000	19000
32	680/1000	18000	33	670/1000	17000
34	660/1000	16000	35	650/1000	15000
36	640/1000	14000	37	630/1000	13000
38	620/1000	12000	39	610/1000	11000
40	600/1000	10000	41	590/1000	9000
42	580/1000	8000	43	570/1000	7000
44	560/1000	6000	45	550/1000	5000
46	540/1000	4000	47	530/1000	3000
48	520/1000	2000	49	510/1000	1000
50	500/1000	0	51	490/1000	−1000
52	480/1000	−2000	53	470/1000	−3000
54	460/1000	−4000	55	450/1000	−5000
56	440/1000	−6000	57	430/1000	−7000
58	420/1000	−8000	59	410/1000	−9000
60	400/1000	−10000	61	390/1000	−11000
62	380/1000	−12000	63	370/1000	−13000
64	360/1000	−14000	65	350/1000	−15000
66	340/1000	−16000	67	330/1000	−17000
68	320/1000	−18000	69	310/1000	−19000
70	300/1000	−20000	71	290/1000	−21000
72	280/1000	−22000	73	270/1000	−23000
74	260/1000	−24000	75	250/1000	−25000
76	240/1000	−26000	77	230/1000	−27000
78	220/1000	−28000	79	210/1000	−29000
80	200/1000	−30000	81	190/1000	−31000
82	180/1000	−32000	83	170/1000	−33000
84	160/1000	−34000	85	150/1000	−35000
86	140/1000	−36000	87	130/1000	−37000
88	120/1000	−38000	89	110/1000	−39000
90	100/1000	−40000	91	90/1000	−41000
92	80/1000	−42000	93	70/1000	−43000
94	60/1000	−44000	95	50/1000	−45000
96	40/1000	−46000	97	30/1000	−47000
98	20/1000	−48000	99	10/1000	−49000

Table 6.4: Abbreviations of the different simulated algorithm variants. Note that all variants assume the parallel execution of Algorithm 8, which implements both approaches, drift calibration (DC) and drift stabilization (DS).

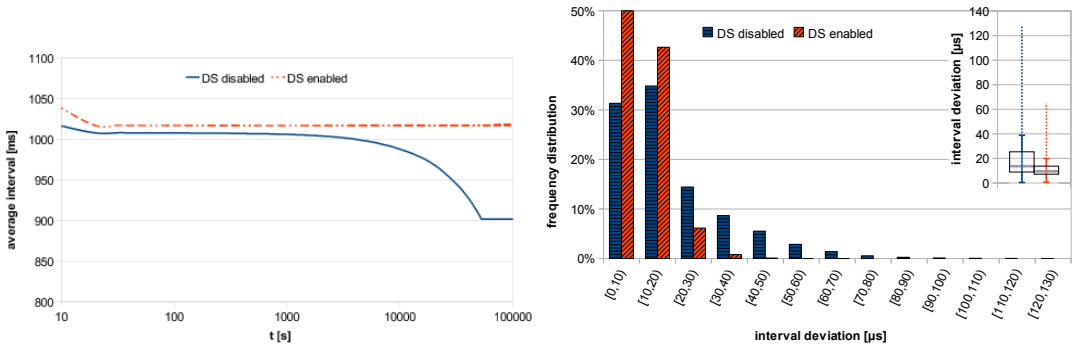
	Algorithm	R-RFA / FTA-RFA	DC	DS	CD	MDE	DESYNC	APM
V1	Alg. 6	R-RFA	✓	✓				
V2	Alg. 7	FTA-RFA	✓	✓				
V3	Alg. 9	FTA-RFA	✓	✓	✓			
V4	Alg. 11	FTA-RFA	✓	✓	✓	✓		
V5		FTA-RFA	✓	✓	✓	✓	✓	
V6	Alg. 12	FTA-RFA	✓	✓	✓	✓		✓
V7	Alg. 14	FTA-RFA	✓	✓	✓	✓	✓	✓
V8		FTA-RFA	✓	✓		✓		

takes effect after a settling time of about 50 periods resulting in a constant average interval, the other approach shows the occurrence of a common-mode drift where the average interval continuously decreases over time.

Note that the interval deviation of $70\mu s$ with respect to the average interval T_{avg} results in a new maximum drift rate $\rho' = 70\mu/T_{avg}$. If we additionally assume a worse interval deviation of $100\mu s$, then the new maximum drift rate can be assumed to be upper bounded by $\rho'_{max} = 100\mu s \cdot (1+\rho)/T$ with $T = 1s$ and $\rho = 10^6 ppm$ and consequently equals $\rho'_{max} = 110ppm$. Therefore, if we assume that the rate calibration scheme reduces the worst case drift to $\rho = 110ppm$, then based on the parameters from Table 6.2 and Theorem 1, the worst case precision for $\alpha > 1.002$ equals $\Pi^U = 2.322ms$. Note that without the rate calibration scheme, we would have $\Pi^U = 322ms$.

Table 6.5: Calculated bounds for the coupling factor and the time to sync.

Number of Nodes n	**5**	**10**	**20**	**50**	**100**
Upper bound for α	1.158	1.065	1.030	1.011	1.006
Tight upper bound for α	1.0439	1.0102	1.0025	1.0004	1.0001
Coupling factor α	**1.15**	**1.1**	**1.05**	**1.01**	**1.005**
Estimated time to sync [s]	7	10	18	82	163



(a) Average interval diagram

(b) Interval deviation histogram

Figure 6.2: Demonstration of the drift stabilization (DS) approach in a network comprising 8 nodes, where one node is erroneous, and a configured maximum relative message staggering delay of 0.3.

Algorithm version V1

The simulation results of Figure D.1 are according to algorithm version V1. That is, we executed the proposed E-RFA algorithm with respect to different network sizes and maximum message staggering delays. First, Figure D.1a compares the distribution of the time required for reaching a synchronized pulse state. Note that each distribution is based on hundred simulations, each started with a random initial configuration. The network always achieved synchronicity independent of the used parameter choices. From the diagram, we can deduce that, in average, the network achieves synchronicity after about 100 periods with outliers up to 400 periods, independent of the number of nodes and the maximum message staggering delay. Clearly, there exists a limit of the network size due to the upper bound of the coupling factor where the network may not achieve synchronicity any more. Similarly, there exists also a limit for the message staggering delay due to an increasing number of message omissions until the nodes are not able to communicate with each other any more. Fortunately, within the all-to-all topology, this limit is very small, because there are no faulty nodes which may disrupt the communication and the reception of at least the fastest node is enough to maintain network synchronicity.

The other diagrams are based on a prepared evaluation with an initial configuration of the nodes as stated in Table 6.3. The group spread distribution in Figure D.1c is almost the same for each block of simulations with the same network size. In general, we can deduce that the average group spread equals the message delay which agrees with our theoretical results. Note that in the case of 5 nodes, the group spread variance is much more higher compared to the other network sizes. This comes from the fact that the deviation of the cycle periods among the nodes is very high due to an increasing variation of the drift calibration. If more nodes are present, then more information about the average interval is available resulting in a more accurate clock adjustment. The higher group spread variance in the case of 100 nodes and $r_{msd}^{max} = 0.05$ is due to the limited communication bandwidth resulting in an exponentially increasing message collision and omission rate as shown in Figure D.1e and Figure D.1f, respectively. Consequently, less information is available for an accurate synchronization. To sum up, the main advantage of this approach is that the clock adjustments are always positive and additionally very small as visualized in Figure D.1d, because in the case of a synchronized network, the nodes always adjust to the fastest node.

Algorithm version V2

Simulation results from algorithm V2 are presented in Figure D.2 and show a lot of similarities with respect to variant V1. For example, the time to sync distributions, the message collision distributions, and the message omission distributions are almost the same compared to Figure D.1. The only main difference concerns the group spread and the clock adjustment distribution. In detail, Figure D.2c shows that the average group spread of V2 equals about ε/n where ε is the delay jitter of $500\mu s$ and n denotes the network size. This is a typical property of the FTA algorithm. In detail, the FTA approach guarantees an upper bound of the synchronization precision even in the presence of f Byzantine faulty nodes as stated in Equation 4.7. Since we assume that $f = 0$, the group spread should never exceed $\varepsilon + \Gamma$ with Γ be the drift offset as defined in Section 2.2.2. Note that in this case, this worst case precision is independent of the number of nodes. Consequently, an increasing number of message collisions and omissions should hardly affect the precision as long as some few nodes are able to communicate among each other. Note that the aforementioned statement contradicts the results of Figure D.2c. The reason for the increasing precision comes along with a smaller deviation among the clock cycle periods due to a more precise calculation of the average cycle period in the case of an increasing number of nodes. Additionally, since the delay jitter

ε is equally distributed, the averaging function also reduces the influence of ε, if the network size is very large. Unfortunately, the advantage of the better group spread deviation is at the expense of an increased clock adjustment value as presented in Figure D.2f. The reason for the increased clock adjustment lies in the fact that the calculated adjustment value is always smaller than the negated constant message delay of $d = 2.2ms$. For instance, assume two perfectly synchronized nodes with a constant message delay jitter of $d(m) = 2.2ms$ for each message m ($\varepsilon = 0$). In this case, the synchronization information based on the received message from the other node indicates that the clock always loses $2.2ms$.

Algorithm version V3

Prepared simulation results for algorithm V3 are depicted in Figure D.3. The only difference between V3 and V2 concerns the activated Clique Discovery (CD) approach. This approach fastens up the average time to sync through recognizing existing cliques of nodes during the startup phase. This behavior is exactly what can be observed in Figure D.3a. In detail, the median of all simulations is situated at about 25 periods and all distributions have a very small variance compared to V2. As a consequence, we can say that the CD approach speeds up the average convergence time by at least a factor of 2. However, it should be noted that in the case of equidistant initialized nodes (*i.e.*, no cliques are present), then the time to sync of algorithm V3 would be the same as algorithm V2. This case is not covered due to the randomized initialization of the nodes and therefore Figure D.3a does not contain such outliers. The other diagrams show similar results compared to V2, because the Drift Calibration (DC) approach only takes effect until the network achieved synchronicity.

Algorithm version V4

The simulation evaluation of algorithm V4, depicted in Figure D.4, shows promising results regarding a small group spread distribution in combination with small clock adjustments due to the activated MDE approach. In detail, V4 provides a similar group spread distribution as V3 and is typical for the FTA algorithm, but additionally provides adjustment values in the same order as identified in the E-RFA approach as used in algorithm V1, except the fact that the adjustment values are negative. The reason for the improvement of the maximum absolute adjustment value in V4 comes from the fact that a node estimates the minimum message delay. Consequently, the lost time of a received clock value from a neighboring node can be compensated by adding the estimated minimum message delay. As a result, the median of the adjustment value distribution is reduced by exactly the estimated minimum message delay. The presented approach hardly affects the group spread among all nodes as long as all nodes estimate the same minimum message delay and thus add the same time to a received clock value. However, it should be noted that, in the case of a multi-hop network, the amount of time which is used for compensating the lost time should not exceed the lower limit d of the message delay. Otherwise, if the estimated message delay of all nodes is greater than the lower limit (*e.g.*, equals exactly $d + \varepsilon/2$), then the actual delay of some received messages is higher or lower due to the delay jitter. Since the actual message delay of the same message received at different nodes may be different due to the variation in processing time at the receivers, some nodes may re-estimate a new minimum message delay which is higher respectively lower than $d + \varepsilon/2$, even if the additional delay jitter at the receiver is orders of magnitudes smaller than ε. This effect worsens if messages are not received at all nodes due to message collisions or additionally Byzantine faulty nodes are present. The deviation of the re-estimated minimum message delay among the nodes results in the fact that the nodes do not adjust to the same average clock value, even if all nodes are exactly synchronized and the calculated adjustment value before the time compensation is the same at all nodes. This lead

to a vicious circle, because the degraded group spread increases the deviation of the message delay estimation which again degrades the group spread. On this account, the estimated minimum message delay for the time compensation should never exceed the constant part d of the general message delay assumption $d(m) = [d, d+\varepsilon]$. For this, Figure D.4g visualizes the distribution of the minimum and maximum estimated message delay values among all nodes in the time interval $[(t_s+t_e)/2, t_e]$ for each simulation. Therein, we can see that the configured algorithm parameters ensure that the MDE value hardly exceed the lower limit of $2.2ms$, especially if more than 20 nodes are present. Note that if the distributions of the adjustment value diagram is shifted by such a constant time of about $2.2ms$, then the clock adjustment distributions would look similar with respect to the one presented for algorithm V3. Finally note that the difference between the maximum and minimum measured overall clock adjustment value in general does not exceed the delay jitter of $\varepsilon = 500\mu s$, because in the case of no message collisions and omissions, the achievable precision would be upper bounded by $\varepsilon + \Gamma$.

Algorithm version V5

In contrast to algorithm V4, algorithm V5 additionally applies the offset desynchronization scheme which we expect to decrease the average number of message collisions and omissions. For this, Figure D.5 again presents all required simulation results. As we can see in Figure D.5c, the average group spread of V5 almost equals the results from V4. However, the outliers of V4 are lower compared to V5. Similarly, the clock adjustment distributions of V5 are similar to V4. Further, Figure D.5e shows that the number of message collisions equals V5, except the case of the network comprising 20 nodes. In this case, we can observe that no collisions occurred at all, whereas V4 contained some few rounds where the reception of the same single message was disrupted at nearly all nodes. However, the aforementioned narrow differences between V4 and V5 do not justify the effectiveness of the offset desynchronization approach. Nevertheless, if we consider the number of message omissions as shown in Figure D.5f, then we can see that the approach takes effect and the omissions were completely eliminated as long as enough bandwidth was available for the message exchange among all nodes. Otherwise, if the available bandwidth is too small, then the omission rate nearly equals the results from algorithm version V4. For instance, in our case we configured a bandwidth limit of $B = 4ms$. Consequently, in the case $r_{msd}^{max} = 0.2$ (which equals $200ms$, if $T = 1s$), then there is enough bandwidth available for at most $(r_{msd}^{max} - r_{msd}^{min}/B) = 190/4 = 47$ nodes. This fact is exactly what can be observed in Figure D.5f. Therein, the number of message omissions is greater than 0 and additionally increases, if $(r_{msd}^{max} - r_{msd}^{min}/B) < n$, where n is the network size. Summing up, the offset desynchronization approach is an effective and powerful feature that improves the average number of message omissions and consequently the synchronization precision. Note that the other diagrams (*e.g.*, the message delay estimation distribution in Figure D.5g), are very similar to the results of V4. As a consequence, the other parameters are hardly influenced by the offset desynchronization approach.

Algorithm version V6

Compared to algorithm version V4, V6 additionally executes the APM approach which increases the overall network lifetime by letting some nodes passively participate in the synchronization process. All simulation results regarding V6 are prepared in Figure D.6. For this, Figure D.6h shows the maximum measured number of active nodes between the time interval $[(t_s+t_e)/2, t_e]$. Interestingly, after a settling time of about 25 rounds, each simulated network converged to synchronicity where exactly 8 nodes are active and the rest changed to the passive mode, independently of the initially configured network size. Consequently,

all other results are very similar to the simulation result of a network comprising 8 nodes. Since the offset desynchronization approach is disabled in V6, some message omissions are still present due to the random nature of the preponed message transmission. Note that the algorithm can be modified in order to let more than 8 nodes remain in the active state. This can be necessary, if the synchronization precision has to be improved.

Algorithm version V7

Algorithm version V7 finally activates the offset desynchronization in addition to the APM mode as already discussed within the results of algorithm V6. At first view, the simulation results between V7 and V6 as shown in Figure D.7 and Figure D.6, respectively, are almost the same. However, if we consider the number of message omissions as presented in Figure D.7f, then we can see that V7 completely eliminates all omissions in nearly all simulations. This comes from the fact that even in the case of $r_{msd}^{max} = 0.05$, there is enough bandwidth available for at most 22 nodes. Since the number of active nodes is always lower than 22, the offset desynchronization approach greatly takes effect and the message omissions are strongly reduced.

6.6.2 The Faulty Case

Within this section, the same simulations like in the previous section are performed, but with the difference that erroneous nodes are simulated. The prepared evaluations use the same initial configuration as presented in Table 6.3. However, if a network contains f erroneous nodes, then we assume that these nodes belong to the nodes with the configured ids in the range between 0 and $f-1$. Note that this does not mean that these nodes transmit with this id. Instead, the two-faced malicious behavior allows them to transmit with a different randomly chosen id to each distinct neighbor. We further use the abbreviation "f/n simulated network" to mean the simulation of a network comprising n nodes where f nodes are erroneous.

Algorithm version V1 (with disabled DC)

Within the theoretical discussion of the R-RFA algorithm, we stated that this approach no more provides the property of self-stabilization even in a fault-free system, because of the invalidity of the Closure condition as required for Definition 31. However, the Convergence condition still holds with a degraded synchronization precision. An empirical proof of this fact was simply done by simulating a network comprising eight nodes, where one node is simulated to be erroneous in such a way that it simply omits all message transmissions. Since the R-RFA algorithm applies the E-RFA algorithm only on the received synchronization information where the f highest and lowest deviations are excluded, all non-faulty nodes eventually synchronize to the $(f+1)$-fastest node, but not to the fastest node as it would be in the non-faulty case. This is necessary, because the erroneous nodes may simulate different and strongly deviating clock drifts with respect to each individual neighboring node, resulting in the fact that each non-faulty node may synchronize to a completely different phase. Thus, only the $(f+1)$-fastest node is a node where the received phase is definitely situated within the other non-faulty nodes. This leads to the fact that, under certain conditions which are discussed in Chapter 5, the non-faulty nodes eventually achieve synchronicity with respect to the $(f+1)$-fastest non-faulty node with a lower bounded synchronization precision that degrades with an increasing number of erroneous nodes. However, since the f fastest non-faulty nodes eventually overtake the other nodes and are consequently excluded from the synchronization process, these nodes then get away from the synchronized group and, after some time, resynchronize to the rest of the nodes until they get away again. Exactly this

behavior can be observed in Figure 6.3. In order to worsen the effect, we decided to switch off the DC approach, and initially configure the nodes as presented in Table 6.6. Hence, the fastest non-faulty node equals the node with $id = 7$ and periodically resynchronizes to the rest of the group, but always desynchronizes after some time, because in the case of complete network synchronicity the other non-faulty nodes eventually ignore the information received from this node and thus do not adjust their clocks with respect to the fastest non-faulty node any more.

Table 6.6: Initial node configuration for a prepared simulation evaluation of V1 with deactivated DC.

id_i	phase $\varphi_i(0)$	drift rate ρ_i [ppm]	id_i	phase $\varphi_i(0)$	drift rate ρ_i [ppm]
0	600/1000	2	1	600/1000	−2
2	600/1000	3	3	100/1000	−5
4	550/1000	5	5	50/1000	−3
6	500/1000	10	7	0/1000	−10

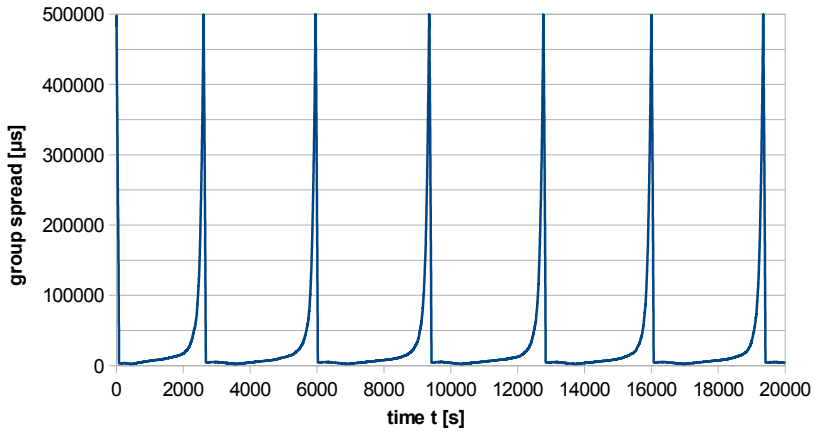


Figure 6.3: Demonstration of the impossibility of maintaining synchronicity in the case of the R-RFA algorithm with $f = 1$ in a fault-free system comprising eight nodes.

Algorithm version V4

In contrast to V1, V4 now validates the required Closure condition through automatically switching from the R-RFA to the FTA approach. As a consequence, V4 should solve the self-stabilizing pulse synchronization problem as formulated in Definition 31. For sake of comparison, Figure D.8 visualizes the distributions of all important quality aspects, based on the prepared simulation results of different network configurations. Similarly to the fault-free case, the main outcome is that the network still does not achieve synchronicity, if the available bandwidth for message communication is too small. For instance, Figure D.8a shows the number of achieved network synchronicities out of 100 simulations, each started with a randomly initialized configuration. From the diagram we can deduce that the configurations 10/71 and 10/100 never achieved synchronicity, because too much faulty nodes were present. These faulty nodes are able to force a transmission even if a transmission of a non-faulty node is currently in process. Consequently, the f faulty nodes are able to continuously occupy a fixed amount of bandwidth and additionally destroy the message transmissions of at most f non-faulty nodes. Similar problems occur if r_{msd}^{max} is set too small. If so, then too less synchronization information will be exchanged and the drift calibration or even the state

synchronization approach will not work at all. This behavior can be observed for smaller values of r_{msd}^{max} in the case of 5/36, 5/71, and 5/100 configured networks. Furthermore, if too less nodes are present ($3f+1 < n < 7f+1$), then the state synchronization will work, but the drift calibration approach cannot take effect even if enough bandwidth is available. This is especially the case for the 1/5 configured network. As a consequence, the time until synchronicity dramatically increases and results in a degraded synchronization precision. Note that in the worst case where all f faulty nodes continuously destroy the synchronization messages of exactly f distinct non-faulty nodes, then the 1/5 configured network will never achieve synchronicity due to the impossibility of convergence for the drift calibration approach.

Algorithm version V5

This section discusses the improvements achieved by enabling the offset desynchronization approach in addition to algorithm version V4. The resulting distributions of several evaluation metrics are presented in Figure D.9 and mostly follow the results from algorithm V4 as discovered in Figure D.8. The only difference can be observed with respect to the measured number of message omissions per round. For example, compared to V4, Figure D.9f shows improvements in the case of 1/8, 1/36, and 1/100 configured networks, if $r_{msd}^{max} \geq 0.15$, because it is ensured that enough bandwidth is available for efficient offset desynchronization. However, the number of message omissions and collisions worsen, if the available bandwidth underruns the required limit, that is, $(r_{msd}^{max} - r_{msd}^{min}/B) < n$. This explains the more frequent occurrence of outliers which are additionally higher than it can be observed in Figure D.9e for algorithm V4. Other quality aspects of V5 are similar to algorithm version V4 and therefore not discussed.

Algorithm version V7

Simulation results regarding algorithm version V7 are presented in Figure D.10. In contrast to version V4, V7 additionally executes the APM feature. Consequently, the number of active nodes should be reduced to at least $7f+2$ and at most $9f+4$ as proven in Theorem 9. This theoretical aspect is practically verified by the simulation results presented in Figure D.10h. Therein, the number of active nodes for the 1/36, 1/71, and 1/100 simulated networks are constantly reduced to 14 (or 13 nodes in the case of $r_{msd}^{max} = 0.1$), as long as enough bandwidth is available (*i.e.*, $r_{msd}^{max} \geq 0.1$). Clearly, the degraded group spread of the simulated networks where the APM approach takes effect comes along with the reduced number of synchronization information a node receives from its neighbors. Note that in the case of a 5/71 or 5/100 simulated network, we would expect a maximum number of 46 active nodes per round. However, Figure D.10h shows that this is not true. The reason for the increased number of active nodes originates from the higher number of message omissions. The same statement applies for the 1/36, 1/71, and 1/100 simulated networks with $r_{msd}^{max} = 0.05$. More important, Figure D.10e and Figure D.10f show a dramatically reduced number of message collisions and omissions, respectively, which comes along with the smaller number of active nodes. Note that the smaller number of active nodes also requires a small number of message collisions and omissions. Therefore, the process of reducing both the number of active nodes and the number of message collisions and omissions are interlinked with each other and depend on a carefully configured maximum message staggering delay for providing enough bandwidth. This leads to the conclusion that the maximum message staggering delay r_{msd}^{max} can be reduced after the nodes recognize that synchronicity is achieved. However, this aspect is beyond the scope of this thesis and may be studied in a future work. To sum up, the APM approach is very effective as long as enough bandwidth is

available, but comes along with a degraded synchronization precision compared to the results if APM is disabled. For instance, in the case of the 1/36 simulated network, Figure D.10c shows that the maximum measured group spread degrades from $40\mu s$ to $100\mu s$. However, in many situations a small duty-cycle is required and, hence, requires a tradeoff between a small bandwidth and a sufficient synchronization precision. Other undiscussed evaluation metrics are similar to algorithm version V5.

Finally, the diagram in Figure D.10i plots the standard deviation of the measured real-time durations of the pulse clock cycle period among all nodes over time t. The simulated network in this case comprises 71 nodes where 10 nodes are Byzantine faulty. Therein, we can see that the higher rate of message omissions prevents the drift calibration algorithm from converging the nodes to a common average cycle period. Consequently, the time until synchronicity dramatically increases and even can be infinite. The reason for this dilemma lies again in the fact that too less bandwidth is available and too much messages are omitted due to the presence of the faulty nodes.

6.7 Simulating Multi-hop Topologies

In contrast to fully connected networks, achieving a network-wide synchronization in multi-hop topologies is a more interesting attempt with respect to real world applications. For instance, many sensor networks are based on a source-to-sink communication topology with a communication path consisting several hops. Therefore, different types of topologies are simulated in dependence of the network size and other parameter choices. The results then should give a feeling about the robustness, achievable precision, and average time to sync for the use in applications which have a similar topology structure.

6.7.1 The Fault-free Case

In a fault-free system, no faulty or erroneous nodes exist by definition. However, message omissions and collisions due to interfering message transmissions are an inherent problem in all wireless communication networks. Since the CD, APM, and offset desynchronization approach are not aimed for the use in multi-hop networks and simulation results have shown that the establishment of these approaches worsened the results in such topologies, we only concentrate on the simulation of the three algorithm variants V1, V2, and V8.

Chain Topologies

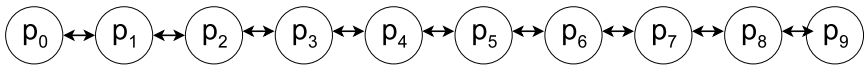


Figure 6.4: Visualization of the communication topology of a chain-structured multi-hop network comprising 10 nodes. The arrows visualize a bidirectional communication link.

This communication topology is the simplest multi-hop topology and is used to evaluate the scalability property of the algorithms with respect to the network size n. Figure 6.4 shows a typical chain topology containing 10 nodes.

Algorithm version V1. The first algorithm version executes the E-RFA approach. For the case of 2 nodes, Theorem 1 states the achievable worst case precision as long as every message is received within the predefined bounds. However, since in the worst case, every pair of nodes over each hop cannot synchronize better than this bound, we expect an achievable

worst case precision of a chain network comprising n nodes to be about $(n-1)\cdot\Pi^U$ with Π^U from Theorem 1. Generally, we can say that in the worst case, if a network contains at most h hops, then the E-RFA approach cannot synchronize better than $h\cdot\Pi^U$ with Π^U from Theorem 1.

Figure D.11 presents the simulation results of chain networks with a network size of $n=5$, $n=10$, $n=20$, and $n=30$ nodes. Based on the aforementioned precision estimation, we expect a worst case precision of $\Pi=(n-1)\cdot\Pi^U$ with $\Pi^U=2.94ms$ for $r_{max}=0.2$, $\varepsilon=0.5ms$, and $d=2.2ms$. Consequently, we get $\Pi=11.76ms$, $\Pi=26.5ms$, $\Pi=55.87$, and $\Pi=85.28ms$ for $n=5$, $n=10$, $n=20$, and $n=30$, respectively. This is exactly what we observe in Figure D.11c. We see that the worst case group spread comes in the vicinity of these bounds, but never exceeds them. However, this heavily depends on the number of message omissions and collisions which is very low due to the low number of neighboring nodes (at most 2 neighbors) as presented in Figure D.11e and Figure D.11f. The major strengths of the E-RFA approach, however, is the fact that a node adjusts its clock only forward and only by a small amount as it can be seen in Figure D.11d. Thus, in the worst case, the nodes usually adjust their clocks in the order of some few microseconds but with the disadvantage that two neighboring nodes are worse synchronized. Figure D.11a further shows that nearly all randomized simulation evaluations achieved a synchronized pulse state within 1000 rounds with an average time to sync in the order of some hundred rounds as presented in Figure D.11b. Note that further practical tests have shown that the few simulations that did not achieve a synchronized pulse state within 1000 rounds achieved, achieved this state at most after 2000 rounds.

Algorithm version V2. The simulation results of the FTA-RFA approach without message delay estimation as presented in Figure D.12 are closely related with the results of the same approach applied in fully connected networks. For instance, the precision is much better compared to V1 due to the fact that the constant part of the message delay is no more included in the worst case precision. This comes along with the high negative clock adjustments as shown in Figure D.12d. Based on Equation 4.7, the worst case precision in the case of FTA applied on topologies containing at most h hops is expected to be $\Pi=h\cdot(\varepsilon+\Gamma)$. Thus, if we assume that $\varepsilon=0.5ms$, then we expect a worst case precision of $\Pi=2.8ms$, $\Pi=6.3ms$, $\Pi=13.3ms$, and $\Pi=20.3ms$ for $n=5$, $n=10$, $n=20$, and $n=30$, respectively. Interestingly, all simulated networks with a configured maximum relative message staggering delay of $r_{msd}^{max}=0.2$ did not exceed this theoretical bound. However, a lower r_{msd}^{max} clearly increases the number of message omissions and collisions and consequently also the worst case group spread among the nodes as shown in Figure D.12c. As anticipated, the number of maximum message collisions and omissions per round are similar to V1. Finally, similar to V1, at most all simulated systems with a random initial configuration achieved a synchronized pulse state within 1000 rounds. The remaining simulation that did not achieve this state, achieved this state some rounds later.

Algorithm version V8. Simulation results regarding algorithm version V8 are presented in Figure D.13. Therein, Figure D.13g shows that the MDE approach takes effect and every node estimates a minimum message delay which is not very accurate, but mostly situated between $2.2ms$ and $2.7ms$ in the case of $r_{msd}^{max}=0.3$. The inaccuracy comes from the fact the a node has at most 2 neighboring nodes for the estimation process. In contrast, the grouped multi-hop approach should eliminate this problem as presented in the next section. The main advantage of the MDE approach is that the maximum clock adjustment values are reduced by about this MDE value and is visualized in Figure D.13d. Interestingly, the average and worst case group spread is almost a little bit better than the corresponding results

from algorithm V2, except the case of $n = 30$ nodes. The number of achieved network synchronicities is a little bit worse compared to V2, especially for the case of $n = 10$ and $n = 20$. However, the systems which did not achieve synchronicity within 1000 rounds still entered the synchronized pulse state some rounds later. Finally, the time to sync distribution is almost similar to that from V2.

Grouped Multi-hop Topologies

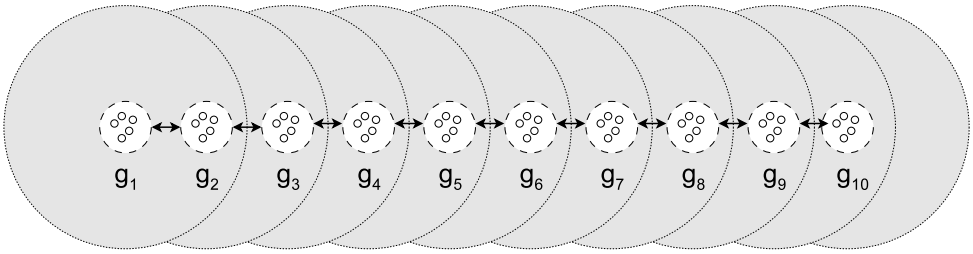


Figure 6.5: Visualization of the communication topology of a grouped multi-hop network with a group size of $g = 5$ and 10 groups. The shaded area demonstrates the transmission range of a complete group.

A grouped multi-hop topology is similar to a chain topology, except that the nodes are replaced by groups comprising several nodes. Whereas the nodes in such a group are fully connected, the groups are connected in a chain-like way. Further, all nodes in a group have a bidirectional communication link to all nodes in the immediate neighboring groups. Figure 6.5 demonstrates such a topology with a group size of $g = 5$ and 9 hops. Note that all simulated grouped multi-hop systems comprise exactly 10 groups and thus suffer from at most 9 hops. Simulation evaluations are performed with different group sizes and parameter choices.

Algorithm version V1. Simulation results regarding this algorithm version are presented in Figure D.14. All simulated systems entered the synchronized pulse state within 1000 rounds and have an average time to sync of about 200 rounds. Interestingly, the group spread distribution strongly deviates among all simulations and therefore seems to be independent of the group size. This comes from the fact that the E-RFA approach cannot synchronize better than at least the maximum message delay. If we assume that $\rho = 100\text{ppm}$, $\varepsilon = 0.5ms$, $d = 2.2ms$, and $r_{msd}^{max} = 0.05$, then the resulting worst case precision is greater than $\Pi = 9 \cdot 2.9ms > 26ms$. Figure D.14c shows that this upper precision bound is never exceeded, even in the presence of a high number of message omissions and collisions. The inherent advantage of the E-RFA algorithm again corresponds to the small clock adjustment values as demonstrated in Figure D.14d.

Algorithm version V2. Figure D.15 contains the simulation results of the FTA-RFA approach. The general outcome of this simulation evaluation is that an increasing group size slightly improves the average group spread and thus also the real-time precision. This comes from the fact that a node has more neighboring nodes and is able to perform a more accurate estimation of the minimum message delay. Furthermore, a higher number of neighboring nodes additionally improves the behavior of the drift calibration approach. Compared to V1, the number of achieved network synchronicities within 1000 rounds and the average time to sync of 200 rounds for all simulated systems is very similar. Furthermore, the average group spread among all simulated systems dramatically improved to about $1.5ms$ due to the

FTA approach. In addition to that, the maximum measured group spread never exceeded the theoretical worst case precision for the FTA approach of $\Pi = 7ms$. The only disadvantage is again the high negative average clock adjustment as presented in Figure D.15d which is, as expected, in the order of the minimum message delay. To sum up, the FTA-RFA approach seems to be very robust even in the presence of high message omissions and collisions.

Algorithm version V8. Simulation results are presented in Figure D.16 and are very similar to the results of V2 except the fact that the average group spread among all simulations improved a bit to about $0.5ms$. Furthermore, the nodes performed an accurate estimation of the message delay for $g \geq 3$ and $r_{msd}^{max} >= 0.1$ (Figure D.16g). Compared to V2, this leads to a reduction of the adjustment value by about $2ms$. The average time to sync and the message omission and collision distributions are the same as evaluated for V2.

Regular Grid Topologies

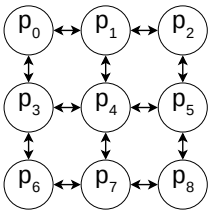


Figure 6.6: Visualization of the communication topology of a regular grid-structured multi-hop network with a network size of $3x3$. The arrows visualize a bidirectional communication link.

A typical regular grid topology containing 9 nodes ($3x3$ grid) is visualized in Figure 6.6. Simulation results regarding this topology type should give a feeling about the size scalability property in more than one dimension (as it is in chain topologies). Furthermore, such a topology may be of more practical interest than the previous ones.

Note that every regular grid topology contains at least one maximum BRC C_{max} with $l(C_{max}) = 4$. On this account, the inequality of Theorem 8 should be valid in order to reduce the probability of existing stable infeasible firing configurations. For instance, the resulting lower bound for L according to this inequality with $\alpha = 1.01$, $d_{max} = 2.7$ms, and $n = 4$ would be $L > 92.96$. Furthermore, the upper bound for L according to Theorem 7 with $d_{max} = 2.7$ms and $n = 4$ (length of the maximum BRC C_{max} in a regular grid topology) is $L < 92.6$. Clearly, this reveals that the algorithm is not very scalable, if the network contains BRCs, because in this case it is impossible to eliminate the occurrence of unwanted stable system states. However, a remedy can be the appropriate reconfiguration of the other parameters. In detail, the lower bound for L according to Theorem 8 with $\alpha = 1.02$, $d_{max} = 2.7$ms, and $n = 4$ now equals $L > 9.5$. Note that all simulations were performed with the same parameter settings as used for the other topologies and thus may contain system states that never converge to a synchronized pulse state.

Algorithm version V1. Simulation evaluations based on algorithm V1 are presented in Figure D.17. Nearly all randomized simulations achieved synchronicity within 1000 periods as shown in Figure D.17a. The remaining simulations still achieved a synchronized pulse state some rounds later. Note that the theoretical upper bound for the worst case precision in the case of no message loss is about $\Pi = 2 \cdot 2.9ms = 5.8ms$, $\Pi = 4 \cdot 2.9ms = 11.6ms$, $\Pi = 8 \cdot 2.9ms = 23ms$, and $\Pi = 18 \cdot 2.9ms = 52ms$ for a 2x2 grid, 3x3 grid, 5x5 grid, and a

10x10 grid, respectively. All simulated systems never exceeded this bound as shown in Figure D.17c. Therefore, we can deduce that the algorithm is very robust and provides graceful degradation in the case the number of message omissions and collisions is increasing. Interestingly, the group spread diagram shows that the average group spread sometimes improves with a smaller maximum relative message staggering delay r_{msd}^{max} within the same network size. This is especially the case, if the message loss does not dramatically increase with a decreasing r_{msd}^{max}. Lat but not least, the clock adjustments are again very small which comes along with a degraded precision.

Algorithm version V2. The FTA-RFA approach again shows its strength in the group spread diagram of Figure D.18 and provides an improved worst case group spread of at least a factor of 2 compared to V1. Furthermore, nearly all simulated systems with $r_{msd}^{max} = 0.2$ (except the 2x2 grid) maintained the theoretical worst case bound of $\Pi = 2(k-1) \cdot 0.7ms$ for a *k*x*k* grid network. As expected, the clock adjustments are in the order of the message delay. Unfortunately, only 70 percent of the randomized simulations with a 10x10 grid structure and $r_{msd}^{max} = 0.2$ achieved a synchronized pulse state. Further experiments have shown that the remaining 30 simulations never achieved at all till 10000 rounds. This may come from the fact that the inequality of Theorem 8 is not valid in the case of $\alpha = 1.01$ and $L = 4$.

Algorithm version V8. The main difference to V2 concerns the number of syncs diagram and the group spread diagram in combination with the reduced clock adjustment diagram in Figure D.19. In detail, in contrast to V2, it seems that the probability of algorithm version V8 for achieving a synchronized pulse state is higher, especially for larger systems. The MDE approach again takes effect and additionally improved the group spread by a factor of 2 compared to V2. Furthermore, the average clock adjustments are reduced by about $1.5ms$ which results in a worst case adjustment in the order of some few microseconds. The diagram in Figure D.19g further shows that the 2x2 grid network with $r_{msd}^{max} = 0.2$ estimated the minimum message delay most accurately with the smallest variance.

Ring Topologies

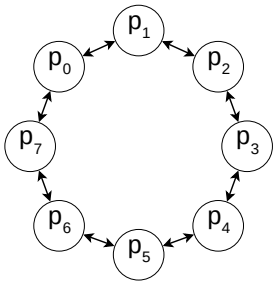


Figure 6.7: Visualization of the communication topology of a ring-structured multi-hop network with a ring size of $n = 8$. The arrows visualize a bidirectional communication link.

The ring topology (Figure 6.7) was simulated in order to evaluate the scalability property of the algorithms with respect to an increasing BRC and to practically test the correctness of Theorem 8. Based on this theorem, Table 6.7 lists the required bounds for the coupling factor α and the FTA convergence threshold L in order to reduce the probability of entering an infeasible firing state. As a consequence, we can say that the FTA-RFA algorithm is worse with respect to the size scalability in the dimension of an increasing BRC. However,

FTA-RFA is still scalable in flat topologies that contain only small BRCs. In contrast, RFA works well independent of the network size, but may require a long time for convergence.

Figure 6.8 provides a more detail insight in the devised lower bound theorem for L. It visualizes the lower bound in dependence of the coupling factor α and the worst case message delay $d_{max} = d + \varepsilon$ for different sized networks. Note that a negative lower bound does not mean that any L is possible. In contrast, this means that the inequation of Theorem 8 never holds and the possibility of stable infeasible firing states cannot be eliminated. In the case of $n = 5$, Figure 6.8a shows that there is enough space for choosing an appropriate value for L. However, for $n \geq 10$ it is not possible to chose an L which is smaller than 100 with $d_{max} = 2.7ms$ (as assumed in our system model) and any α. This demonstrated the impossibility of FTA-RFA to provide scalability in communication that suffer from high message delays and additional contain BRCs.

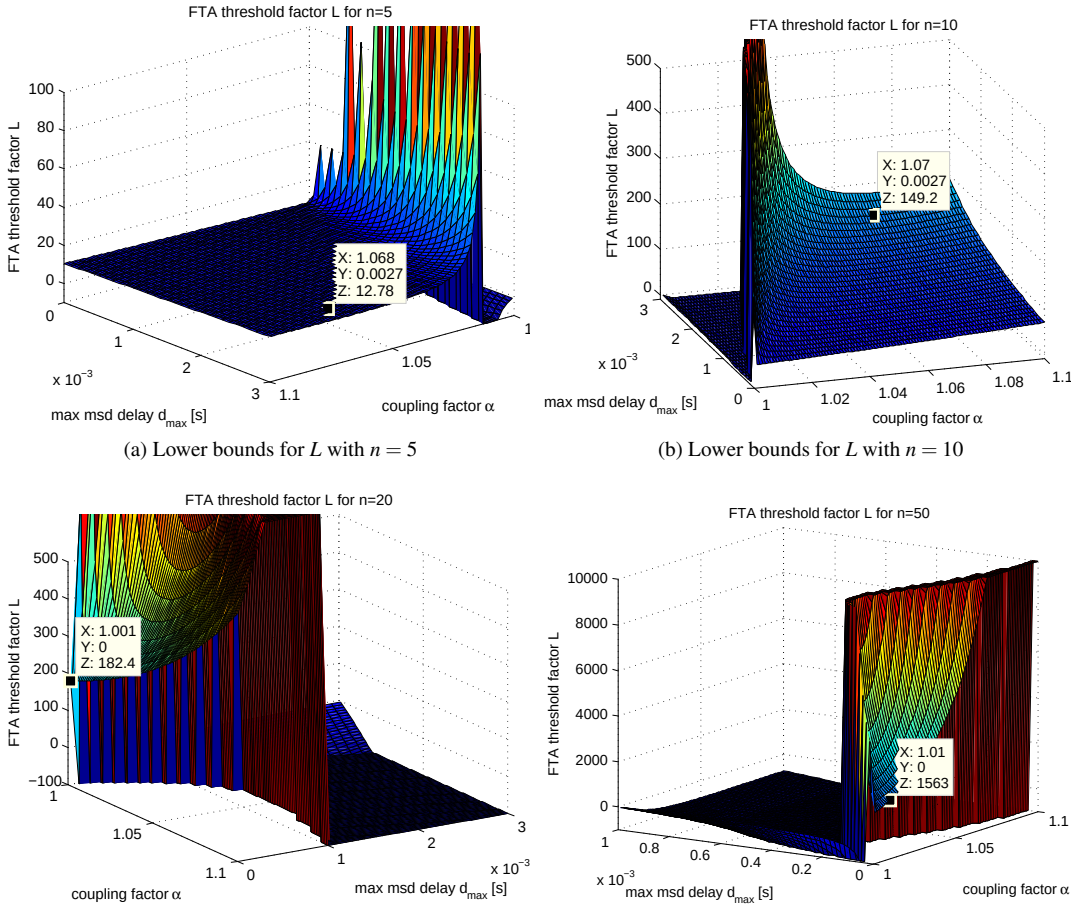
Table 6.7: Calculated bounds for the coupling factor α and the convergence threshold L in ring topologies.

Ring size n	Theorem	Assumption	5	10	20	50
Upper bound for α	Theorem 4	$d_{max} = 2.7$ms, $\rho = 100$ppm	1.240	1.107	1.049	1.017
Upper bound for L	Theorem 7	$d_{max} = 2.7$ms	74	37	18	7
Lower bound for L	Theorem 8	$\alpha = 1.07$, $d_{max} = 2.7$ms	13	150	∞	∞
Lower bound for L	Theorem 8	$\alpha = 1.01$, $d_{max} = 0$ms	9	43	201	1564

Algorithm version V1. The number of syncs diagram in Figure D.20 shows that the E-RFA approach did not always achieve synchronicity within 5000 rounds, especially in the case of big-sized rings. One reason may be the increased message loss. However, we were not able to completely identify the reason for this problem and belongs to future research. Generally, we can say that the average time to sync is about $20 \cdot n$ rounds. Furthermore, in the case network synchronicity is achieved, the worst case group spread of $\Pi = n \cdot 2.9ms$ is always maintained, even in the case of high message collisions and omissions. The group spread distribution of simulations with big-sized rings comprising 50 nodes and $r_{msd}^{max} \leq 0.1$ that never achieved synchronicity are also included in Figure D.20c. Interestingly, these systems never synchronized better than $200ms$ and therefore were not able to achieve synchronicity. The main reason behind this problem highly likely belongs to the increased message collisions and omissions in this case. Finally, the main advantage of the E-RFA algorithm again concerns to the very low clock adjustments as presented in Figure D.20d.

Algorithm version V2. Results regarding V2 are presented in Figure D.21. Compared to V1, V2 provides a decreased percentage of achieved network synchronicities, especially for the case of $n = 50$ nodes. The reason for this is twofold. First, similar to V1 the increasing message loss prohibits the network from entering a synchronized pulse state. In addition to that, the invalidity of the inequation of Theorem 8 is also responsible for reduced number of network syncs. The worst case group spread for $r_{msd}^{max} = 0.2$ improved by at least a factor of 2. Similar to the previous simulations, FTA-RFA suffers from high average clock adjustments in the order of $2ms$. The remaining diagrams are almost similar to V1.

Algorithm version V8. The main difference of V8 with respect to V2 concerns the clock adjustment diagram of Figure D.22. In detail, the MDE approach approximately estimates the message delay and leads to a reduction of the clock adjustments by about $1.5ms$. Note that in the case of small-sized rings in combination with a high maximum message staggering delay, the network estimated the message delay more accurately than big-sized rings.



(a) Lower bounds for L with $n = 5$

(b) Lower bounds for L with $n = 10$

(c) Lower bounds for L with $n = 20$

(d) Lower bounds for L with $n = 50$

Figure 6.8: Lower bounds for the FTA convergence threshold L in ring topologies according to Theorem 8. (X,Y,and Z correspond to the coupling factor α, the worst case message delay d_{max}, and the resulting lower bound for L, respectively.)

Random Geometric Topologies

The simulation of a random geometric topology is of major practical interest and demonstrates the robustness of the applied algorithms in the case of randomly connected networks. In our case, we scattered always 100 nodes on a 100x100 square meter field. Since the communication range of single node is at most 20 meters, this should ensure a high probability of a connected communication network.

Algorithm version V1. From Figure D.23, we can deduce that the V1 algorithm applied in a random geometric topology as described is very powerful. In detail 99 percent of the performed randomized simulations achieved synchronicity within 1000 rounds and required an average time to sync of about 250 rounds. Note that the theoretical upper bound for the worst case precision in such a network equals $\Pi = 2 \cdot (100/25 - 1) \cdot 2.9ms > 17ms$. This comes from the fact that in the worst case, a connected network in such a two dimensional square field comprising 100 nodes, each having a transmission range of at most 25 meters, contains at most $2 \cdot (100/25 - 1)$ hops and corresponds to a 4x4 grid network. The group spread diagram of Figure D.23d shows that this bound is never exceeded. The clock adjustments are again very small which is typical for the RFA approach and comes along with a degraded precision.

Algorithm version V2. Most simulations results for V2 as presented in Figure D.24 are similar to V1. Clearly, the group spread distributions are better due to the use of the FTA approach for maintaining the clocks synchronized. The simulations again hardly exceeded the theoretical upper bound for the worst case precision in the absence of message loss of $\Pi = 2 \cdot (100/25 - 1) \cdot 0.7ms > 4.2ms$. Only the system configured with $r_{msd}^{max} = 0.1$ contained some few outliers exceeding this threshold. This may result from the fact that the randomly initiated communication topology reacted more sensitive to message loss. Note that the average clock adjustment of $2.5ms$ also equals the average group spread.

Algorithm version V8. The improvements of V8 mainly concerns the reduction of the average clock adjustment by about $2ms$ as shown in Figure D.25. Consequently, the nodes mostly adjusted their clocks in the order of the delay jitter ε. Generally, the group spread distributions also improved a little by about $1ms$ which is traced back to the decreased clock adjustment value and the more or less accurate estimation of the minimum message delay in the order of $2.25ms$.

6.7.2 The Faulty Case

In the faulty case, we assume the existence of erroneous nodes in a fault-free system. Note that this is in contrast to a coherent system where Byzantine faulty nodes that behave in an adversary manner are allowed and therefore theoretically never converge as proven in the previous chapter. In our case, the erroneous only transmit different random values to the distinct neighbors, but do not behave in an adversary manner. Simulation results based on such a system yield estimations about the robustness property of the applied algorithm.

Grouped Multi-hop Topologies

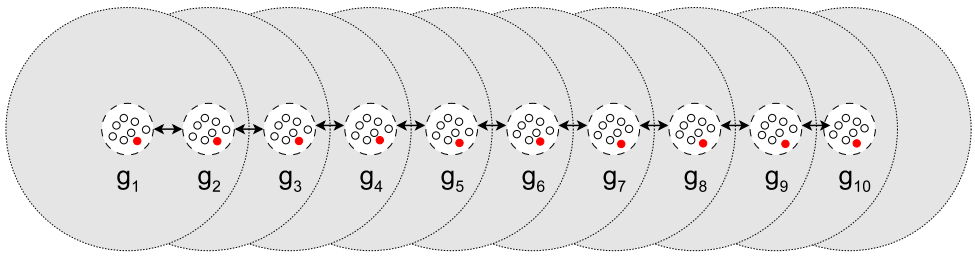


Figure 6.9: Visualization of the communication topology of a grouped multi-hop network with a group size of $g = 8$, 10 groups, and each group containing one erroneous node. The shaded area demonstrates the transmission range of a complete group. The arrows visualize a bidirectional communication link.

The grouped multi-hop topology with a group size g is the simplest topology that maintains g-connectivity. That is, there exist at least g different communication paths between any two nodes. This is a major requirement in the case an algorithm has to provide resilience against Byzantine nodes. In detail, we require that the group size $g \geq 7f + 1$ in order to correctly execute the fault-tolerant drift calibration algorithm in the presence of unpredictable message loss. Furthermore, the R-RFA approach additionally requires at least a $(5f + 1)$-connected network as discussed in the previous chapter. Figure 6.9 presents a schematic structure of such a grouped multi-hop topology with a group size of $g = 8$. Note that the main advantage of the grouped multi-hop network is that it allows the presence of at most f_g faulty nodes in each group. In our case, we assume $f = 1$ and the faulty nodes are highlighted with red filled circles. In contrast, whereas our definition of a grouped multi-hop

network comprising k groups allows the presence of at most $k \cdot f_g$ erroneous nodes (or f_g Byzantine nodes after convergence) with the constraint of at most f_g faulty nodes in each group, general $(7f+1)$-connected networks are resilient to at most f faulty nodes.

However, note that we assume that each node knows the exact number of neighboring faulty nodes. This means that in our topology example of one faulty node in each group ($f_g = 1$), all nodes of the border groups assume $f = 2$ and the remaining nodes assume $f = 3$. This is important, since otherwise, if all nodes would assume $f = 3$, then all nodes of the border groups may never converge to the remaining nodes if $g = 7f + 1$, because they may ignore all information received from the neighboring group. Note that it is still possible to configure all nodes with the same number of faulty nodes $f = 3f_g$. However, if so then the group size must satisfy $g \geq 18f_g + 1$. This comes from the fact that in the worst case, at least one node of each neighboring group must be included in the calculation process. That is, assume a group g_k having two neighboring groups, each containing at most f_g faulty nodes. Thus, a node of g_k will always exclude the $2 \cdot (3f_g)$ lowest and $2 \cdot (3f_g)$ highest deviations as stated in the drift calibration algorithm. However, the $3f_g$ faulty nodes may always corrupt the message transmission of $3f_g$ other correct nodes and thus a node will never receiver messages from more than $(3g-1)-6f_g$ neighboring nodes. This node further excludes the $6f_g$ highest and $6f_g$ lowest deviations from these received messages leading to at most $(3g-1-18f)$ remaining values for the calculation process. Since at least one remaining value is required from each neighboring group in order to guarantee convergence, the inequality $(3g-1-18f_g) > 2f_g + 1$ ensures this due to the pigeon hole principle. Thus, in the case every node assumes the same $f = 3f_g$, the group size must be at least $g \geq 18f_g + 1$.

Algorithm version V2. The simulation results of algorithm version V2 in a $f/g = 1/8$ grouped multi-hop topology as visualized in Figure 6.9 are presented in Figure D.26. The behavior of this algorithm for the two system configurations with $r_{msd}^{max} = 0.3$ and $r_{msd}^{max} = 0.2$ is very similar. Only the system with $r_{msd}^{max} = 0.1$ presented worse results compared to the other ones, because only 24 out of 100 simulations achieved synchronicity due to the increased number of message omissions. The average time to sync for the two other systems is about 250 rounds. The worst case groups spread never exceeded $2.5ms$. Note that the theoretical upper bound for the worst case precision in this situation equals $\Pi = 9 \cdot 0.7ms > 6ms$. Thus, we can say that both the drift calibration and the FTA approach work well in such a topology. The average clock adjustment also equals about $2.5ms$.

Algorithm version V8. For sake of comparison, algorithm version V8 is applied on the same grouped multi-hop topology as used for V2. The resulting distribution diagrams are visualized in Figure D.27. Therein, we can deduce that the activated MDE additionally improves the group spread by a factor of 2 compared to V2. In detail, for the case of $r_{msd}^{max} = 0.3$, the nodes never deviated more than $1.2ms$. All randomized simulations for $r_{msd}^{max} = 0.3$ and $r_{msd}^{max} = 0.2$ achieved network synchronicity within 1000 rounds with an average time to sync of about 250 rounds. A further important improvement due to the MDE approach belongs to the very small clock adjustments which are mostly in the order of $0.25ms$. This means, that the MDE approach reduced the average clock adjustments by about $2.2ms$ which is exactly the minimum message delay. Figure D.27f further shows that the MDE works effectively, because in average the nodes estimate the minimum message delay with $2.25ms$ and only deviates by $50\mu s$ from the correct real minimum message delay.

CHAPTER 7

Discussion

In this thesis, a robust and efficient pulse synchronization algorithm that satisfies the convergence condition in fault-free systems and additionally maintains synchronicity by the use of FTA even in coherent systems is presented. FTA-RFA, presented in Section 5.3.1, comprises several topology-dependent mechanisms that improve different quality aspects.

For example, simulation results have shown that the FTA-RFA approach in combination with the MDE feature of Section 5.4.2 is very robust and works well in a variety of topologies. Furthermore, the combination with the FTA approach provides a much better precision than the E-RFA will ever achieve. This comes from the fact that, in the case of synchronized nodes, FTA adjusts the clock by the average message delay and thus provides an upper bound of the worst case precision which is in the order of the delay jitter ε. However, this precision improvement comes along with high clock adjustments in the order of the message delay. The E-RFA or R-RFA approach proposed in Section 5.2, in turn, perform only very small clock adjustments in the order of some few microseconds. Nevertheless, the main disadvantage of the RFA approach is the fact that it cannot synchronize two nodes better than at least the message delay as proved in Theorem 1. Thus, a logical consequence was to combine the advantages of both the RFA approach and a convergence-averaging mechanism in order to provide self-stabilization and an improved worst case precision, respectively. So the advantages of the FTA-RFA with MDE are twofold. First, it overcomes the problem of the worse synchronization precision resulting from the E-RFA algorithm through dynamically switching to the FTA approach. Second, the MDE feature establishes a distributed message delay estimation which allows the nodes to efficiently reduce the average clock adjustments. In addition to that, simulation results have shown that the MDE further improves the worst case and the average group spread by a factor of two as long as the number of message omissions and collisions are very low. To sum up, the FTA-RFA algorithm in combination with MDE seems to be a powerful and robust synchronization primitive with a low message complexity for pure internal pulse synchronization of low-cost sensor nodes in nearly all connected networks where no external time sources are available and the nodes additionally are suffering from high clock drift but low drift variation with respect to the resynchronization period.

Beside the FTA-RFA approach, an efficient and fault-tolerant drift calibration approach (Section 5.3.2), which is resilient against at most f Byzantine nodes, was developed. This algorithm can be executed in parallel to the aforegoing discussed state correction mechanism in any $(7f+1)$-connected coherent system without an increase in the message complexity. The main advantage of our drift calibration scheme is the fact that it does not suffer from a common-mode drift and reaches approximate agreement through distributed fault-tolerant

average consensus in the absence of an external time server. As a result, in combination with a state correction algorithm like FTA-RFA, the system achieves synchronicity with a high precision, but the synchronization period may strongly deviate from real-time. This is exactly the reason why the algorithm achieves synchronicity rather than time synchronization. However, in many situations, a precise internal synchronization is more important than a high accuracy with respect to, *e.g.*, the UTC time base.

Theoretical studies of the RFA approach have shown that every connected system containing more than two nodes eventually converges to one out of 2^n different stable fixpoints which correspond to a synchronized pulse state. Consequently, the state space contains an infinite amount of possible fixpoint states that the system theoretically will never exit in the perfect case. However, the assumption of practical inaccuracies (*e.g.*, calculation inaccuracies, delay jitter) usually guarantees that the system is eventually pushed out of such an unwanted fixpoint state as long it is a repeller and the calculation granularity is high enough. Note that exiting such a state in the perfect case equals the symmetry breaking problem, which is impossible to solve in an anonymous system. Unfortunately, beside the aforementioned metastable states, many systems still consist of stable states that do not correspond to the synchronized pulse state and, therefore, the system will never exit even in the case of high inaccuracies and delay jitter. On this account, upper and lower bounds for all parameters in the fault-free case were elaborated. In practice, this should dramatically reduce the existence of such unwanted stable states.

However, the main outcome of a theoretical analysis is that the FTA-RFA approach is not very scalable with respect to an increasing BRC. As a result, FTA-RFA scales well in all connected networks as long as the maximum message delay and the maximum BRC is very small. As a consequence, FTA-RFA does not scale well in ring topologies. In contrast, the RFA approach without FTA scales well in rings, but provides a worse synchronization precision. Beside this problem, it is nearly impossible to devise theoretical bounds which cover all possible unwanted stable states. This is exactly what practical evaluations have shown: Systems that maintained the devised theoretical parameter bounds have shown a very high probability of convergence. However, in some few cases, a randomly initialized system was not able to converge or required a very long time to sync in the order of several thousands of rounds. In most cases, a high number of message collisions and omissions was the reason for the impossibility of convergence.

Fortunately, real WSNs usually do not behave like ring topologies. In contrast, the nodes are randomly scattered. Therefore, practical evaluations with random networks have shown that FTA-RFA is still scalable in random geometric networks, because the probability of big BRCs is very small in such topologies. This is exactly what is known as the small-world phenomenon [WC03, p. 13]. Therein, Wang *et al*. state that the ability to achieve synchronization in a large-sized nearest neighbor coupled network, as it is typical for ring topologies, can be drastically improved by adding some few nodes that connect far away nodes together. As a consequence, the network evolves to a small-world model. In our case, this means that the BRC of length n in a ring topology comprising n nodes is dramatically reduced and, consequently, the probability of achieving synchronicity is much higher.

Last but not least, our approach is in the class of self-organizing systems as formally described in Definition 4 due to the fact that the algorithm is self-stabilizing. In detail, according to Definition 1, let S_γ be the set of all possible initial configurations for a connected network γ. Thus, Γ denotes the set of all possible initial configurations for all possible connected network structures. Let the performance function $P(S_\gamma)$ be the probability that the system achieves a synchronized pulse state after some time. In the case of the assumption that the system always exits an unwanted metastable state, then W should be 1. However, in many cases, it is sufficient that the probability of achieving synchronicity exceeds some threshold

p_t. This is exactly what was empirically analyzed by simulation such that a statement can be made about $P(S_\gamma) \in W$ with $W = [p_t, 1]$. In other words, the algorithm performs acceptable well with every source in the family $\{S_\gamma\}$, $\gamma \in \Gamma$, for $W = [p_t, 1]$. Beside the adaptivity property, the system executing our algorithm further adapts by changing its structure as long as the system is fault-free (as assumed for guaranteeing convergence). Finally, since the algorithm is distributed, it does not employ any centralized control.

CHAPTER 8

Conclusion

The main contributions of this thesis are manifold: Based on an intensive research about existing clock synchronization protocols for WSNs, we applied a classification taxonomy in order to compare some selected approaches. This comparison revealed important requirements that a perfect synchronization protocol should satisfy. These are: Self-organization, asymmetric global synchronization, high synchronization precision, robustness against Byzantine attacks, scalability with respect to network size, low message complexity, small-sized messages, low computational complexity, and anonymity. The RFA [WATP+05] turned out to be an appropriate algorithm maintaining most of the aforegoing identified requirements and therefore was taken as a groundwork for our ongoing research. Within this thesis, we extended and devised new approaches for this algorithm in order to overcome the disadvantages of RFA. A formal analysis and an evaluation by means of simulation provided promising results regarding the robustness aspect and the average achievable synchronization precision. The main contributions with respect to the devised objectives in the beginning of this work are briefly reviewed in the following sections.

8.1 Fault-tolerant Clock Synchronization in WSNs

This work presents a meticulous research about existing work in the area of robust and fault-tolerant clock synchronization protocols in WSNs. A comparison among the protocols with respect to a classification taxonomy lead to the result that only few algorithms provide resilience against Byzantine nodes and that many approaches suffer from a high message complexity which is inappropriate in the case of wireless communication. Beside traditional clock synchronization, this thesis concentrates on a more interesting basic synchronization approach named pulse synchronization. Tremendous research with respect to pulse synchronization exists in the area of wired distributed systems. However, in the case of wireless communication only the RFA approach was found to be a practically usable algorithm for self-stabilizing pulse synchronization. Furthermore, in contrast to other synchronization protocols, RFA is based on simple calculations and is aimed at establishing a network-wide (global) synchronization in an ad-hoc multi-hop network without the requirement of structure (*e.g.*, clustered network). Within this thesis, an extended and more robust version of RFA is devised. This algorithm reduces several problems of the original RFA approach (*i.e.*, a long exponential convergence time, a worse achievable synchronization precision, and strongly degrading synchronization qualities in the presence of Byzantine attacks).

8.2 Robust Self-organizing Synchronization

Throughout this thesis E-RFA, an improved version of RFA, is presented. E-RFA provides a bit better synchronization precision through introducing a short refractory period. Within this period, a node does not react to the firing events of other nodes anymore. A robust version of E-RFA, namely R-RFA, is devised in Section 5.3.1. It is based on a fault acceptance mechanism as used in the FTA synchronization algorithm. Therefore, in the presence of f erroneous nodes, R-RFA calculates the new phase advance according to the remaining received clock values after excluding the f lowest and f highest clock deviations. Note that R-RFA only provides robustness and not fault tolerance. That is, it practically converges with a very high probability in the presence of at most f erroneous nodes. In contrast to Byzantine nodes, erroneous nodes do not collude or behave in an adversary manner, but are still capable of transmitting different randomly chosen incorrect values to the distinct neighbors. A further important assumption in our system model is that a faulty or erroneous node does not transmit more than one message per round and additionally is able to corrupt the broadcast of at most an other node through an intentional transmission at the same time. This assumption was necessary to prohibit jamming attacks which are impossible to defeat in the case redundant communication channels are not available.

This thesis proves that, in the case of R-RFA, Byzantine nodes which behave in an adversary manner are always able to prevent the system from converging to a synchronized pulse state. Unfortunately, in the presence of clock drift and $f > 0$ erroneous nodes, R-RFA is able to practically converge to a synchronized pulse state, but is not able to maintain this state. Theoretical results and simulation evaluations have shown that a combination of both the RFA and the FTA approach resolves this problem and provides a robust and more precise pulse synchronization. In detail, whereas the R-RFA approach is used for ensuring convergence in a fault-free system, the synchronized pulse state is maintained through dynamically switching to the FTA algorithm and thus ensures the closure condition in a coherent system comprising at most f Byzantine nodes.

Beside this basic synchronization scheme, some additional optional features in dependence of the network topology are presented. For instance, the MDE feature enables the nodes to establish a distribute estimation of the average message delay in the overall system. Consequently, the nodes are able to reduce the clock adjustments by this amount. This is an important improvement in the case the minimum message delay is very high and FTA is executed, because without MDE the nodes in average adjust the amount of the message delay in every round. Whereas the FTA-RFA approach in combination with the MDE feature is feasible in any $(5f + 1)$-connected multi-hop networks, an additional feature named CD is presented solely for the use in single-hop networks. In detail, this feature allows the node to continuously discover the existence of cliques in a single-hop network. If so, then the nodes instantaneously adjust their clock to the average of this clique instead of performing several phase adjustments due to the RFA part of FTA-RFA until the clique is entered. In the average case, this dramatically reduces the convergence time as it would be without CD in single-hop networks. A further important feature applicable in single-hop networks is the APM mechanism. This feature dynamically switches a desired amount of nodes into a passive mode in a self-organized manner. In this mode, a node does not actively participate in the synchronization process anymore. In contrast, such a node only listens to the medium for new synchronization messages and adapts its clock according to this information. As a consequence, the system's life time can be increased. Last but not least, the offset desynchronization approach is a special feature which allows the nodes to select an offset value in a self-organized way for the preponed transmission of the synchronization message without

the existence of collisions. This dramatically reduces the number of message collisions and message omissions in single-hop networks.

In parallel to the FTA-RFA approach, a fault-tolerant drift calibration approach which ensures approximate agreement of the real-time duration of the period time among all participating nodes in any $(7f+1)$-connected coherent system is presented and implemented. This is very usable for low-cost nodes which usually suffer from a drift rate in the order of $\rho = 10^5$ppm. If the assumption of no drift variation with respect to the chosen resynchronization interval is valid, then practical evaluations have shown that this approach reduces the worst case drift rate to about $\rho = 100$ppm. The presented algorithm also eliminates the problem of common-mode drift such that the nodes eventually stabilize at some constant average interval duration. This implicates that the nodes do not establish time synchronization with respect to real-time. In contrast, our approach only performs internal synchronization with a high precision, but may suffer from high inaccuracies with respect to real-time.

8.3 Performance Aspects

The main advantage of the FTA-RFA approach in combination with MDE is that it is an anonymous algorithm (no ids required) and the message complexity is limited to at most n broadcasts per round, even if the non-anonymous drift calibration algorithm is executed in parallel. This comes from the fact that RFA and FTA share the same synchronization messages and the information required for drift calibration can be packed into this synchronization messages. A further advantage of FTA-RFA is that it transmits only small-sized messages.

Based on the afore discussed approaches and the system model presented in Chapter 3, a detailed evaluation by simulation was performed in order to get a feeling about the performance aspects (*e.g.*, the average time to synchronicity, the achievable average and worst case synchronization precision, *etc.*). The main results of this evaluation are that FTA-RFA in combination with MDE provides a high probability of convergence and a very high average synchronization precision in the order of the delay jitter ε in nearly all connected networks, even in the presence of at most f erroneous nodes (in the case R-RFA is executed) or at most f Byzantine nodes (in the case FTA is executed). In the case of single-hop networks, the activation of the additional features took effect and thus were able to mostly reduce the time to synchronicity to a constant number of rounds irrespective of the network size. However, whereas the simulation results have shown that the FTA-RFA approach is very effective in nearly all connected multi-hop topologies, it is hardly scalable in ring topologies and therefore also less applicable in multi-hop topologies that contain rings. In contrast, the simple E-RFA approach is also scalable in rings, but provides a worse synchronization precision. Last but not least, a special topology type named grouped multi-hop topology is presented. Based on this topology, a simulation of the FTA-RFA algorithm in combination with MDE and fault-tolerant drift calibration provided promising results. In detail, 10 groups of nodes each containing $7f+1$ nodes and f erroneous nodes are connected in a chain-like structure. Simulation results in such a 8-connected network ($f = 1$) have shown that FTA-RFA performs well and achieves an average synchronization precision of $500\mu s$ with an average clock adjustment of $-250\mu s$. These results are promising regarding the fact that the message delay is randomly distributed in the interval $[2.2ms, 2.7ms]$. In other words, it seems that it is possible to synchronize a multi-hop network with an average synchronization precision in the order of the delay jitter ε without knowing the constant part of the message delay.

8.4 Outlook

Future research will rely on a more detailed formal analysis of the convergence speed of E-RFA and R-RFA in fully connected networks in the general case of $n > 2$. Additionally, a formal proof of the convergence speed of the drift calibration will be useful and may be based on the average-consensus proof as stated in [OSS05]. Beside this formal aspect, it may be possible to improve the duty-cycle of the synchronization algorithm by dynamically reducing the maximum message staggering delay after reaching synchronicity. This may be especially important in the case of activated APM in large-sized networks. Therein, the nodes usually require a large bandwidth for a consistent convergence, because otherwise a small bandwidth (small maximum message staggering delay) will increase the number of message collisions and message omissions and thus may prevent the network from getting synchronized. After reaching synchronicity, the maximum message staggering delay can be dramatically reduced, because the reduced number of active nodes resulting from the activated APM feature do not require a large communication bandwidth. A further research question can be the optimal setting of the minimum and maximum message staggering delay with respect to the number of nodes and the underlying communication topology. Beside these improvements, other future work may build on making the APM feature feasible for the use in general multi-hop topologies. An interesting paper that can be used as a groundwork for solving this problem is published by Jang and Lee [JL09]. Last but not least, a practical evaluation on a real testbed system will be useful in order to confirm the correctness of the simulation results.

APPENDIX A

Acronyms

APM Automatic Passive Mode

BP Beacon Period

BRC Basic Rest Circle

CCA Clear Channel Assessment

CD Clique Discovery

COTS Commercial Off-the-shelf

CNI Communication Network Interface

CSMA Carrier Sense Multiple Access

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance

DC Drift Calibration

DFTA Differential Fault-tolerant Averaging

DFTM Differential Fault-tolerant Midpoint

DoS Denial-of-Service

DSSS Direct-sequence Spread Spectrum

FHSS Frequency-hopping Spread Spectrum

FIFO First-in First-out

FTA Fault-tolerant Averaging

FTM Fault-tolerant Midpoint

GPS Global Positioning System

IDL Interface Definition Language

IQR Interquartile Range

ISO International Standards Organization

MAC Media Access Control

MDE Message Delay Estimation

MEMS Microelectromechanical Systems

MIC Message Integrity Code

NTP Network Time Protocol

OSI Open System Interconnection

PCO Pulse-coupled Biological Oscillators

PIB PAN information base

PRC Phase Response Curve

QoS Quality of Service

RAM Random Access Memory

RFA Reachback Firefly Algorithm

RM-ODP Reference Model for Open Distributed Processing

ROM Read-only Memory

RTS/CTS Request To Send / Clear To Send

SNR Signal-to-noise Ratio

TDMA Time Division Multiple Access

TMR Triple Modular Redundancy

UTC Universal Time Coordinated

VCXO Voltage Controlled Crystal Oscillator

VLSI Very Large Scale Integration

WSN Wireless Sensor Network

APPENDIX B

Bibliography

[ACFP09] G. Anastasi, M. Conti, M. D. Francesco, and A. Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, 2009.

[ADG92] A. Arora, S. Dolev, and M. G. Gouda. Maintaining digital clocks in step. In *WDAG '91: Proceedings of the 5th International Workshop on Distributed Algorithms*, pages 71–79, London, UK, 1992. Springer-Verlag.

[AP98] E. Anceaume and I. Puaut. Performance evaluation of clock synchronization algorithms. Technical Report PI 1208, Institut de Recherche en Informatique et Systèmes Aléatoires, October 1998.

[Ash62] W.R. Ashby. Principles of the self-organizing system. In H. von Foerster and G.W. Zopf, editors, *Principles of Self-Organization*, pages 255–278. Pergamon Press, 1962.

[ASH07] E. Armengaud, A. Steininger, and A. Hanzlik. The effect of quartz drift on convergence-average based clock synchronization. *12th IEEE Conference on Emerging Technologies and Factory Automation*, September 2007.

[AW04] H. Attiya and J. L. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (Second Edition)*. McGraw-Hill, 2004.

[AY07] A.A. Abbasi and M. Younis. A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30(14-15):2826–2841, 2007.

[BB76] J. Buck and E. Buck. Synchronous fireflies. *Scientific American*, 234:74–9, 82–5, May 1976.

[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.

[BMT04] P. Blum, L. Meier, and L. Thiele. Improved interval-based clock synchronization in sensor networks. In *IPSN '04: Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, pages 349–358, New York, NY, USA, 2004. ACM.

[BODH08] M. Ben-Or, D. Dolev, and E. N. Hoch. Fast self-stabilizing byzantine tolerant digital clock synchronization. In *PODC '08: Proceedings of the Twenty-seventh ACM Symposium on Principles of Distributed Computing*, pages 385–394, New York, NY, USA, 2008. ACM.

[BS98] G.S. Blair and J.B. Stefani. *Open Distributed Processing and Multimedia*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[BS00] J. Brzezinski and M. Szychowiak. Self-stabilization in distributed systems - a short survey. *Foundations of Computing and Decision Sciences*, 25(1), 2000.

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

[CFS+03] S. Camazine, N.R. Franks, J. Sneyd, E. Bonabeau, J.-L. Deneubourg, and G . Theraula. *Self-Organization in Biological Systems*. Princeton University Press, Princeton, NJ, USA, 2003.

[CL07] L. Chen and J. Leneutre. Toward secure and scalable time synchronization in ad hoc networks. *Computer Communications*, 30(11-12):2453–2467, 2007.

[Com97] H.-P. Company. Fundamentals of quartz oscillators. *HP Application Note 200-2*, September 1997.

[Cri89] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, 1989.

[DBR08] J. Degesys, P. Basu, and J. Redi. Synchronization of strongly pulse-coupled oscillators with refractory periods and random medium access. In *SAC '08: Proceedings of the 2008 ACM Symposium on Applied Computing*, pages 1976–1980, New York, NY, USA, 2008. ACM.

[DD05] A. Daliot and D. Dolev. Self-stabilization of byzantine protocols. In T. Herman and S. Tixeuil, editors, *Self-Stabilizing Systems*, volume 3764 of *Lecture Notes in Computer Science*, pages 48–67. Springer, 2005.

[DD06] A. Daliot and D. Dolev. Self-stabilizing byzantine agreement. In *PODC '06: Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, pages 143–152, New York, NY, USA, 2006. ACM.

[DD08] A. Daliot and D. Dolev. Self-stabilizing byzantine pulse synchronization (revised version). *The Computing Research Repository (CoRR)*, abs/cs/0608092, February 2008. informal publication.

[DDP06] A. Daliot, D. Dolev, and H. Parnas. Linear-time self-stabilizing byzantine clock synchronization. *The Computing Research Repository (CoRR)*, abs/cs/0608096, 2006.

[DDP08] A. Daliot, D. Dolev, and H. Parnas. Self-stabilizing pulse synchronization inspired by biological pacemaker networks. *The Computing Research Repository (CoRR)*, abs/0803.0241, 2008.

[DG06] A. K. Datta and M. Gradinariu, editors. *Stabilization, Safety, and Security of Distributed Systems, 8th International Symposium, SSS 2006, Dallas, TX, USA, November 17-19, 2006, Proceedings*, volume 4280 of *Lecture Notes in Computer Science*. Springer, 2006.

[DH04] H. Dai and R. Han. Tsync: A lightweight bidirectional time synchronization service for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 8:125–139, 2004.

[DH07a] D. Dolev and E. N. Hoch. Byzantine self-stabilizing pulse in a bounded-delay model. In T. Masuzawa and S. Tixeuil, editors, *SSS*, volume 4838 of *Lecture Notes in Computer Science*, pages 234–252. Springer, 2007.

[DH07b] D. Dolev and E. N. Hoch. On self-stabilizing synchronous actions despite byzantine attacks. In Andrzej Pelc, editor, *Distributed Algorithms*, volume 4731/2007 of *Lecture Notes in Computer Science*, pages 193–207, September 2007.

[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.

[Dij74] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[DLP+83] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching approximate agreement in the presence of faults. In *Symposium on Reliability in Distributed Software and Database Systems*, pages 145–154, 1983.

[DLP+86] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):499–516, July 1986.

[DN08] J. Degesys and R. Nagpal. Towards desynchronization of multi-hop topologies. In *SASO '08: Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 129–138, Washington, DC, USA, 2008. IEEE Computer Society.

[Dol00] S. Dolev. *Self-Stabilization*. MIT Press, 2000.

[Dou02] J.R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[Dre07] F. Dressler. *Modeling Complex Systems (Graduate Texts in Contemporary Physics)*. John Wiley & Sons, 2007.

[DRPN07] J. Degesys, I. Rose, A. Patel, and R. Nagpal. Desync: self-organizing desynchronization and tdma on wireless sensor networks. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 11–20, New York, NY, USA, 2007. ACM.

[DW93] S. Dolev and J. L. Welch. Wait-free clock synchronization. In *PODC '93: Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*, pages 97–108, New York, NY, USA, 1993. ACM.

[DW04] S. Dolev and J. L. Welch. Self-stabilizing clock synchronization in the presence of byzantine faults. *J. ACM*, 51(5):780–799, 2004.

[EGE02] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *OSDI '02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pages 147–163, New York, NY, USA, 2002. ACM.

[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.

[EPS04] W. Elmenreich, S. Pitzek, and M. Schlager. Modeling distributed embedded applications on an interface file system. In *Proceedings of the 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 175–182, May 2004.

[ER90] S. Even and S. Rajsbaum. Unison in distributed networks. In *Sequences: combinatorics, compression, security, and transmission*, pages 479–487. Springer, New York, NY, USA, 1990.

[ER03] J. Elson and K. Römer. Wireless sensor networks: a new regime for time synchronization. *SIGCOMM Computer Communications*, 33(1):149–154, January 2003.

[Fai07] Y. R. Faizulkhakov. Time synchronization methods for wireless sensor networks: A survey. *Journal on Programming and Computer Software*, 33(4):214–226, 2007.

[FC95] C. Fetzer and F. Cristian. An optimal internal clock synchronization algorithm. In *Proceedings 10th Annual IEEE Conference on Computer Assurance*, Gaithersburg, MD, June 1995.

[GČHS05] S. Ganeriwal, S. Čapkun, C.C. Han, and M.B. Srivastava. Secure time synchronization service for sensor networks. In *WiSe '05: Proceedings of the 4th ACM Workshop on Wireless Security*, pages 97–106, New York, NY, USA, 2005. ACM.

[GFH+03] Giovanna, N. Foukia, S. Hassas, A. Karageorgos, S.K. Mostéfaoui, O.F. Omer, M. Ulieru, P. Valckenaers, and C. Van Aart. Self-organisation: Paradigms and applications. In G.D.M. Serugendo, A. Karageorgos, O.F. Rana, and F. Zambonelli, editors, *Engineering Self-Organising Systems*, volume 2977 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2003.

[GGH+05] S. Ganeriwal, D. Ganesan, M. Hansen, M.B. Srivastava, and D. Estrin. Rate-adaptive time synchronization for long-lived sensor networks. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 374–375, New York, NY, USA, 2005. ACM.

[GGS+05] S. Ganeriwal, D. Ganesan, H. Shim, V. Tsiatsis, and M.B. Srivastava. Estimating clock uncertainty for efficient duty-cycling in sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 130–141, New York, NY, USA, 2005. ACM.

[GH90] M. G. Gouda and T. Herman. Stabilizing unison. *Information Processing Letters*, 35(4):171–175, 1990.

[GK04] S. Graham and P.R. Kumar. Time in general-purpose control systems: the control time protocol and an experimental evaluation. volume 4, pages 4004–4009 Vol.4, Dec. 2004.

[GK06] A. Giridhar and P.R. Kumar. Distributed clock synchronization over wireless networks: Algorithms and analysis. pages 4915–4920, Dec. 2006.

[GKS03a] S. Ganeriwal, R. Kumar, and M.B. Srivastava. Network-wide time synchronization in sensor networks. *NESL Technical Report*, May 2003.

[GKS03b] S. Ganeriwal, R. Kumar, and M.B. Srivastava. Timing-sync protocol for sensor networks. In *SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 138–149, New York, NY, USA, 2003. ACM.

[GKW+02] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. An empirical study of epidemic algorithms in large scale multihop wireless networks. Technical Report IRB-TR-02-003, Intel Research, March 2002.

[GM04] J. Gehrke and S. Madden. Query processing in sensor networks. *IEEE Pervasive Computing*, 3(1):46–55, 2004.

[GPČS08] S. Ganeriwal, C. Pöpper, S. Čapkun, and M.B. Srivastava. Secure time synchronization in sensor networks. *ACM Transactions on Information and System Security*, 11(4):1–35, 2008.

[HDD06] E. N. Hoch, D. Dolev, and A. Daliot. Self-stabilizing byzantine digital clock synchronization. In Datta and Gradinariu [DG06], pages 350–362.

[HdM08] R. Holzer and H. de Meer. On modeling of self-organizing systems. In *Autonomics '08: Proceedings of the 2nd International Conference on Autonomic Computing and Communication Systems*, pages 1–6, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[HDQK09] P.H. Huang, M. Desai, X. Qiu, and B. Krishnamachari. On the multihop performance of synchronization mechanisms in high propagation delay networks. *IEEE Transactions on Computers*, 58(5):577–590, 2009.

[HE04] B. Huber and W. Elmenreich. Wireless time-triggered real-time communication. In *Proceedings of the Second Workshop on Intelligent Solutions for Embedded Systems (WISES'04)*, pages 169–182, Graz, Austria, June 2004.

[Hil90] M.D. Hill. What is scalability? *SIGARCH Comput. Archit. News*, 18(4):18–21, 1990.

[HL02] L. Huang and T. Lai. On the scalability of IEEE 802.11 ad hoc networks. In *MobiHoc '02: Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking & computing*, pages 173–182, New York, NY, USA, 2002. ACM.

[Hoc07] E. N. Hoch. Self-stabilizing byzantine pulse and clock synchronization. Master's thesis, School of Engineering and Computer Science, The Hebrew University of Jerusalem, Israel, March 2007.

[HPJ01] Y. Hu, A. Perrig, and D.B. Johnson. Wormhole detection in wireless ad hoc networks. Technical Report TR01-384, Department of Computer Science, Rice University, December 2001.

[HPJ03] Y. Hu, A. Perrig, and D.B. Johnson. Packet leashes: a defense against wormhole attacks in wireless networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, volume 3, pages 1976–1986 vol.3, 2003.

[HS03] A. Hu and S. D. Servetto. Asymptotically optimal time synchronization in dense sensor networks. In *WSNA '03: Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications*, pages 1–10, New York, NY, USA, 2003. ACM.

[HS05] Y. 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.

[HWM06] K. Herrmann, M. Werner, and G. Mühl. A methodology for classifying self-organizing software systems. *International Transactions on Systems Science and Applications*, 2(1):41–50, 2006.

[IEC96] ISO IEC. Information technology - open distributed processing - reference model: Architecture. international standard iso/iec 10746-3:1996(e), 1996.

[IEE90] IEEE. IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, December 1990.

[iee99] IEEE standard definitions of physical quantities for fundamental frequency and time metrology - random instabilities. *IEEE Std 1139-1999*, 1999.

[iee03] Information technology- telecommunications and information exchange between systems- local and metropolitan area networks- specific requirements-part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *ANSI/IEEE Std 802.11, 1999 Edition (R2003)*, pages i–513, 2003.

[IGE00] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67, New York, NY, USA, 2000. ACM.

[JFF08] Z. Jerzak, R. Fach, and C. Fetzer. Adaptive internal clock synchronization. In *SRDS '08: Proceedings of the 2008 Symposium on Reliable Distributed Systems*, pages 217–226, Washington, DC, USA, 2008. IEEE Computer Society.

[JL09] U. Jang and S. Lee. Reduced node k-coverage in dense wireless sensor networks. *Software Technologies for Future Dependable Distributed Systems*, 0:225–229, 2009.

[JPS03] D. Johnson, S. PalChaudhuri, and A. Saha. Probabilistic clock synchronization service in sensor networks. Technical Report TR03-418, Department of Computer Science, Rice University, Houston, TX, April 2003.

[KC03] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[KDL+06] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler. Elapsed time on arrival: a simple and versatile primitive for canonical time synchronisation services. *International Journal of Ad Hoc and Ubiquitous Computing*, 1(4):239–251, 2006.

[Kli97] P. A. Kline. *Atomic Clock Augmentation for Receivers Using the Global Positioning System*. Dissertation, Bradley Department of Electrical Engineering, Faculty of the Virginia Tech, 1997.

[KO87] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers*, C-36(8):933–940, 1987.

[Kop97] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[KS02] H. Kopetz and N. Suri. Compositional design of RT systems: A conceptual basis for specification of linking interfaces. Research Report 37/2002, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[LE08] R. Leidenfrost and W. Elmenreich. Establishing wireless time-triggered communication using a firefly clock synchronization approach. In *Proceedings of the Sixth International Workshop on Intelligent Solutions in Embedded Systems (WISES'08)*, pages 227–244, Regensburg, Germany, June 2008.

[LE09] R. Leidenfrost and W. Elmenreich. Firefly clock synchronization in an 802.15.4 wireless network. 2009.

[LH08] J. Leu and H. Hsieh. An autonomous wireless sensor network with fault resilience. In *WIMOB '08: Proceedings of the 2008 IEEE International Conference on Wireless & Mobile Computing, Networking & Communication*, pages 223–227, Washington, DC, USA, 2008. IEEE Computer Society.

[Liu08] J. Liu. Scalable synchronization of clocks in wireless sensor networks. *Ad Hoc Networks*, 6(5):791–804, 2008.

[LJP08] S. Lee, U. Jang, and J. Park. Fast fault-tolerant time synchronization for wireless sensor networks. In *ISORC '08: Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-time Distributed Computing*, pages 178–185, Washington, DC, USA, 2008. IEEE Computer Society.

[LL84a] J. Lundelius and N. Lynch. A new fault-tolerant algorithm for clock synchronization. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 75–88, Vancouver, Canada, August 1984.

[LL84b] J. Lundelius and N. Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62(2/3):190–204, August–September 1984.

[LMS85] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *J. ACM*, 32(1):52–78, 1985.

[Lön99] H. Lönn. A fault tolerant clock synchronization algorithm for systems with low-precision oscillators. In *EDCC-3: Proceedings of the Third European Dependable Computing Conference on Dependable Computing*, pages 88–105, London, UK, 1999. Springer-Verlag.

[LR04] J.-C. Laprie and B. Randell. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004. Fellow-Avizienis,, Algirdas and Senior Member-Landwehr,, Carl.

[LR06] Q. Li and D. Rus. Global clock synchronization in sensor networks. *IEEE Transactions on Computers*, 55(2):214–226, 2006.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[LW04] D. Lucarelli and I-J. Wang. Decentralized synchronization protocols with nearest neighbor communication. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 62–68, New York, NY, USA, 2004. ACM.

[LZ03] T. Lai and D. Zhou. Efficient and scalable IEEE 802.11 ad-hoc-mode timing synchronization function. In *AINA '03: Proceedings of the 17th International Conference on Advanced Information Networking and Applications*, page 318, Washington, DC, USA, 2003. IEEE Computer Society.

[Mah06] N. P. Mahalik. *Sensor Networks and Configuration: Fundamentals, Standards, Platforms, and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Mal06] M. R. Malekpour. A byzantine-fault tolerant self-stabilizing protocol for distributed clock synchronization systems. In Datta and Gradinariu [DG06], pages 411–427.

[MBRS08] O. Mirabella, M. Brischetto, A. Raucea, and P. Sindoni. Dynamic continuous clock synchronization for IEEE 802.15.4 based sensor networks. pages 2438–2444, November 2008.

[MBT04] L. Meier, P. Blum, and L. Thiele. Internal synchronization of drift-constraint clocks in ad-hoc sensor networks. In *MobiHoc '04: Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 90–97, New York, NY, USA, 2004. ACM.

[MFLT05] N. Mitton, E. Fleury, I. G. Lassous, and S. Tixeuil. Self-stabilization in self-organized multihop wireless networks. In *ICDCSW '05: Proceedings of the Second International Workshop on Wireless Ad Hoc Networking*, pages 909–915, Washington, DC, USA, 2005. IEEE Computer Society.

[MFNT00a] M. Mock, R. Frings, E. Nett, and S. Trikaliotis. Clock synchronization for wireless local area networks. pages 183–189, 2000.

[MFNT00b] M. Mock, R. Frings, E. Nett, and S. Trikaliotis. Continuous clock synchronization in wireless real-time applications. pages 125–132, 2000.

[Mil91] D.L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[MK06] V. Mittal and V. Kulathumani. A fault-local self-stabilizing clustering service for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 17(9):912–922, 2006. Member-Demirbas, Murat and Senior Member-Arora, Anish.

[MKSL04] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 39–49, New York, NY, USA, 2004. ACM.

[MM96] R. Mathar and J. Mattfeldt. Pulse-coupled decentral synchronization. *SIAM J. Appl. Math.*, 56(4):1094–1106, 1996.

[MO83] K. Marzullo and S. Owicki. Maintaining the time in a distributed system. In *PODC '83: Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, pages 295–305, New York, NY, USA, 1983. ACM.

[MRS05] M. Manzo, T. Roosta, and S. Sastry. Time synchronization attacks in sensor networks. In *SASN '05: Proceedings of the 3rd ACM workshop on Security of Ad Hoc and sensor networks*, pages 107–116, New York, NY, USA, 2005. ACM.

[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.

[ND00] K. A. Noordin and K. Dimyati. Differential fault-tolerant average for internally synchronising clocks within distributed environments. *TENCON 2000. Proceedings*, 3:234–236 vol.3, 2000.

[Neu94] B.C. Neuman. Scale in distributed systems. In Thomas Lee Casavant and Mukesh Singhal, editors, *Readings in Distributed Computing Systems*, pages 463–489. IEEE CS Press, Los Alamitos, CA, USA, 1994.

[NOKM08] G. Nishikawa, F. Ooshita, H. Kakugawa, and T. Masuzawa. A stable clustering algorithm for mobile ad hoc networks based on attractor selection. In *BIONETICS '08: Proceedings of the 3rd International Conference on Bio-Inspired Models of Network, Information and Computing Sytems*, pages 1–6, ICST, Brussels, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[NSSP04] J. Newsome, E. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: analysis & defenses. In *IPSN '04: Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, pages 259–268, New York, NY, USA, 2004. ACM.

[OSS05] R. Olfati-Saber and J.S. Shamma. Consensus filters for sensor networks and distributed sensor fusion. In *44th IEEE Conference on Decision and Control and European Control Conference*, pages 6698–6703, December 2005.

[PDN07] A. Patel, J. Degesys, and R. Nagpal. Desynchronization: The theory of self-organizing algorithms for round-robin scheduling. In *SASO '07: Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems*, pages 87–96, Washington, DC, USA, 2007. IEEE Computer Society.

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

[Pin03] S. Ping. Delay measurement time synchronization for wireless sensor networks. Technical Report IRB-TR-03-013, Intel Research, Berkeley, June 2003.

[PK00] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[Pol94] S. Poledna. *Replica Determinism in Fault-Tolerant Real-Time Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 1994.

[PSJ04] S. PalChaudhuri, A. K. Saha, and D. B. Johnson. Adaptive clock synchronization in sensor networks. In *IPSN '04: Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, pages 340–348, New York, NY, USA, 2004. ACM.

[PST+02] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. Spins: security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, 2002.

[PT94] M. Papatriantafilou and P. Tsigas. On self-stabilizing wait-free clock synchronization. In *SWAT '94: Proceedings of the 4th Scandinavian Workshop on Algorithm Theory*, pages 267–277, London, UK, 1994. Springer-Verlag.

[Put01] J.R. Putman. *Architecting with RM-ODP*. Prentice Hall, 2001.

[RBM05] K. Römer, P. Blum, and L. Meier. Time synchronization and calibration in wireless sensor networks. In Ivan Stojmenovic, editor, *Handbook of Sensor Networks: Algorithms and Architectures*, pages 199–237. John Wiley & Sons, September 2005.

[RK04] C.H. Rentel and T. Kunz. Network synchronization in wireless ad hoc networks. Technical Report SCE-04-08, Carleton University, Systems and Computer Engineering, July 2004.

[RLK+09] I. Rhee, J. Lee, J. Kim, E. Serpedin, and Y. Wu. Clock synchronization in wireless sensor networks: An overview. *Sensors*, 9(1):56–85, 2009.

[RM04] K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004.

[Röm01] K. Römer. Time synchronization in ad hoc networks. In *MobiHoc '01: Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking & computing*, pages 173–182, New York, NY, USA, 2001. ACM.

[RSB90] P. Ramanathan, K.G. Shin, and R.W. Butler. Fault-tolerant clock synchronization in distributed systems. *Computer*, 23(10):33–42, 1990.

[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.

[SA05] W. Su and I. F. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 13(2):384–397, 2005.

[SAHW90] D. B. Sullivan, D. W. Allan, D. A. Howe, and F. L. Walls. Characterization of clocks and oscillators. *NASA STI/Recon Technical Report N*, 91:22539–+, March 1990.

[San07] D. Sanchez. Secure, accurate and precise time synchronization for wireless sensor networks. In *Q2SWinet '07: Proceedings of the 3rd ACM Workshop on QoS and Security for Wireless and Mobile Networks*, pages 105–112, New York, NY, USA, 2007. ACM.

[Sat88] M. Satyanarayanan. On the influence of scale in a distributed system. In *ICSE '88: Proceedings of the 10th international conference on Software engineering*, pages 10–18, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press.

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

[SBK06] R. Solis, V.S. Borkar, and P.R. Kumar. A new distributed time synchronization protocol for multihop wireless networks. pages 2734–2739, Dec. 2006.

[Sch87] F. B. Schneider. Understanding protocols for byzantine clock synchronization. Technical report, Department of Computer Science, Ithaca, NY, USA, 1987.

[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.

[SCS04] J.P. Sheu, C.M. Chao, and C.W. Sun. A clock synchronization algorithm for multi-hop wireless ad hoc networks. pages 574–581, 2004.

[Sha01] C.E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[SKLD06] J. Sallai, B. Kusy, A. Ledeczi, and P. Dutta. On the scalability of routing integrated time synchronization. *3rd European Workshop on Wireless Sensor Networks (EWSN 2006)*, February 2006.

[SML+04] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 1–12, New York, NY, USA, 2004. ACM.

[SNW06a] K. Sun, P. Ning, and C. Wang. Secure and resilient clock synchronization in wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, 24(2):395–408, February 2006.

[SNW06b] K. Sun, P. Ning, and C. Wang. Tinysersync: secure and resilient time synchronization in wireless sensor networks. In *CCS '06: Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 264–277, New York, NY, USA, 2006. ACM.

[Soc03] 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.

[SS93] St. H. Strogatz and I. Stewart. Coupled oscillators and biological synchronization. *Scientific American*, 269(6):102–108, December 1993.

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

[Sun05] Kun Sun. Fault-tolerant cluster-wise clock synchronization for wireless sensor networks. *IEEE Transactions on Dependable and Secure Computing*, 2(3):177–189, 2005. P. Ning and C. Wang.

[SV03] M. L. Sichitiu and C. Veerarittiphan. Simple, accurate time synchronization for wireless sensor networks. *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, 2:1266–1273 vol.2, March 2003.

[SV04] J. So and N.H. Vaidya. Mtsf: A timing synchronization protocol to support synchronous operations in multihop wireless networks. Technical report, UIUC, January 2004.

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

[SWL90] B. Simons, J. L. Welch, and N. Lynch. An overview of clock synchronization. In *Fault-tolerant Distributed Computing*, pages 84–96. Springer-Verlag, London, UK, 1990.

[SY04] F. Sivrikaya and B. Yener. Time synchronization in sensor networks: a survey. *Network, IEEE*, 18(4):45–50, July 2004.

[SZC05] H. Song, S. Zhu, and G. Cao. Attack-resilient time synchronization for wireless sensor networks. pages 8 pp.–772, November 2005.

[TAB06] A. Tyrrell, G. Auer, and C. Bettstetter. Fireflies as role models for synchronization in ad hoc networks. In *BIONETICS '06: Proceedings of the first international conference on Bio inspired models of network, information and computing systems*, page 4, New York, NY, USA, 2006. ACM.

[TAB07] A. Tyrell, G. Auer, and C. Bettstetter. Biologically inspired synchronization for wireless networks. In F. Dressler and I. Carreras, editors, *Studies in Computational Intelligence*, volume 69, pages 47–62. Springer, 2007.

[TAB08] A. Tyrell, G. Auer, and C. Bettstetter. Emergent slot synchronization in wireless networks. *to be appear in IEEE Transactions on Mobile Computing*, 2008.

[TS06] A. S. Tanenbaum and M. Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[Tul04] D. Tulone. A resource–efficient time estimation for wireless sensor networks. In *DIALM-POMC '04: Proceedings of the 2004 joint Workshop on Foundations of Mobile Computing*, pages 52–59, New York, NY, USA, 2004. ACM.

[vGR03] J. van Greunen and J. Rabaey. Lightweight time synchronization for sensor networks. In *WSNA '03: Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications*, pages 11–19, New York, NY, USA, 2003. ACM.

[Vig00] J. R. Vig. Quartz crystal resonators and oscillators, January 2000. http://www.am1.us/Papers/U11625

[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.

[WC03] X. F. Wang and G. Chen. Complex networks: small-world, scale-free and beyond. *Circuits and Systems Magazine, IEEE*, 3(1):6–20, 2003.

[WLLP01] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister. Smart dust: Communicating with a cubic-millimeter computer. *Computer*, 34(1):44–51, 2001.

[WSS03] A. D. Wood, J. A. Stankovic, and S. H. Son. Jam: A jammed-area mapping service for sensor networks. In *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, page 286, Washington, DC, USA, 2003. IEEE Computer Society.

[YHE04] Wei Ye, John Heidemann, and Deborah Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 12(3):493–506, 2004.

[YKT03] W. Yuan, S. V. Krishnamurthy, and S. K. Tripathi. Synchronization of multiple levels of data fusion in wireless sensor networks. In *In Proceedings of IEEE Globecom*, pages 221–225, 2003.

[Zad63] L.A. Zadeh. On the definition of adaptivity. *Proceedings of the IEEE*, 51(3):469–470, 1963.

[ZG04] F. Zhao and L. Guibas. *Wireless Sensor Networks: An Information Processing Approach (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann, July 2004.

[Zho05] D. Zhou. A compatible and scalable clock synchronization protocol in IEEE 802.11 ad hoc networks. In *ICPP '05: Proceedings of the 2005 International Conference on Parallel Processing*, pages 295–302, Washington, DC, USA, 2005. IEEE Computer Society.

[ZSJ03] S. Zhu, S. Setia, and S. Jajodia. Leap: efficient security mechanisms for large-scale distributed sensor networks. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 62–72, New York, NY, USA, 2003. ACM.

APPENDIX C

Moved Proofs

Proof. (of Lemma 1)
Assume the two clocks p_i and p_j are initially synchronized to $-\Pi^U \leq \Pi \leq \Pi^U$. W.l.o.g. let p_i be the faster node. We further use p_i as the reference for the precision $\Pi(\varphi) = rt_{\varphi_j=\varphi} - rt_{\varphi_i=\varphi}$ where $rt_{\varphi_j=\varphi}$ denotes the real time when p_j's phase φ_j reached φ. We further assume that the next time p_i reaches the threshold 1 is at time $t = 0$. Let $\Pi_0 = \Pi(0)$ be the corresponding precision at $t = 0$. For $t \leq 0$ we then have $\varphi_i(t) = 1 + \frac{t}{T\cdot(1-\rho)}$ and $\varphi_j(t) = 1 + \frac{t-\Pi_0}{T\cdot(1+\rho)}$. Let r_i resp. r_j denote the relative message staggering delay the node p_i resp. p_j has calculated for the last transmission. If the last fire event of p_i was at $\varphi_i = 1 - r_i$, then with respect to the communication delay d, p_j received the phase at $\varphi_j^{recv} = 1 + \frac{-r_i\cdot T\cdot(1-\rho)+d-\Pi_0}{T\cdot(1+\rho)}$ and consequently adds the offset r_i leading to $\varphi_j^{fire} = 1 + r_i\cdot(1-\frac{1}{R}) + \frac{d-\Pi_0}{T\cdot(1+\rho)}$. Similarly, a fire event from p_j with offset r_j is received by p_i at phase $\varphi_i^{fire} = 1 - r_j\cdot(R-1) + \frac{d+\Pi_0}{T\cdot(1-\rho)}$. Let $\varphi_{j,min}^{fire}$, $\varphi_{j,max}^{fire}$, $\varphi_{i,min}^{fire}$, $\varphi_{i,max}^{fire}$ be the minimum resp. maximum possible phases of the calculated firing events. If $\alpha > \max\left(\frac{1}{\varphi_{i,min}^{fire}}, \frac{1}{\varphi_{j,min}^{fire}}\right)$, then it is guaranteed that $\Delta(\varphi_i^{fire}) = 1 - \varphi_i^{fire}$ resp. $\Delta(\varphi_j^{fire}) = 1 - \varphi_j^{fire}$. Since $\varphi_{i,min}^{fire} < \varphi_{j,min}^{fire}$, we have $\alpha > \frac{1}{\varphi_{i,min}^{fire}}$ as stated.

Based on the current precision Π_0 and the phase advance of p_i and p_j at time $t = 0$ labeled by $\Delta_i = \Delta(\varphi_i^{fire})$ and $\Delta_j = \Delta(\varphi_j^{fire})$, we are able to calculate the precision Π_{next} the next time p_i reaches the threshold. That is, $\Pi_{next} = \Pi_0 + \Gamma + T\cdot(\Delta_i\cdot(1-\rho) - \Delta_j\cdot(1+\rho))$. However, we have to distinguish between three cases depending on Π_0. In detail, if $\Pi_0 \in [0, \Pi^U]$, then (1) $\Delta_i = 0$ and $\Delta_j > 0$, or if $\Pi_0 \in [0, \Gamma)$, then also (2) $\Delta_i > 0$ and $\Delta_j > 0$, or finally if $\Pi_0 \in [-\Pi^U, 0]$, then due to Line 4 of Algorithm 1 we have (3) $\Delta_i > 0$ and $\Delta_j = 0$. Note that the overlapping of (1) and (2) is volitional, because if $\Pi_0 \in [0, \Pi^U]$, then both cases can occur and hence must be considered. Further note that the bound of Γ ensures that the interception point of the phase of both nodes is within the last period. In order to keep the clocks within the precision, the inequality $-\Pi^U \leq \Pi_{next} \leq \Pi^U$ must be valid for all three cases. From the first case we get $\Pi^U \geq (1 + r_{max})\Gamma + d + \varepsilon$ and $r_{min} \geq -1 - \frac{\Pi^U + d}{\Gamma}$. From the third case it follows $\Pi^U \geq (1 + r_{max})\Gamma - d$ and $r_{min} \geq -1 - \frac{\Pi^U - d - \varepsilon}{\Gamma}$. Note that r_{min} is always valid due to the definition of Π^U. From the second case, it can be derived that $\Pi^U \geq (1 + 2r_{max})\Gamma + \varepsilon$ and $r_{min} \geq \frac{\varepsilon - \Pi^U}{2\Gamma}$. Again, Π^U ensures that r_{min} is valid. The worst case precision with respect to these three cases then equals $\Pi^U = (1 + r_{max})\Gamma + \varepsilon + \max(\Gamma r_{max}, d)$.

Note that the correctness of the proof requires that a node advances its phase at most once per period. However, if $\Pi_0 > 0$, then p_j may initiate a firing event after p_i already passed the threshold. Simply setting $r_{min} \geq \frac{\Pi^U + d + \varepsilon}{T \cdot (1-\rho)}$ avoids this effect.

In order to get the worst case precision, we further have to incorporate the precision (I) $\Pi(\Delta_i)$ and (II) $\Pi(\Delta_j)$ for all three mentioned cases. In detail, for $\Pi_0 \in [0, \Pi^U]$ we additionally have to analyze for case (1) if the equation $-\Pi^U \leq \Pi_0 - \Delta_j \cdot (1-\rho) \cdot T \leq \Pi^U$ holds and for case (2), if $-\Pi^U \leq (\Delta_i - \Delta_j) \cdot (1+\rho) \cdot T + \Pi \leq \Pi^U$ and $-\Pi^U \leq \Pi - (\Delta_j - \Delta_i) \cdot (1-\rho) \cdot T \leq \Pi^U$ are valid. Similarly for $\Pi_0 \in [-\Pi^U, 0]$ it must be ensured that $-\Pi^U \leq \Delta_i \cdot (1+\rho) \cdot T + \Pi \leq \Pi^U$. From these equations we can derive the following additional bounds: $\frac{R-2}{4-3R} \leq r_{max} \leq \frac{1}{R-1}$, and $\Pi^U \geq (d+\varepsilon)R - \Gamma R r_{min}$. Therefore, if we want that r_{max} is bounded between $[0,1]$, then $\rho < \frac{1}{7}$ must hold. Furthermore, in the case of $\rho = 0$, we have to adapt the worst case precision to $\Pi^U = (1+r_{max})\Gamma + \varepsilon R + \max(\Gamma r_{max}, dR)$ which now equals the worst case upper bound, since all possible cases were considered.

Finally, it should be mentioned that the maximum relative message staggering delay r_{max} must be smaller than $\frac{1}{2}$. Otherwise, assume the case where both nodes are initially $\frac{1}{2}$ apart. Then both nodes will never perform a phase advance due to Line 4 of the algorithm. □

Proof. (of Lemma 2) The proof is based on the fact that if α is too large, then the nodes will infinitely often enter the same firing state. Let p_A and p_B be the two participating processors where p_A is the first node reaching the threshold. The initial firing state then is $C(N,A,1) = (\Delta_{A,1}, \varphi_{B,1})$ with $\Delta_{A,1} < \varphi_{B,1}$. Next, p_B reaches the threshold leading to $C(N,B,1) = (\varphi_{A,1}; \Delta_{B,1})$ with $\varphi_{A,1} = \Delta_{A,1} + 1 - \varphi_{B,1}$ and $\Delta_{B,1} = \Delta(\varphi_{B,1})$. The next time p_A reaches the threshold is at $C(N,A,2) = (\Delta_{A,2}; \varphi_{B,2})$ with $\Delta_{A,2} = \Delta(\Delta_{A,1} + 1 - \varphi_{B,1})$ and $\varphi_{B,2} = \Delta(\varphi_{B,1}) + \varphi_{B,1} - \Delta_{A,1}$. Finally p_B again reaches the threshold at $C(N,B,2) = (\varphi_{A,2}; \Delta_{B,2})$ with $\varphi_{A,2} = \Delta(\Delta_{A,1} + 1 - \varphi_{B,1}) + 1 - \Delta(\varphi_{B,1}) - \varphi_{B,1} + \Delta_{A,1}$ and $\Delta_{B,2} = \Delta(\Delta(\varphi_{B,1}) + \varphi_{B,1} - \Delta_{A,1})$.

If we assume that (1) $\alpha \cdot (\varphi_{B,1}) \geq 1$, then the phase advance can be reduced to $\Delta(\varphi_{B,1}) = 1 - \varphi_{B,1}$. The same applies to (2) $\alpha \cdot (\Delta_{A,1} + 1 - \varphi_{B,1}) \geq 1$ and (3) $\alpha \cdot (1 - \Delta_{A,1}) \geq 1$. Thus, if all three conditions are true, $C(N,B,2)$ can be redefined to $C(N,B,2) = (\varphi_{B,1}, \Delta_{A,0})$. In other words, the nodes will infinitely often enter the initial firing state. We now have to find the lowest α where the inequation $\alpha \geq \max\left(\frac{1}{\varphi_{B,1}}, \frac{1}{1-\Delta_{A,1}}, \frac{1}{1+\Delta_{A,1}-\varphi_{B,1}}\right)$ is valid. Equalizing all three conditions yields $\Delta_{A,1} = \frac{1}{3}$ and $\varphi_{B,1} = \frac{2}{3}$. Thus we get $\alpha \geq \max\left(\frac{3}{2}, \frac{3}{2}, \frac{3}{2}\right) = \frac{3}{2}$. □

Proof. (of Lemma 3) The maximum phase advance occurs if the firing events are at close quarters such that no event is ignored due to Line 9 of Algorithm 5. In detail, assume a node received the firing event at the phases $\varphi_0 < \varphi_1 < \cdots < \varphi_{n-1} = 1$. The first phase advance then equals $\lambda_0 = \varphi_0 \cdot \gamma$, where $\gamma = \alpha - 1$. Due to Line 9 of Algorithm 5, the earliest next time the node performs a phase advance can only be at $\varphi_1 = \varphi_0 + \lambda_0$ and equals $\lambda_1 = (\varphi_1 + \lambda_0)\gamma$. Generally, $\varphi_{k+1} = \varphi_k + \lambda_k$ and $\lambda_{k+1} = (\varphi_{k+1} + \sum_{i=1}^{k} \lambda_i)\gamma$ for $0 \leq k < n-1$. Solving the recursion leads to $\varphi_k = \varphi_0 + \sum_{i=0}^{k-1} \lambda_i$ and thus $\lambda_{k+1} = (\varphi_0 + 2\sum_{i=0}^{k} \lambda_i)\gamma$. Solving the equation for $\lambda_{k+1} - \lambda_k$ then yields $\lambda_k = (1+2\gamma)^k \gamma \varphi_0$. The overall phase advance thus equals $\Delta = \sum_{i=0}^{n-2} \lambda_i = \gamma \sum_{i=0}^{n-2} (1+2\gamma)^i = ((1+2\gamma)^{n-1} - 1)\varphi_0/2$. Since the maximum Δ occurs when $\varphi_{n-1} = 1$, we finally get $\Delta = \frac{(1+2\gamma)^{n-1}-1}{(1+2\gamma)^{n-1}+1}$. □

Proof. (of Lemma 7) Let $P_1 = \langle \varphi_0, \varphi_1, \ldots, \varphi_{n-1} \rangle$ be the pulse state at the beginning of the round. W.l.o.g., we assume that $\varphi_i < \varphi_j$ for all $0 \leq i < j < n$. For sake of simplicity we do not assume that $\varphi_i \leq \varphi_j$. In the perfect case with $f = 0$, if n is odd, then the condition $diff \leq k$ is valid for all nodes p_i with $\lfloor (n-1)/2 \rfloor - \lfloor k/2 \rfloor \leq i \leq \lfloor (n-1)/2 \rfloor + \lfloor k/2 \rfloor$. Consequently, exactly $\lfloor k/2 \rfloor \cdot 2 + 1$ nodes become passive at the end of the round. Similarly, if n is even, exactly $\lfloor (k+1)/2 \rfloor \cdot 2$ nodes become passive at the end of the round.

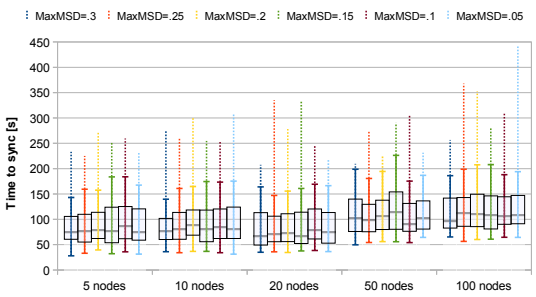
If we consider the fact that $f > 0$, then the Byzantine nodes can vary the number of nodes that can become active at the end of the round between some lower and upper bound. The lower bound simply results from the fact that at most f out of the elected nodes are Byzantine. Since these nodes do not adhere to the code, they may keep active. Consequently, at least $\max(0, k-f) \leq \max(0, \min(\lfloor k/2 \rfloor \cdot 2 + 1 - f, \lfloor (k+1)/2 \rfloor \cdot 2 - f))$ definitively become passive at the end of the round.

For the upper bound, consider the same case like for the lower bound where n is odd, except the fact that none of the elected nodes are Byzantine and that all nodes p_i for all $\lfloor (n-1)/2 \rfloor - \lfloor k/2 \rfloor \leq i \leq \lfloor (n-1)/2 \rfloor + \lfloor k/2 \rfloor$ receive the same synchronization messages. However, the f Byzantine nodes then can send different synchronization information to the other nodes. For this, we generally assume that the nodes p_i for all $0 \leq i < f_1$ and $n - f_2 - 2 \leq i < n$ with $f_1 + f_2 = f$ are faulty. The worst case scenario then occurs when all correct nodes p_i, $i < \lfloor (n-1)/2 \rfloor - \lfloor k/2 \rfloor$, receive different synchronization messages from the set of Byzantine nodes $\{p_{n-f_2-2}, p_{n-f_2-1}, \ldots, p_{n-1}\}$ such that $\varphi_{n-f_2-2} < \varphi_{n-f_2-1} < \cdots < \varphi_{n-1} < \varphi_0 < \varphi_1 < \cdots < \varphi_{n-f_2-3}$ and all correct nodes p_j, $j > \lfloor (n-1)/2 \rfloor + \lfloor k/2 \rfloor$ receive different synchronization messages from the set of Byzantine nodes $\{p_0, p_1, \ldots, p_{f_1-1}\}$ such that $\varphi_{f_1} < \varphi_{f_1+1} < \cdots < \varphi_{n-1} < \varphi_0 < \varphi_1 < \ldots \varphi_{f_1-1}$. As a result, all nodes p_i with $\lfloor (n-1)/2 \rfloor - \lfloor k/2 \rfloor - f_2 \leq i \leq \lfloor (n-1)/2 \rfloor + \lfloor k/2 \rfloor + f_1$ can become passive and corresponds to number of $\lfloor k/2 \rfloor \cdot 2 + f + 1$ nodes. Similarly, if n is even, at most $\lfloor (k+1)/2 \rfloor \cdot 2 + f$ nodes can become passive. Since $k + f + 2 \geq \max(\lfloor k/2 \rfloor \cdot 2 + f + 1, \lfloor (k+1)/2 \rfloor \cdot 2 + f)$, generally at most $k + f + 2$ nodes can become passive at the end of the round. □

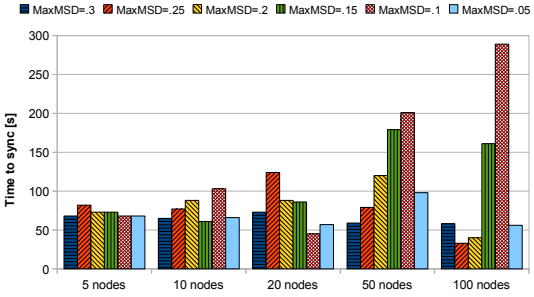
APPENDIX D

Simulation Results

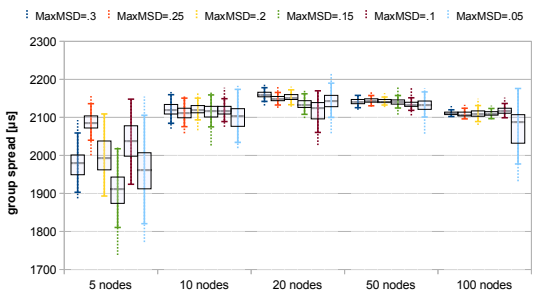
D.1 Fault-free Single-hop System



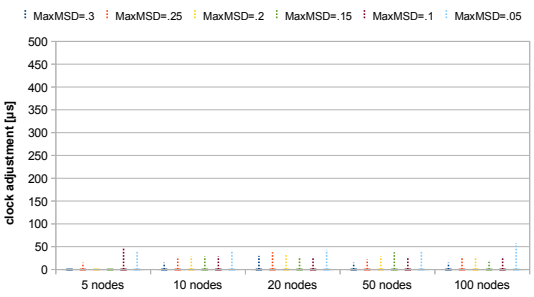
(a) Time to sync diagram (Randomized evaluation)



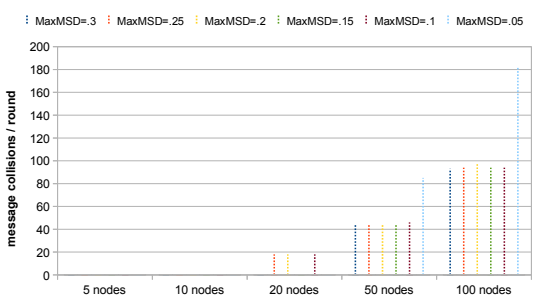
(b) Time to sync diagram (Prepared evaluation)



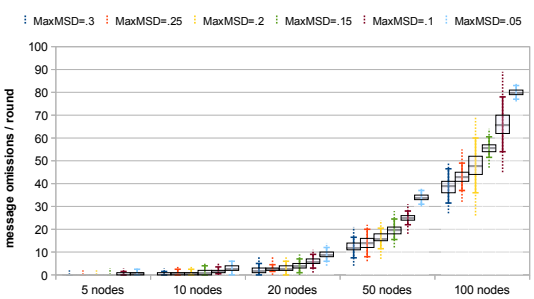
(c) Group spread diagram



(d) Clock adjustment diagram

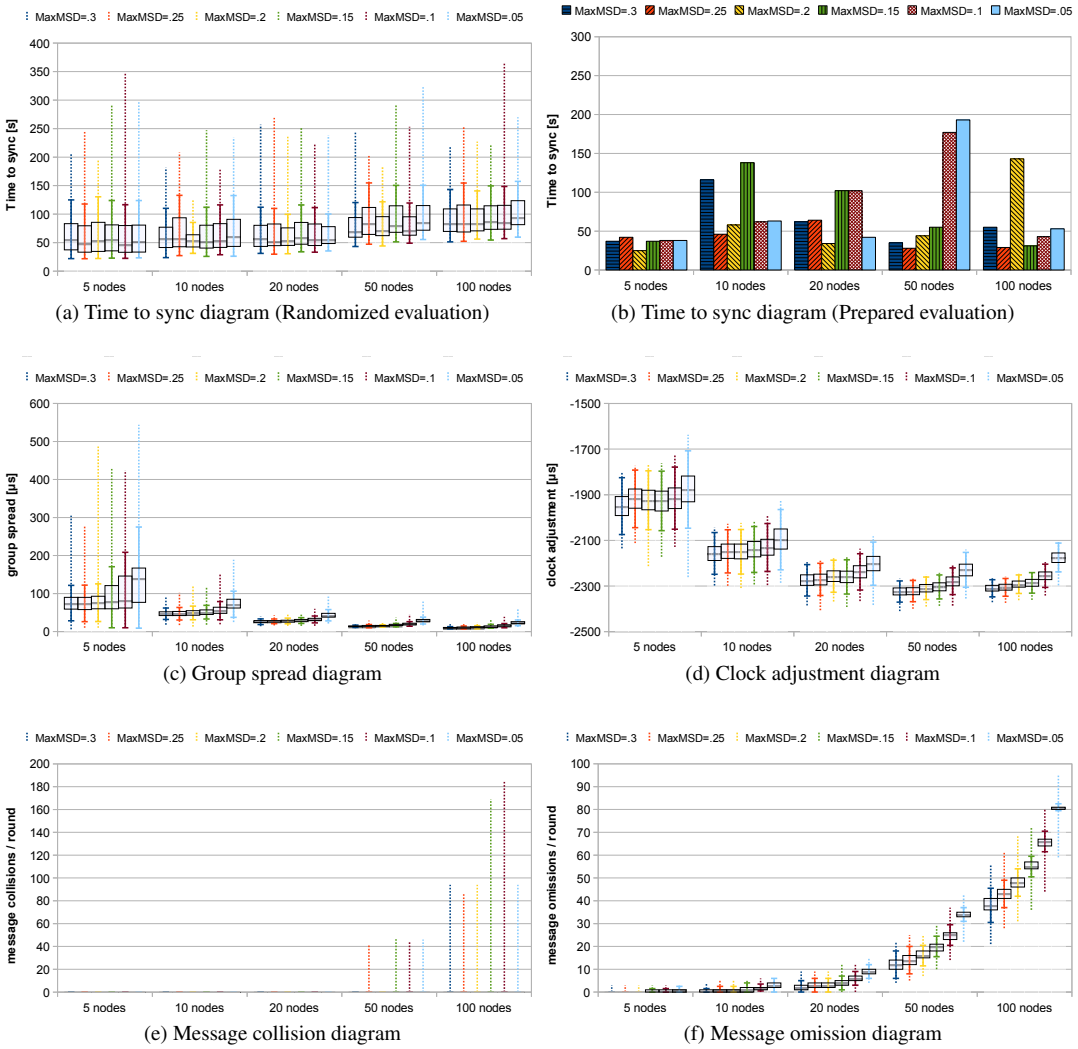


(e) Message collision diagram



(f) Message omission diagram

Figure D.1: Results according to algorithm version V1 in a fault-free all-to-all topology.



(a) Time to sync diagram (Randomized evaluation)

(b) Time to sync diagram (Prepared evaluation)

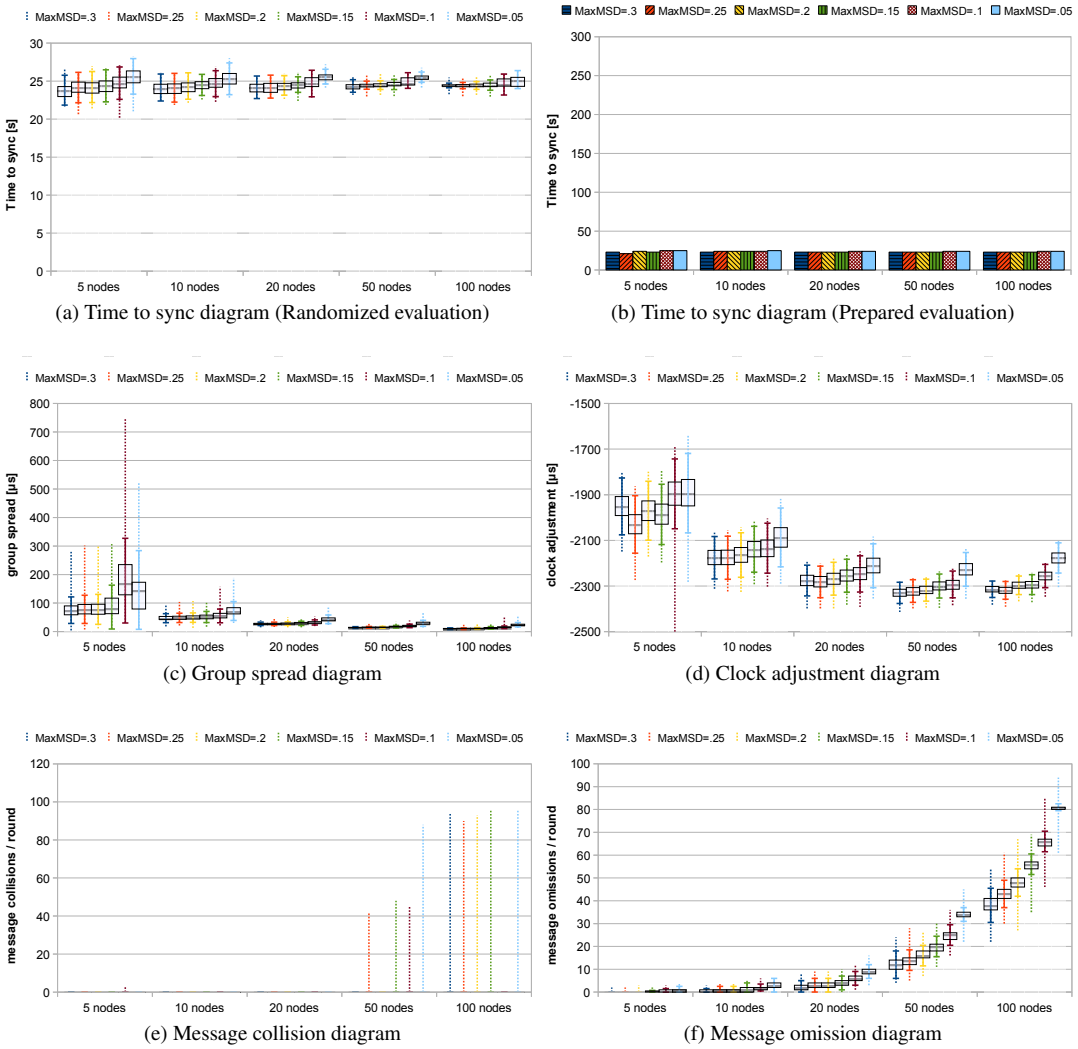
(c) Group spread diagram

(d) Clock adjustment diagram

(e) Message collision diagram

(f) Message omission diagram

Figure D.2: Different quality aspects of algorithm version V2 in a fault-free all-to-all topology.



(a) Time to sync diagram (Randomized evaluation)

(b) Time to sync diagram (Prepared evaluation)

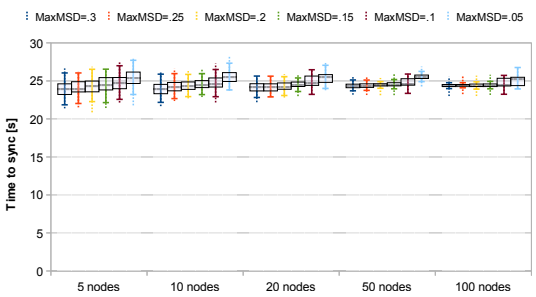
(c) Group spread diagram

(d) Clock adjustment diagram

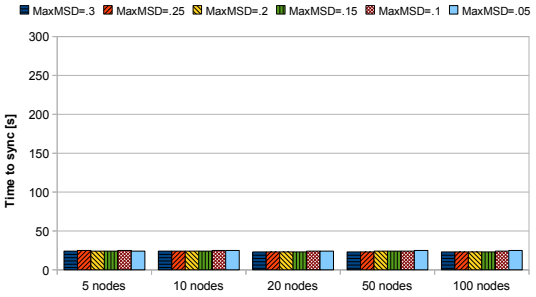
(e) Message collision diagram

(f) Message omission diagram

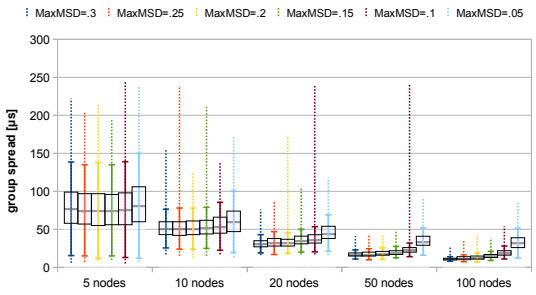
Figure D.3: Different quality aspects of algorithm version V3 in a fault-free all-to-all topology.



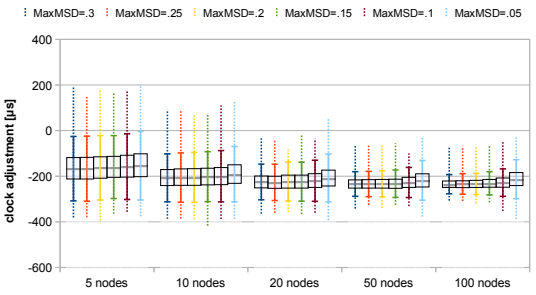
(a) Time to sync diagram (Randomized evaluation)



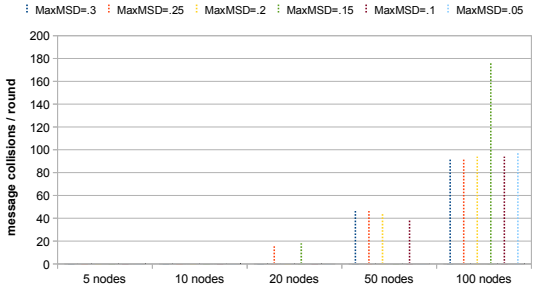
(b) Time to sync diagram (Prepared evaluation)



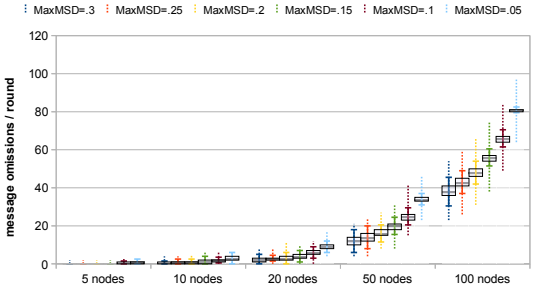
(c) Group spread diagram



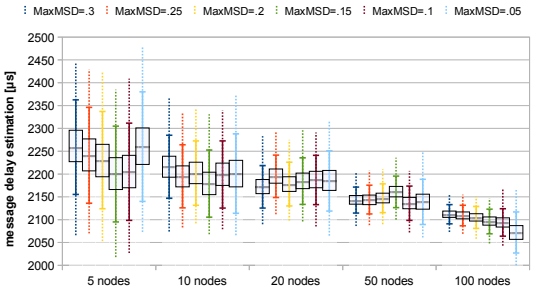
(d) Clock adjustment diagram



(e) Message collision diagram

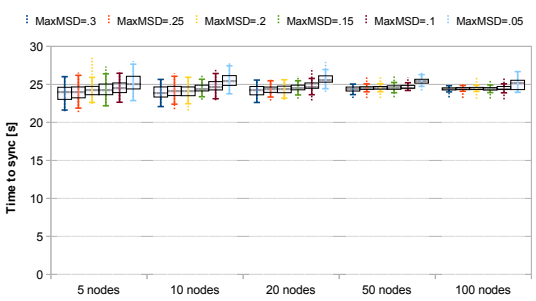


(f) Message omission diagram

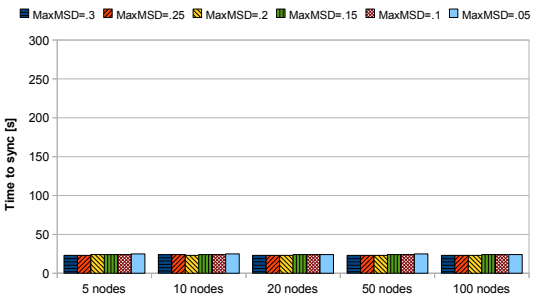


(g) Message delay estimation diagram

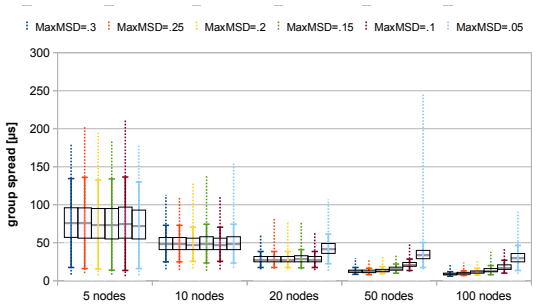
Figure D.4: Different quality aspects of algorithm version V4 in a fault-free all-to-all topology.



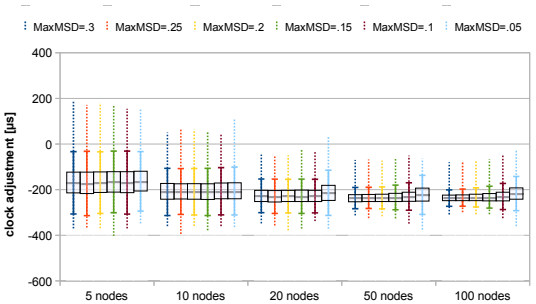
(a) Time to sync diagram (Randomized evaluation)



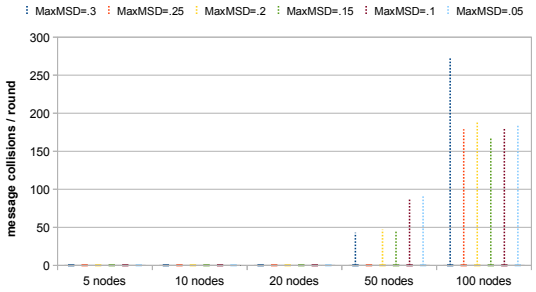
(b) Time to sync diagram (Prepared evaluation)



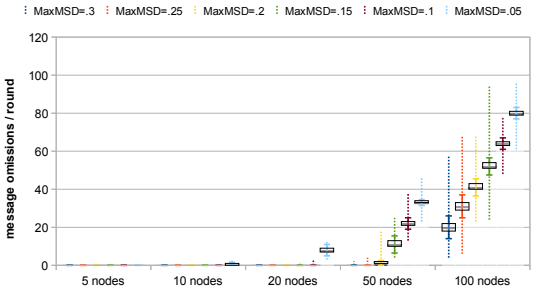
(c) Group spread diagram



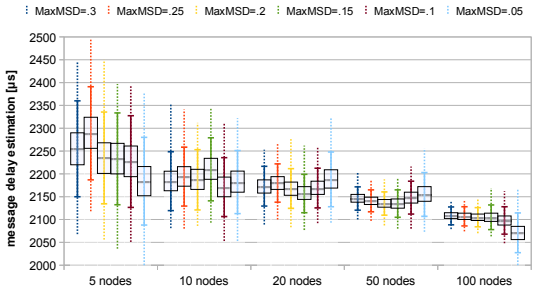
(d) Clock adjustment diagram



(e) Message collision diagram

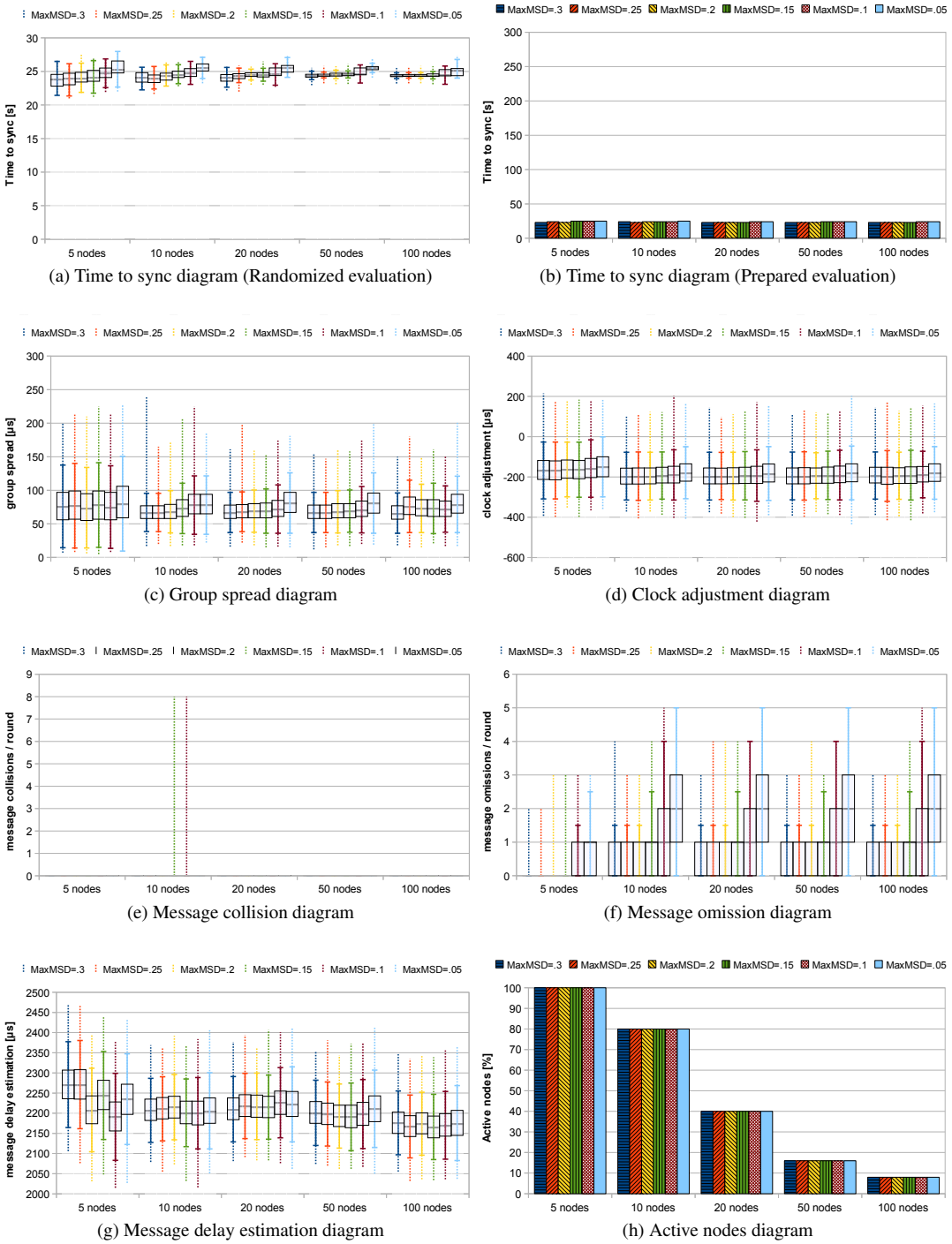


(f) Message omission diagram



(g) Message delay estimation diagram

Figure D.5: Different quality aspects of algorithm version V5 in a fault-free all-to-all topology.



(a) Time to sync diagram (Randomized evaluation)

(b) Time to sync diagram (Prepared evaluation)

(c) Group spread diagram

(d) Clock adjustment diagram

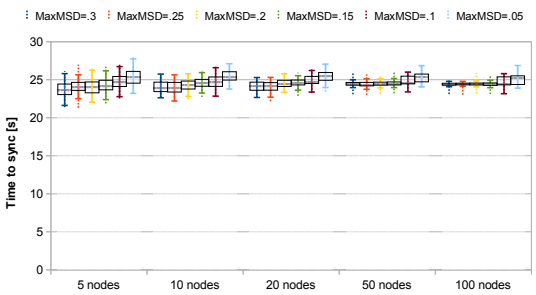
(e) Message collision diagram

(f) Message omission diagram

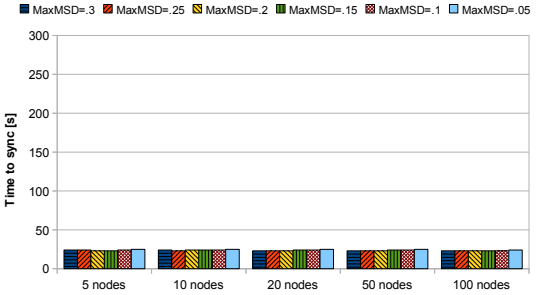
(g) Message delay estimation diagram

(h) Active nodes diagram

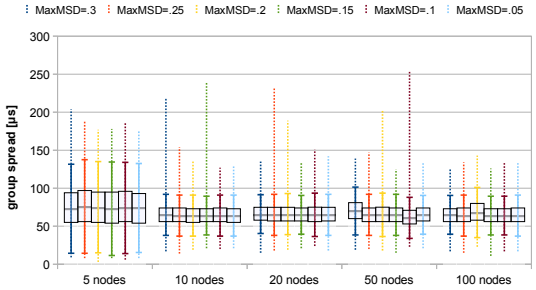
Figure D.6: Different quality aspects of algorithm version V6 in a fault-free all-to-all topology.



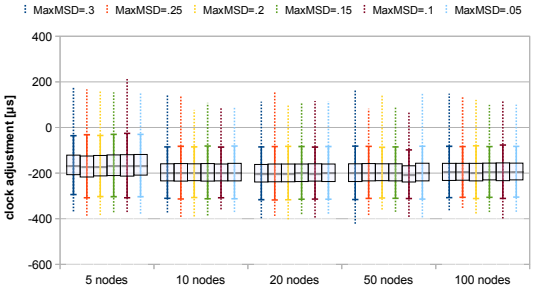
(a) Time to sync diagram (Randomized evaluation)



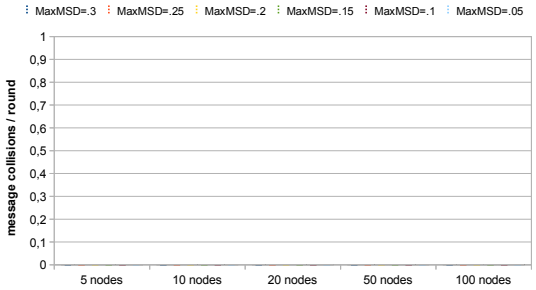
(b) Time to sync diagram (Prepared evaluation)



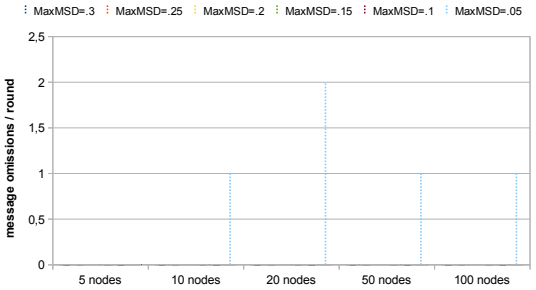
(c) Group spread diagram



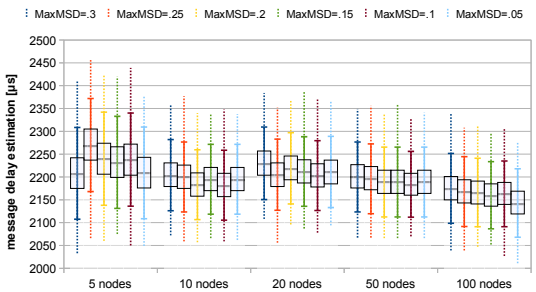
(d) Clock adjustment diagram



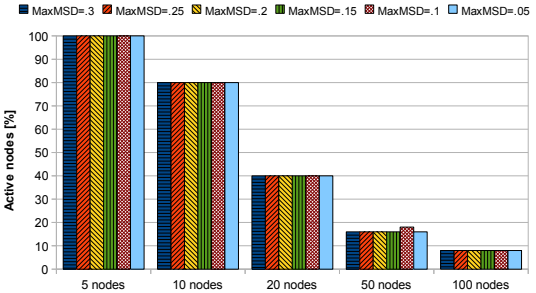
(e) Message collision diagram



(f) Message omission diagram



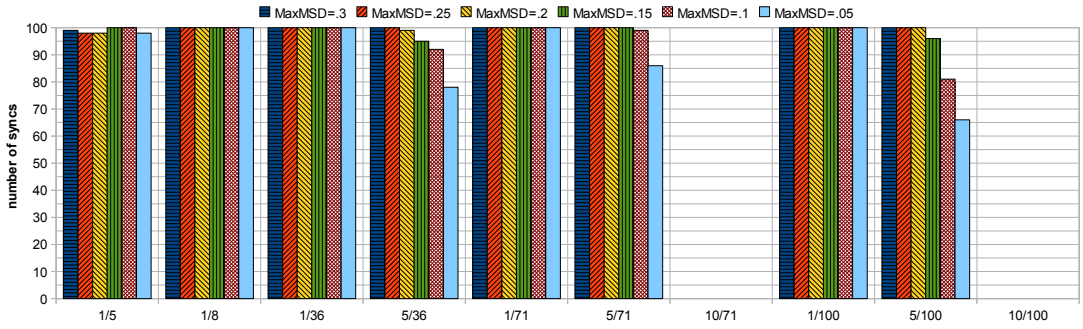
(g) Message delay estimation diagram



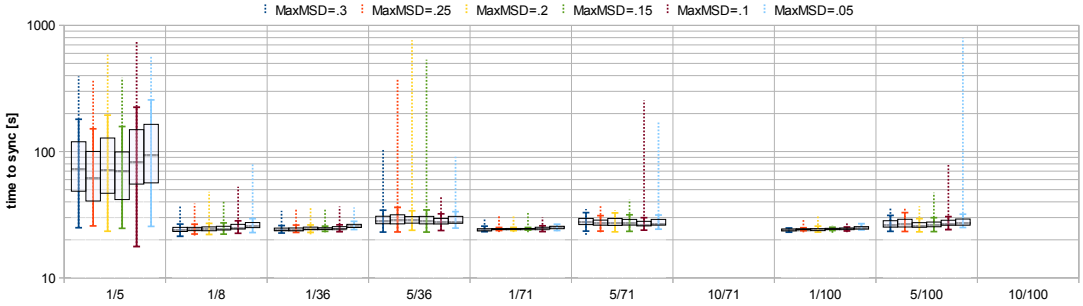
(h) Active nodes diagram

Figure D.7: Different quality aspects of algorithm version V7 in a fault-free all-to-all topology.

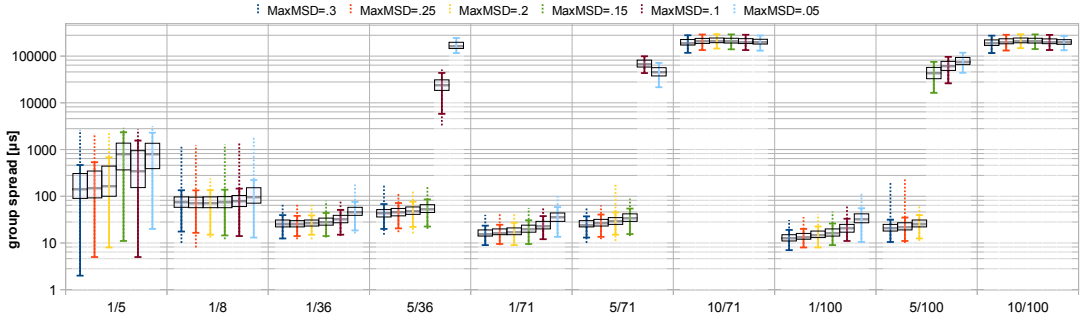
D.2 Coherent Single-hop System



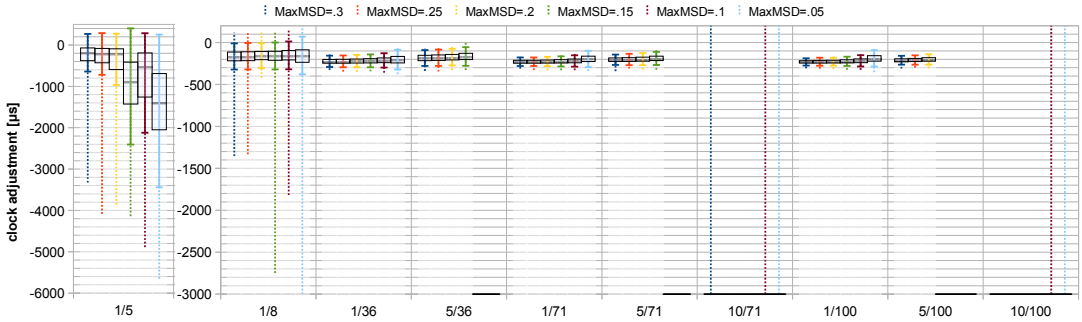
(a) Number of achieved syncs until 1000s in 100 randomized simulations



(b) Time to sync diagram (Randomized evaluation)

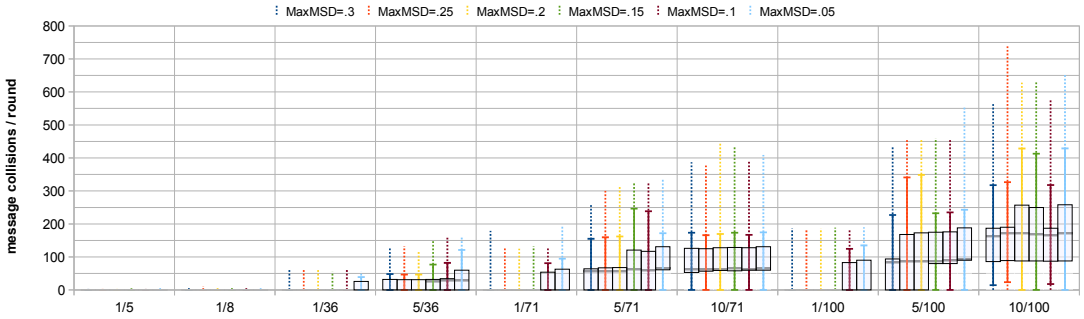


(c) Group spread diagram

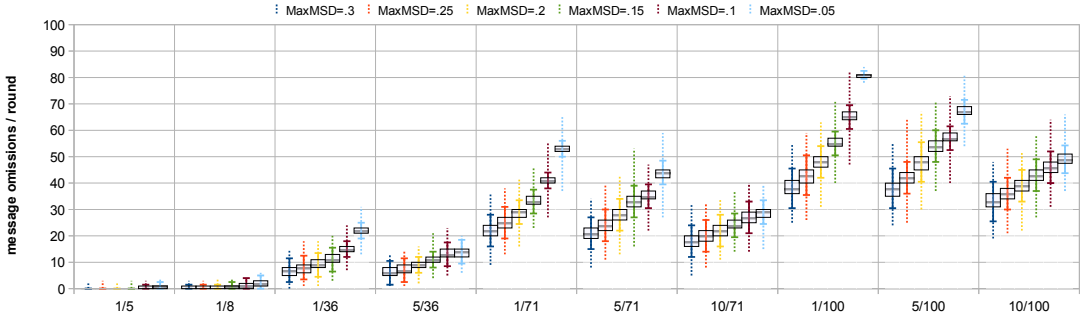


(d) Clock adjustment diagram

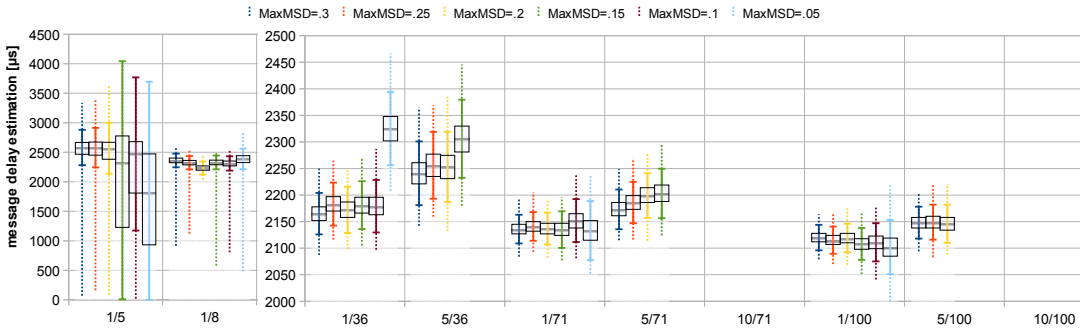
Figure D.8: Different quality aspects of algorithm version V4 in a faulty all-to-all topology.



(e) Message collision diagram

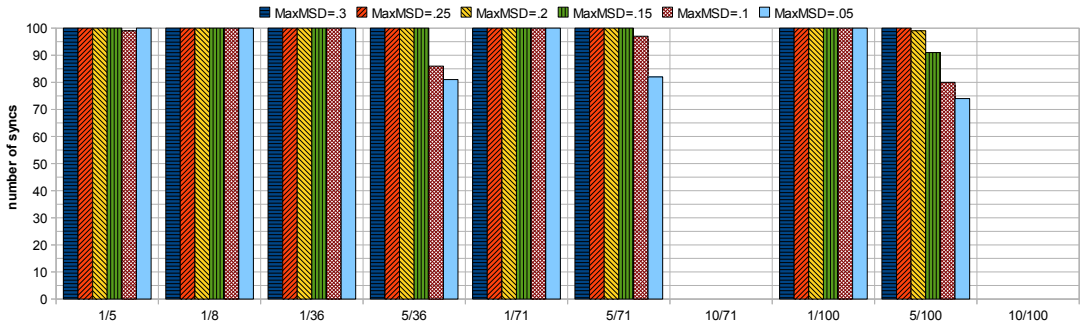


(f) Message omission diagram

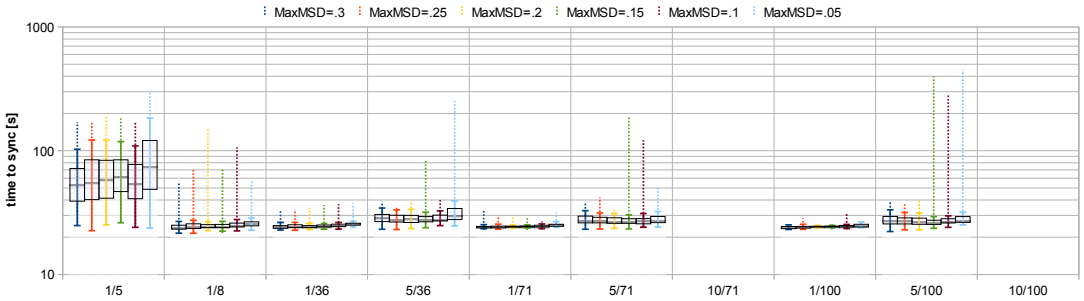


(g) Message delay estimation diagram

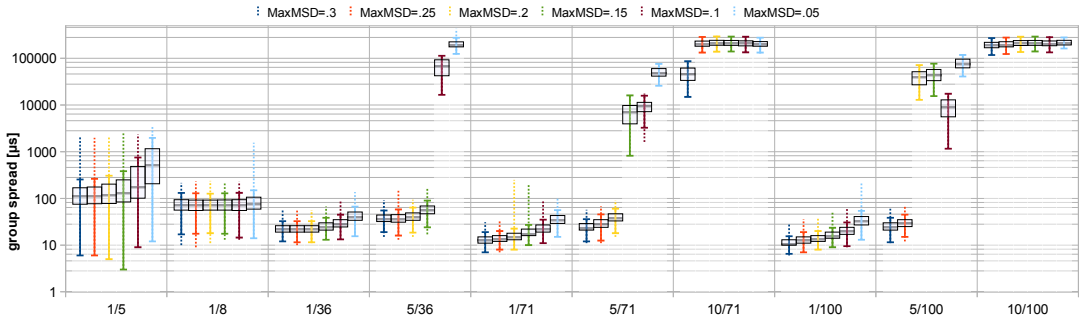
Figure D.8: Different quality aspects of algorithm version V4 in a faulty all-to-all topology.



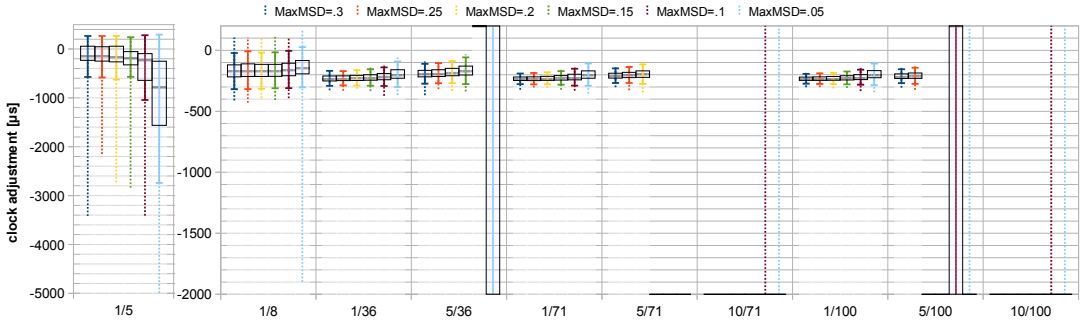
(a) Number of achieved syncs until 1000s in 100 randomized simulations



(b) Time to sync diagram (Randomized evaluation)

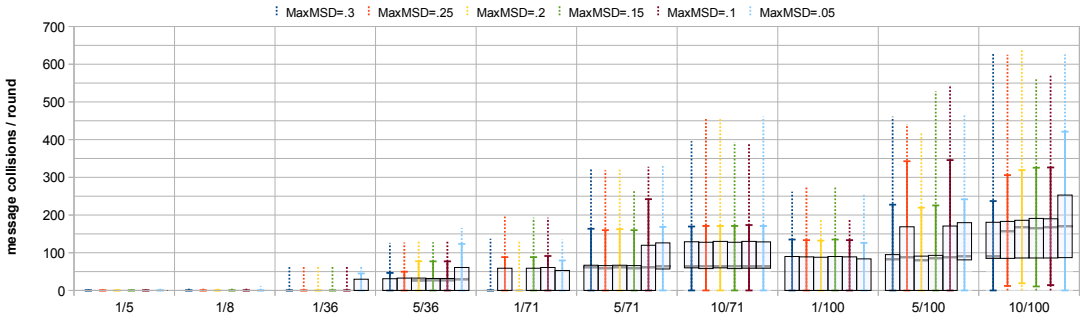


(c) Group spread diagram

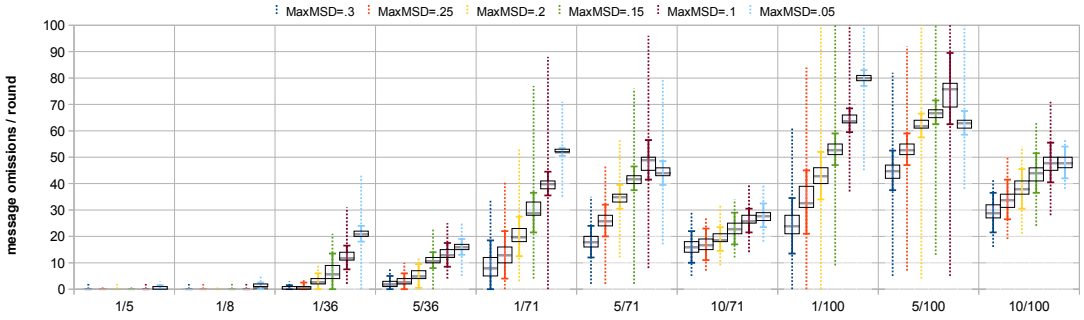


(d) Clock adjustment diagram

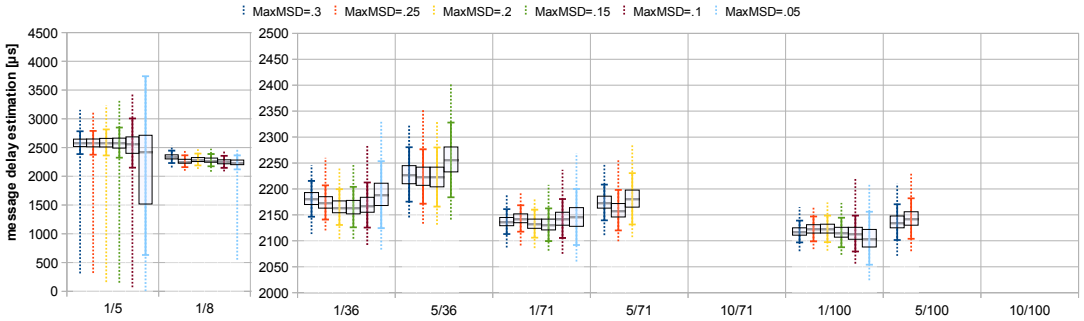
Figure D.9: Different quality aspects of algorithm version V5 in a faulty all-to-all topology.



(e) Message collision diagram

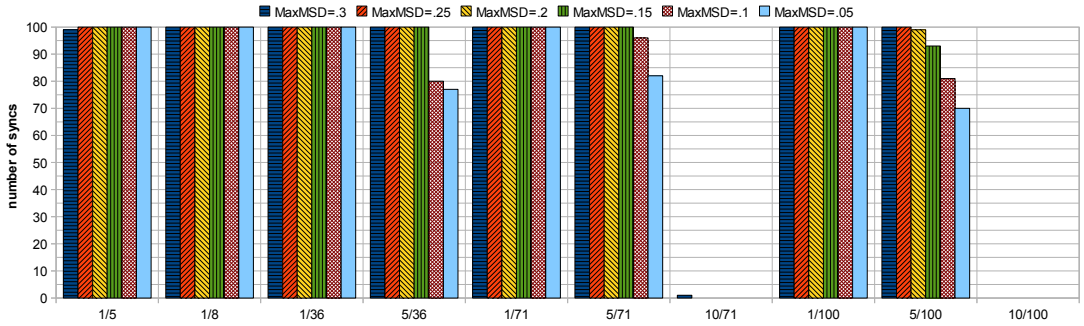


(f) Message omission diagram

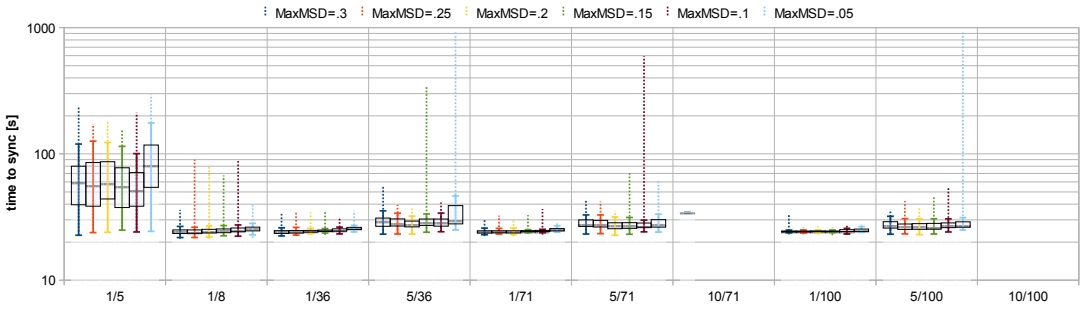


(g) Message delay estimation diagram

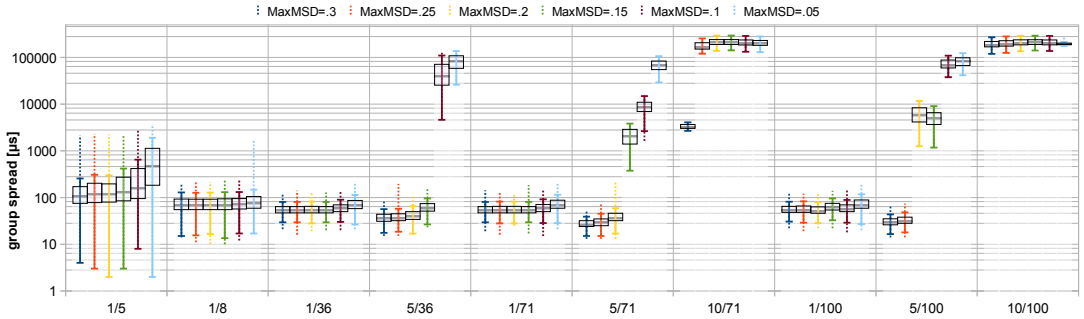
Figure D.9: Different quality aspects of algorithm version V5 in a faulty all-to-all topology.



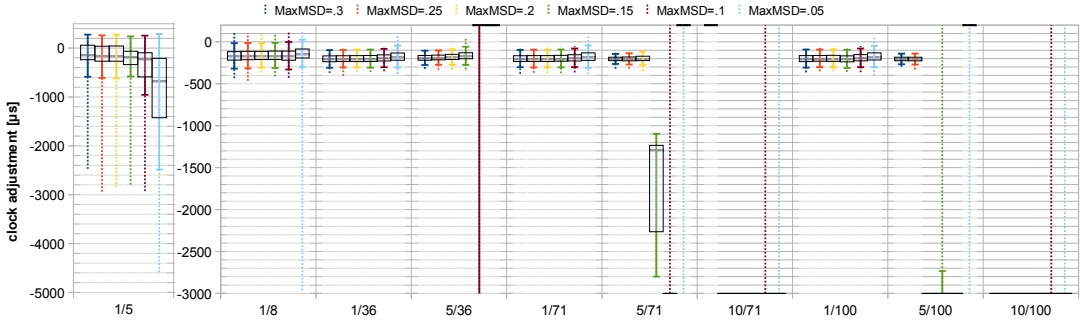
(a) Number of achieved syncs until 1000s in 100 randomized simulations



(b) Time to sync diagram (Randomized evaluation)

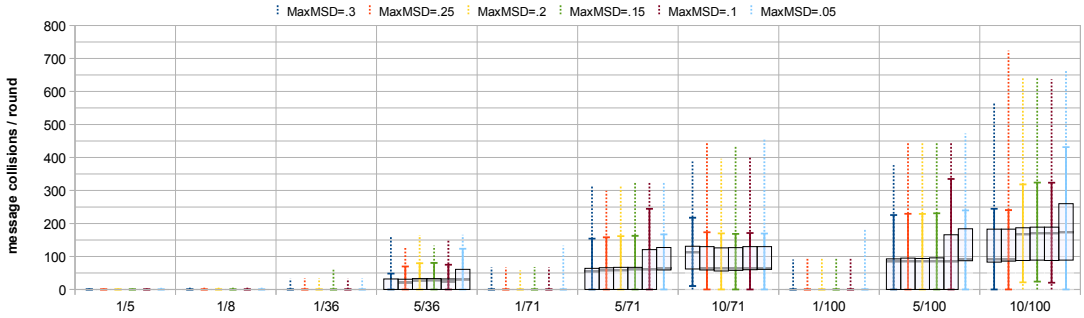


(c) Group spread diagram

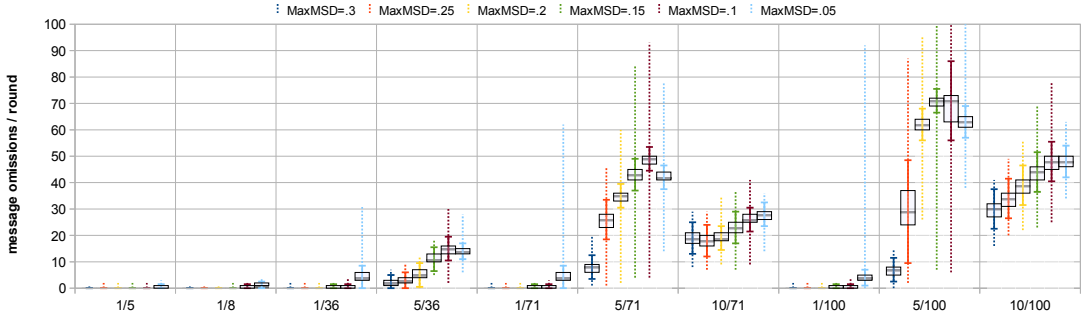


(d) Clock adjustment diagram

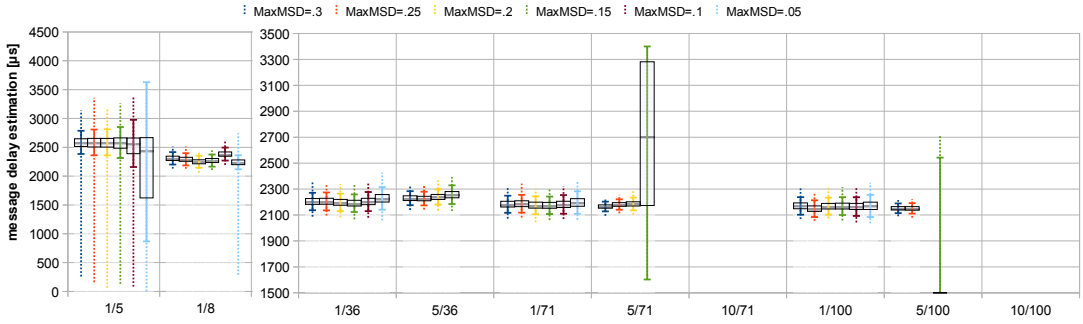
Figure D.10: Different quality aspects of algorithm version V7 in a faulty all-to-all topology.



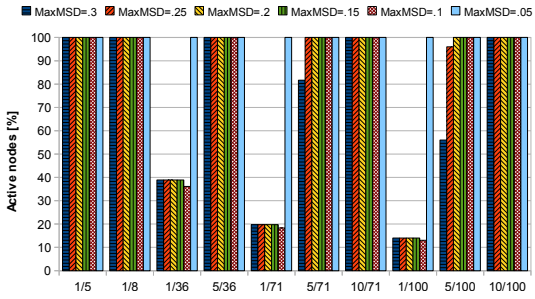
(e) Message collision diagram



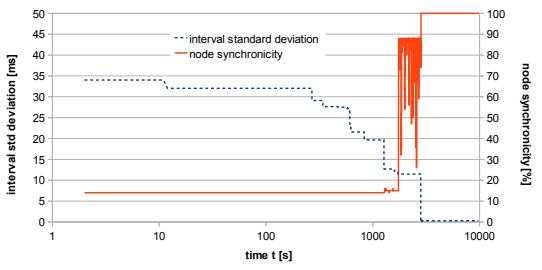
(f) Message omission diagram



(g) Message delay estimation diagram



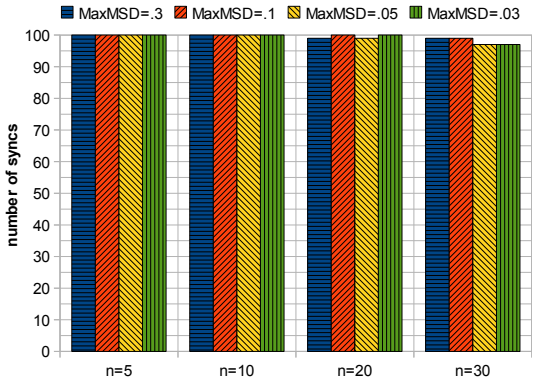
(h) Active nodes diagram



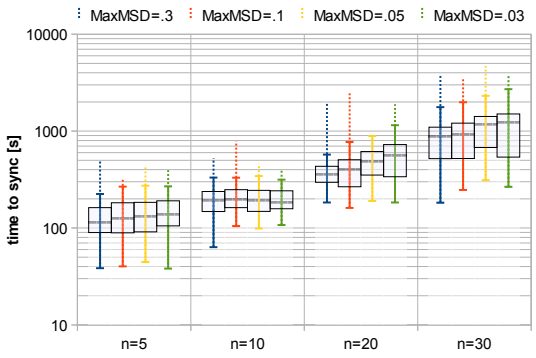
(i) Interval deviation diagram of a network comprising 71 nodes, where 10 nodes are Byzantine faulty, and a maximum relative message staggering delay of 0.3.

Figure D.10: Different quality aspects of algorithm version V7 in a faulty all-to-all topology.

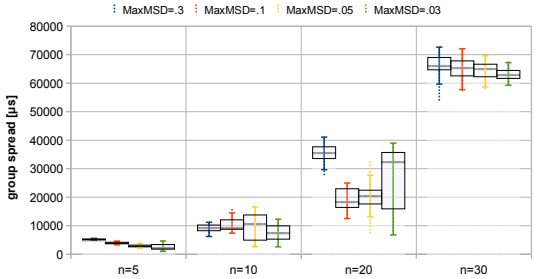
D.3 Fault-free Chain-structured Multi-hop System



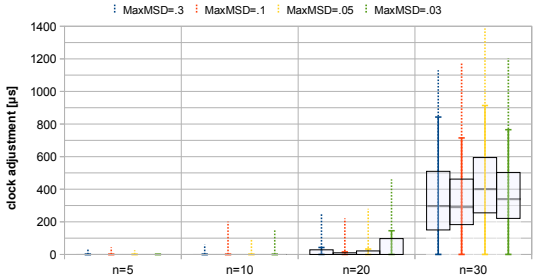
(a) Number of achieved syncs until 5000s in 100 randomized simulations



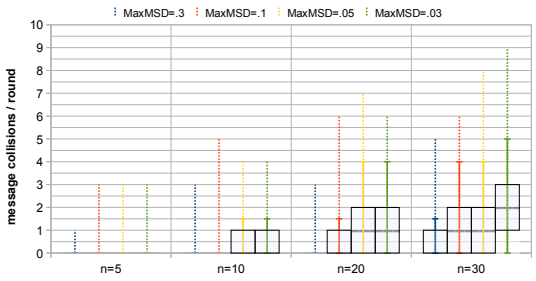
(b) Time to sync diagram (Randomized evaluation)



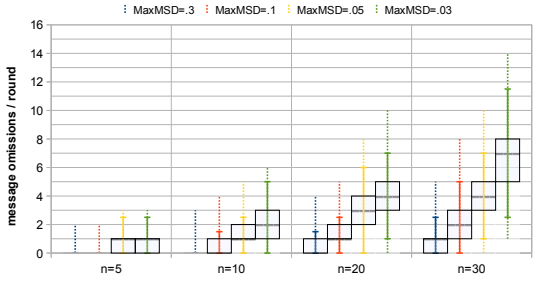
(c) Group spread diagram



(d) Clock adjustment diagram

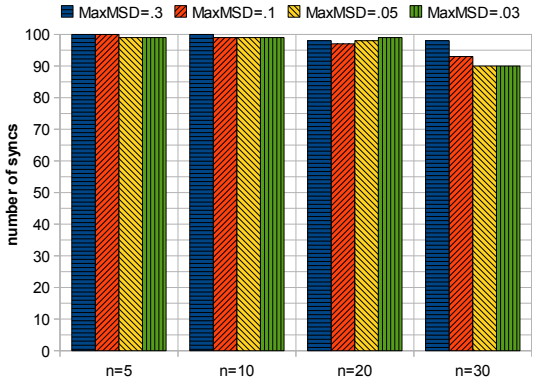


(e) Message collision diagram

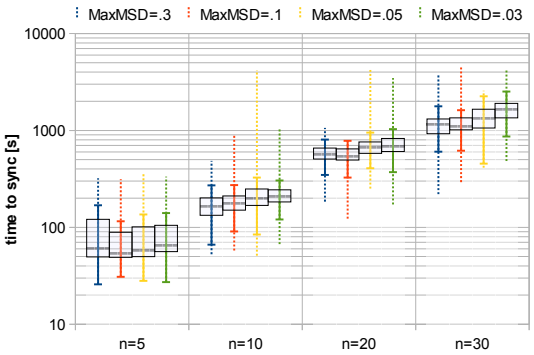


(f) Message omission diagram

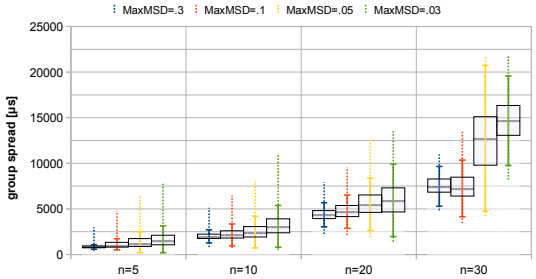
Figure D.11: Different quality aspects of algorithm version V1 in a fault-free chain topology.



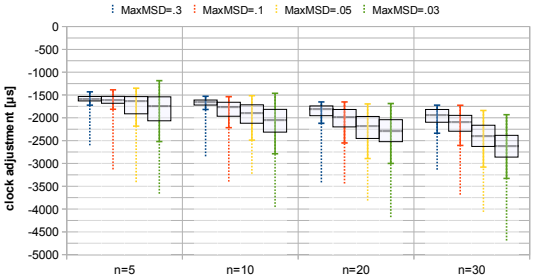
(a) Number of achieved syncs until 5000s in 100 randomized simulations



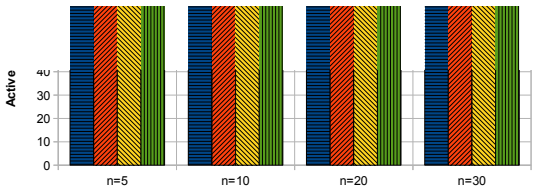
(b) Time to sync diagram (Randomized evaluation)



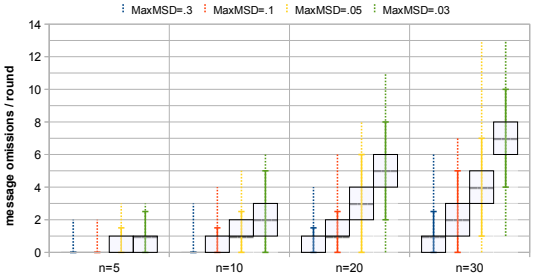
(c) Group spread diagram



(d) Clock adjustment diagram

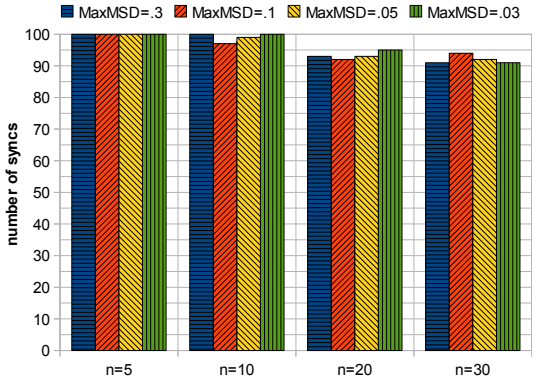


(e) Message collision diagram

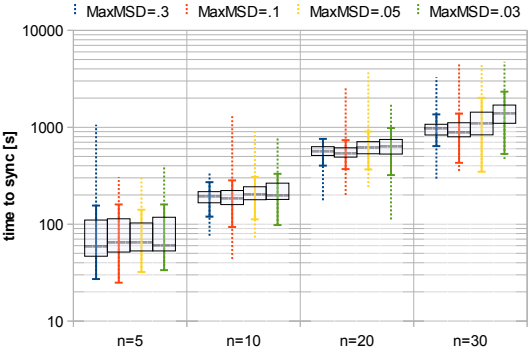


(f) Message omission diagram

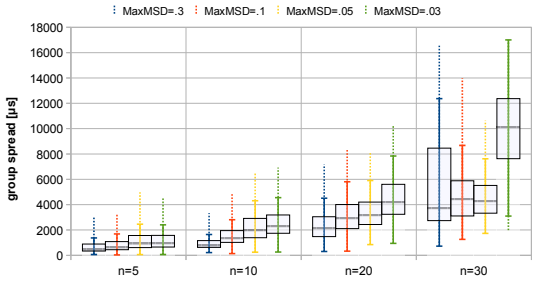
Figure D.12: Different quality aspects of algorithm version V2 in a fault-free chain topology.



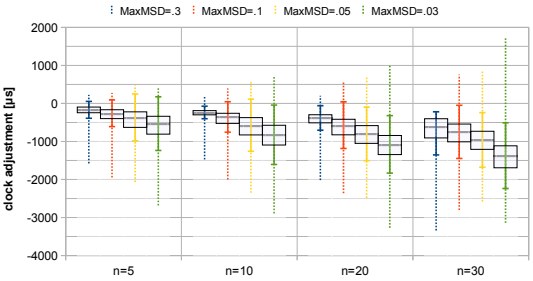
(a) Number of achieved syncs until 5000s in 100 randomized simulations



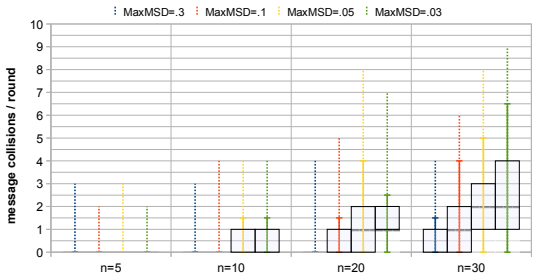
(b) Time to sync diagram (Randomized evaluation)



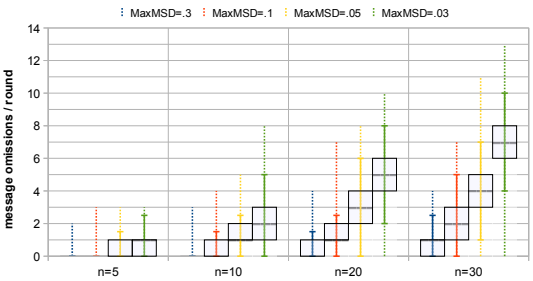
(c) Group spread diagram



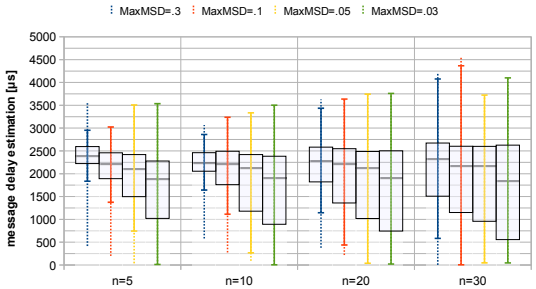
(d) Clock adjustment diagram



(e) Message collision diagram



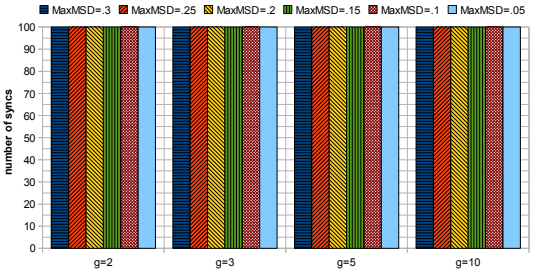
(f) Message omission diagram



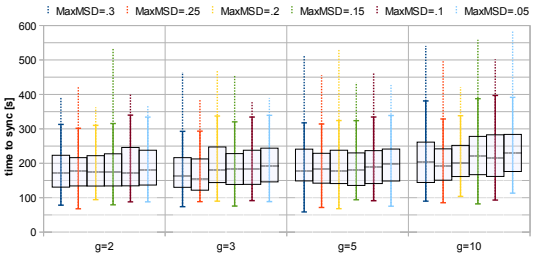
(g) Message delay estimation diagram

Figure D.13: Different quality aspects of algorithm version V8 in a fault-free chain topology.

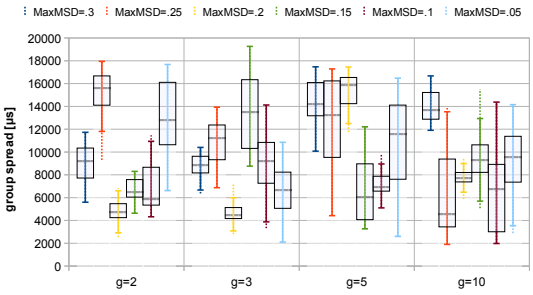
D.4 Fault-free Grouped Multi-hop System



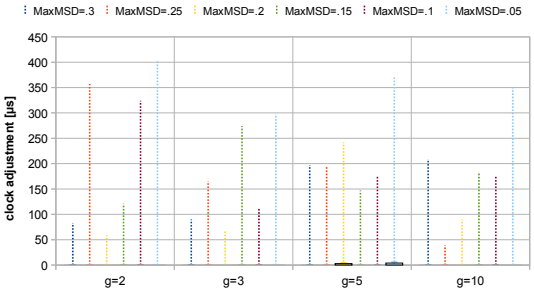
(a) Number of achieved syncs until 1000s in 100 randomized simulations



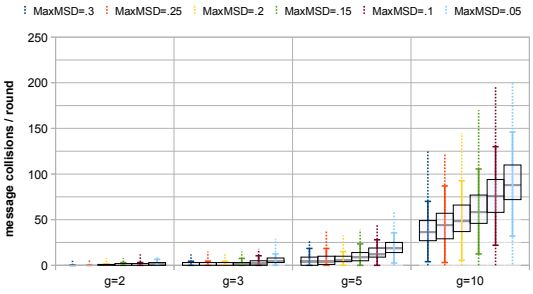
(b) Time to sync diagram (Randomized evaluation)



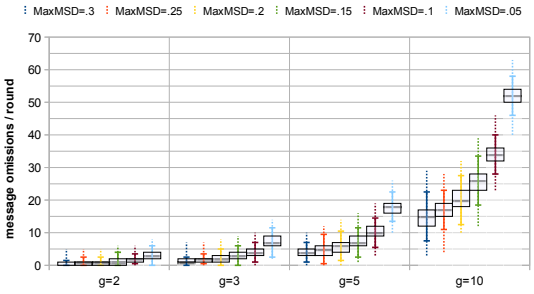
(c) Group spread diagram



(d) Clock adjustment diagram

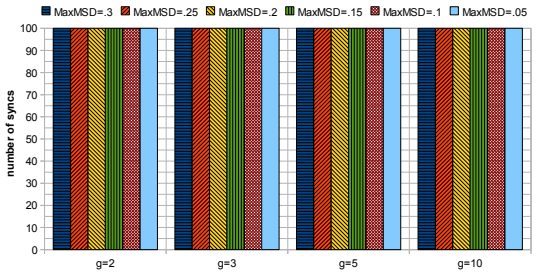


(e) Message collision diagram

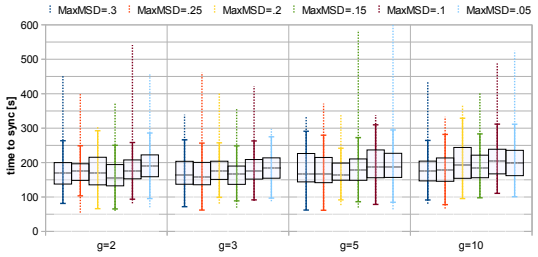


(f) Message omission diagram

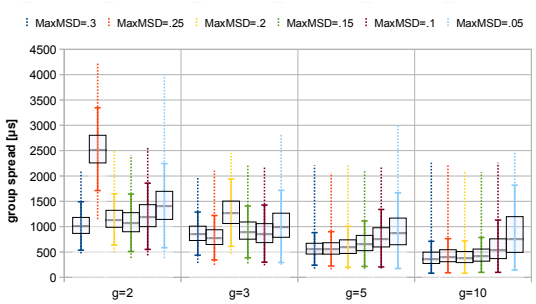
Figure D.14: Different quality aspects of algorithm version V1 in a fault-free grouped multi-hop topology.



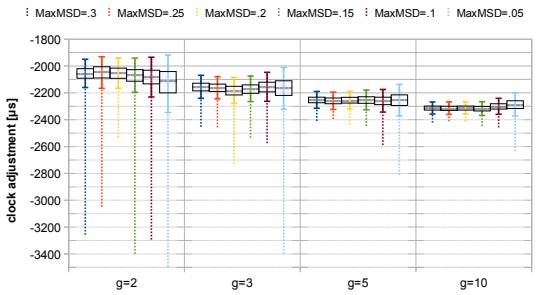
(a) Number of achieved syncs until 1000s in 100 randomized simulations



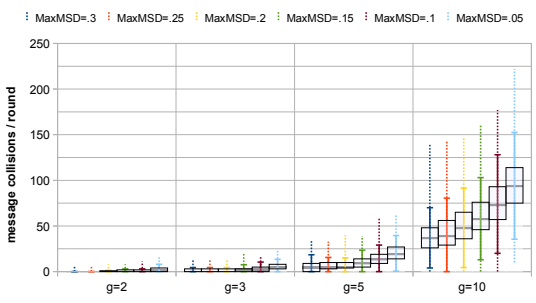
(b) Time to sync diagram (Randomized evaluation)



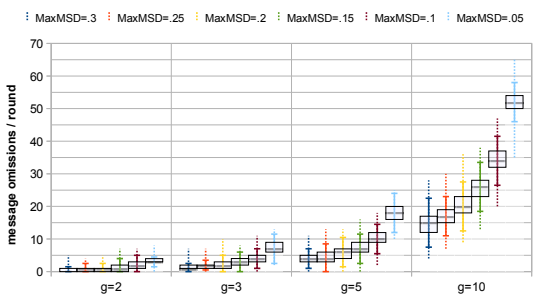
(c) Group spread diagram



(d) Clock adjustment diagram

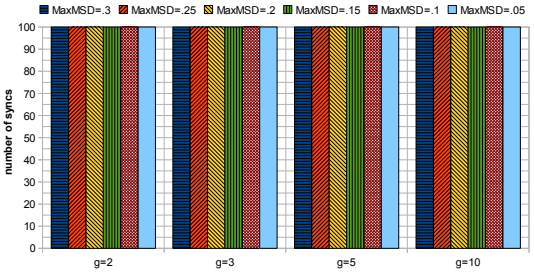


(e) Message collision diagram

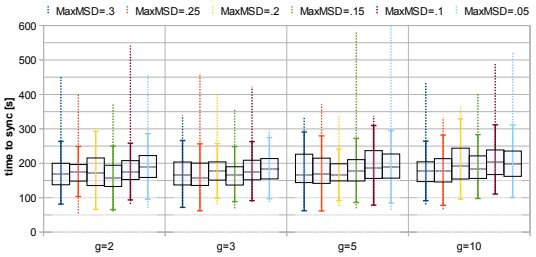


(f) Message omission diagram

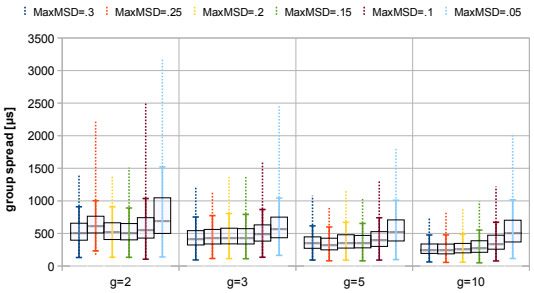
Figure D.15: Different quality aspects of algorithm version V2 in a fault-free grouped multi-hop topology.



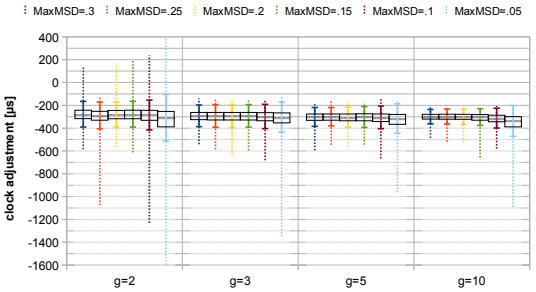
(a) Number of achieved syncs until 1000s in 100 randomized simulations



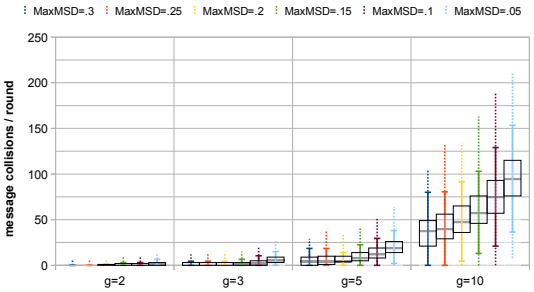
(b) Time to sync diagram (Randomized evaluation)



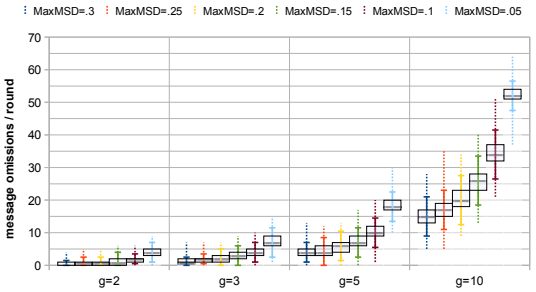
(c) Group spread diagram



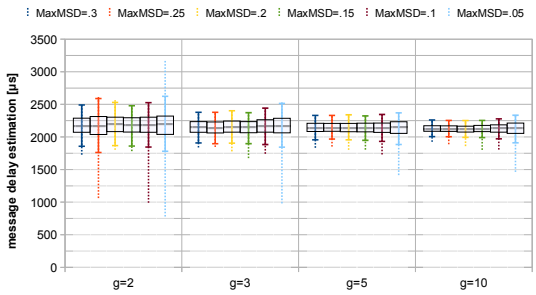
(d) Clock adjustment diagram



(e) Message collision diagram



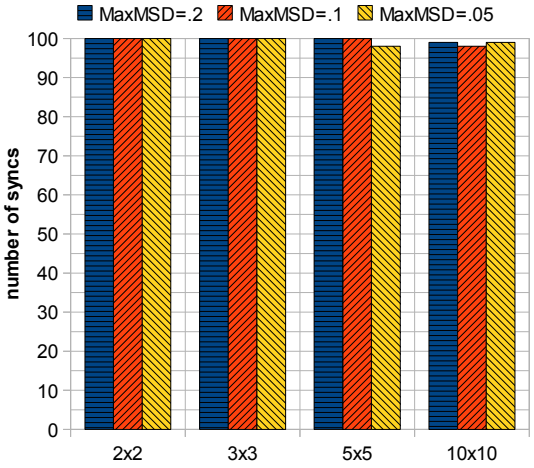
(f) Message omission diagram



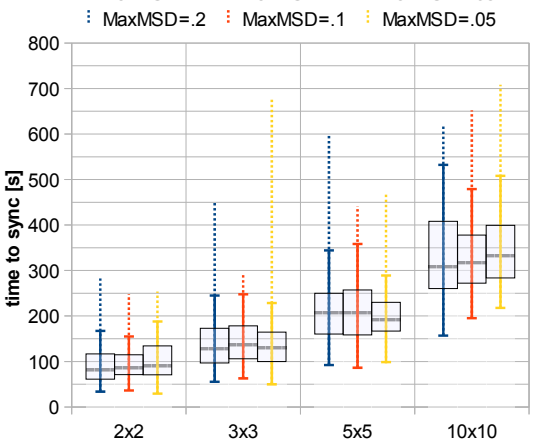
(g) Message delay estimation diagram

Figure D.16: Different quality aspects of algorithm version V8 in a fault-free grouped multi-hop topology.

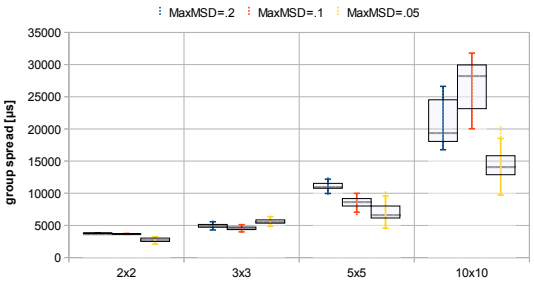
D.5 Fault-free Regular Grid-structured Multi-hop System



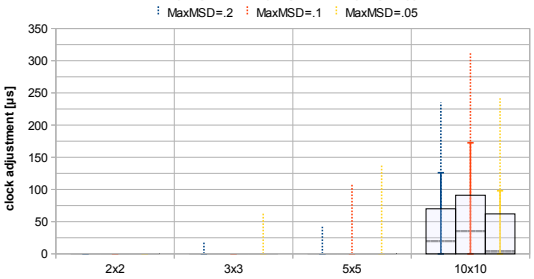
(a) Number of achieved syncs until 1000s in 100 randomized simulations



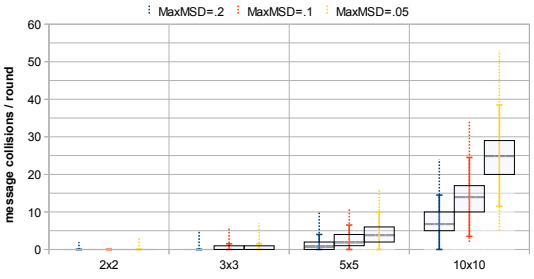
(b) Time to sync diagram (Randomized evaluation)



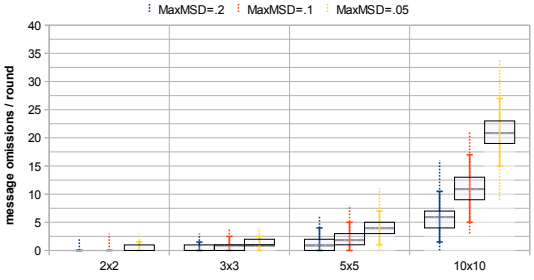
(c) Group spread diagram



(d) Clock adjustment diagram

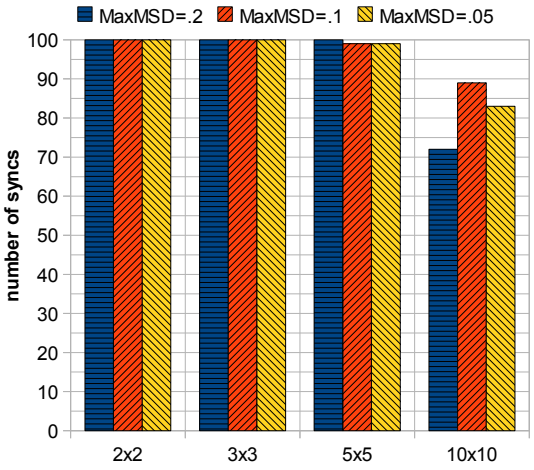


(e) Message collision diagram

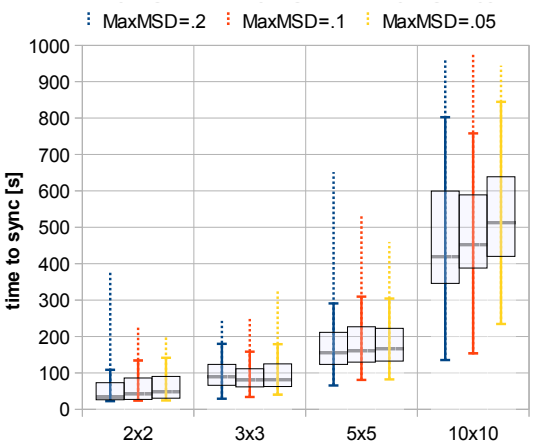


(f) Message omission diagram

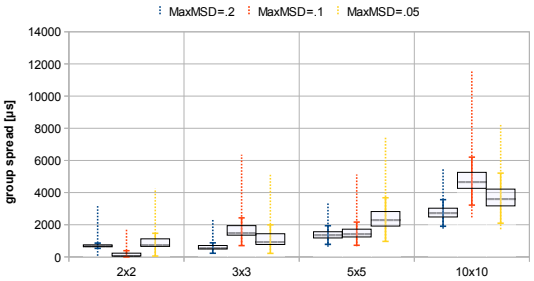
Figure D.17: Different quality aspects of algorithm version V1 in a fault-free regular grid topology.



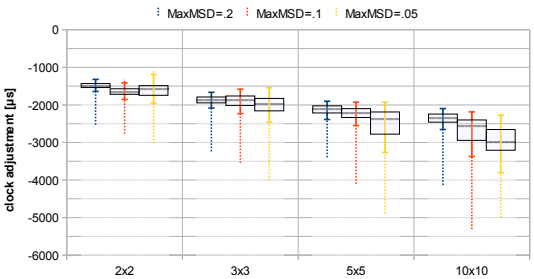
(a) Number of achieved syncs until 1000s in 100 randomized simulations



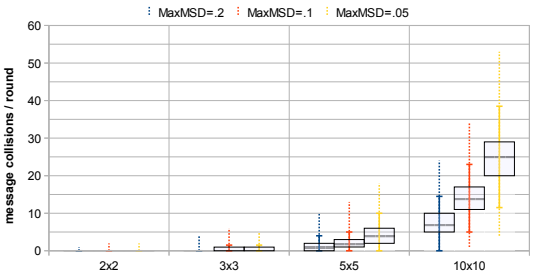
(b) Time to sync diagram (Randomized evaluation)



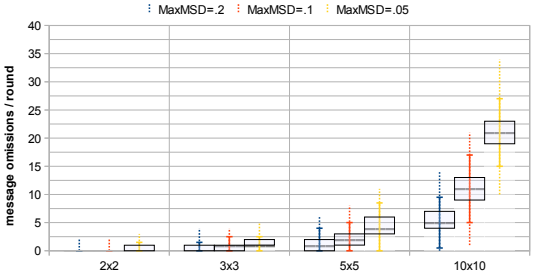
(c) Group spread diagram



(d) Clock adjustment diagram

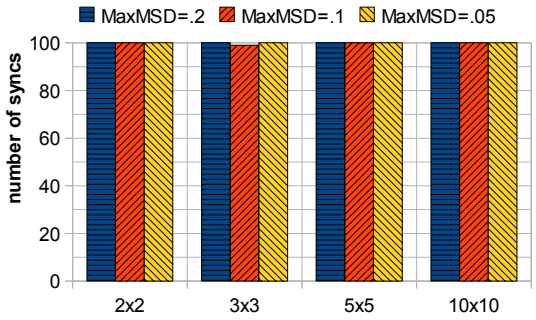


(e) Message collision diagram

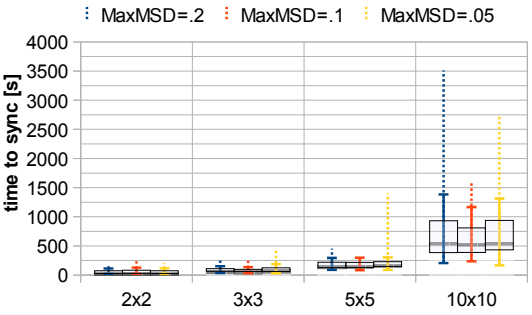


(f) Message omission diagram

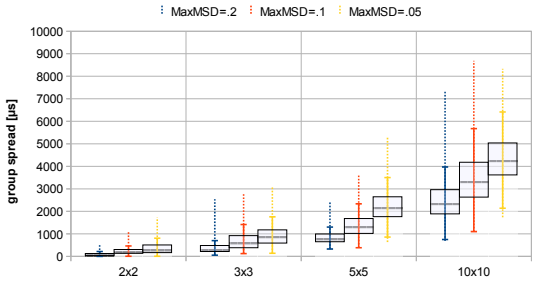
Figure D.18: Different quality aspects of algorithm version V2 in a fault-free regular grid topology.



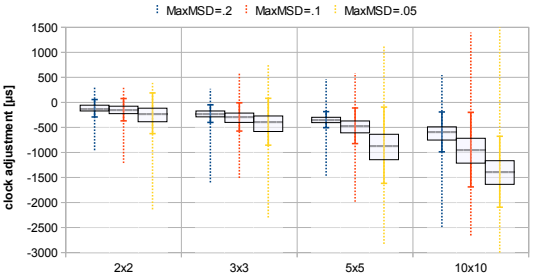
(a) Number of achieved syncs until 5000s in 100 randomized simulations



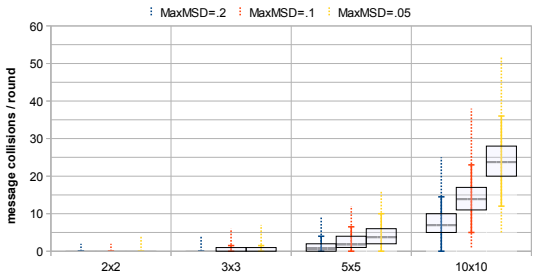
(b) Time to sync diagram (Randomized evaluation)



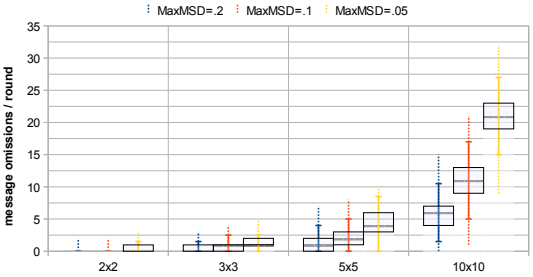
(c) Group spread diagram



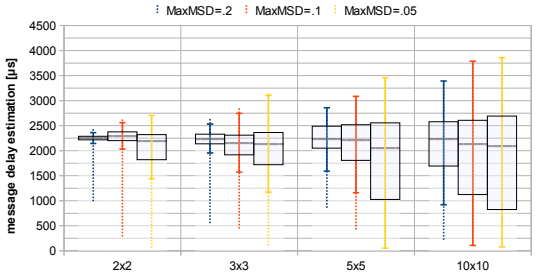
(d) Clock adjustment diagram



(e) Message collision diagram



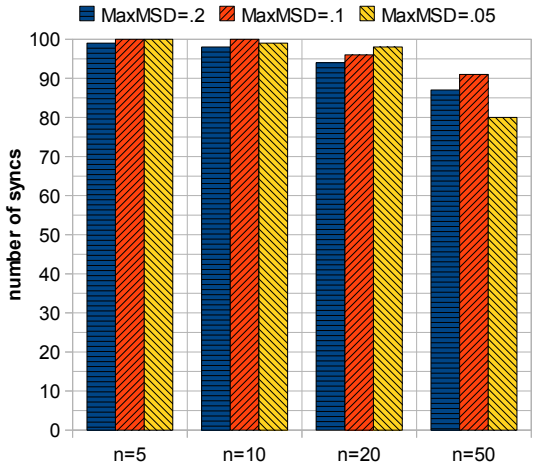
(f) Message omission diagram



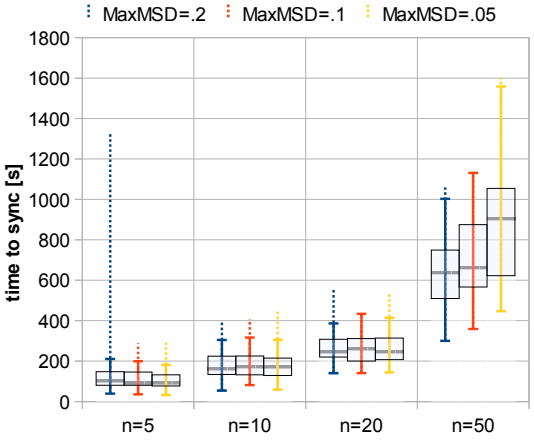
(g) Message delay estimation diagram

Figure D.19: Different quality aspects of algorithm version V8 in a fault-free regular grid topology.

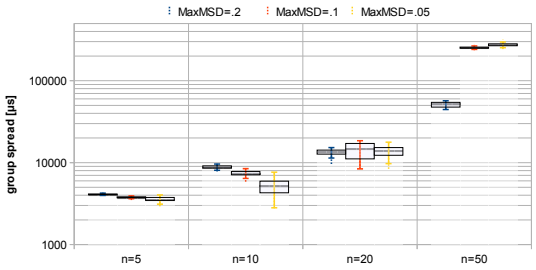
D.6 Fault-free Ring-structured Multi-hop System



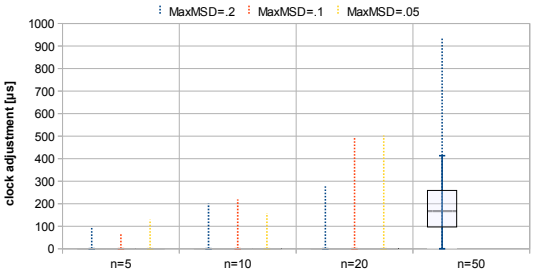
(a) Number of achieved syncs until 5000s in 100 randomized simulations



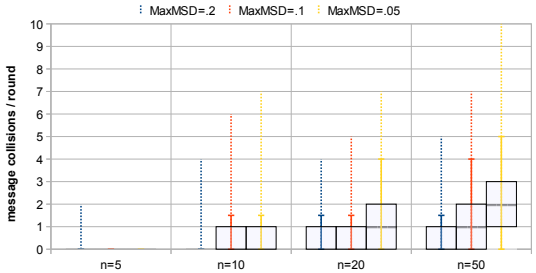
(b) Time to sync diagram (Randomized evaluation)



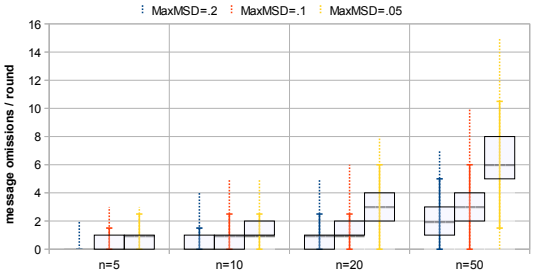
(c) Group spread diagram



(d) Clock adjustment diagram

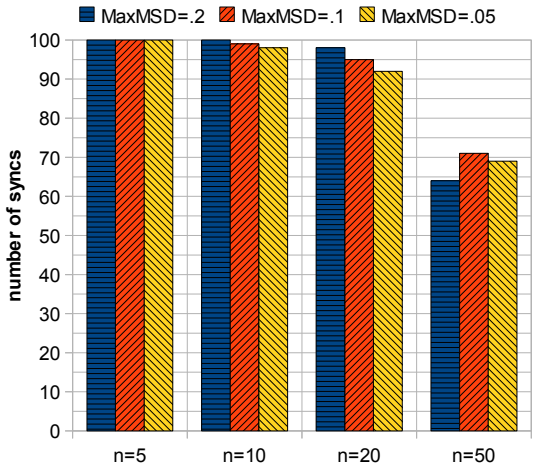


(e) Message collision diagram

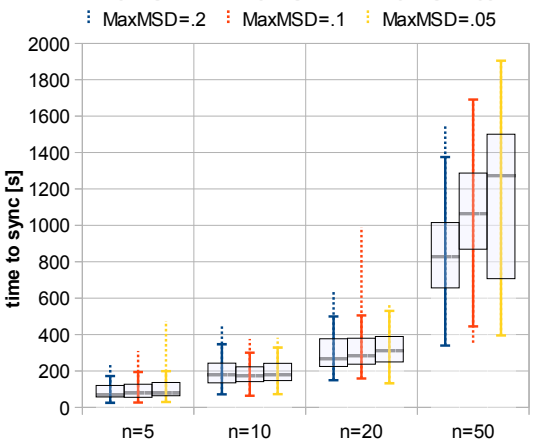


(f) Message omission diagram

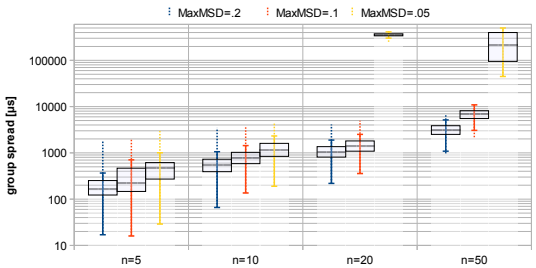
Figure D.20: Different quality aspects of algorithm version V1 in a fault-free ring topology.



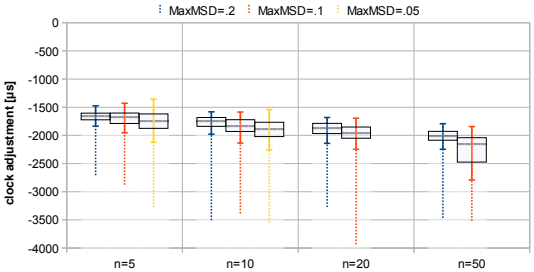
(a) Number of achieved syncs until 5000s in 100 randomized simulations



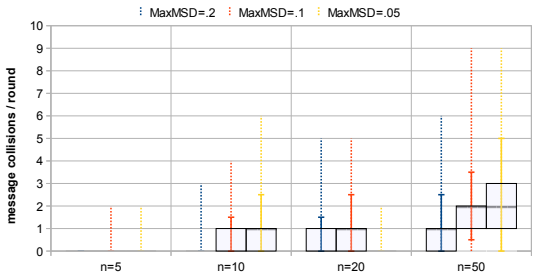
(b) Time to sync diagram (Randomized evaluation)



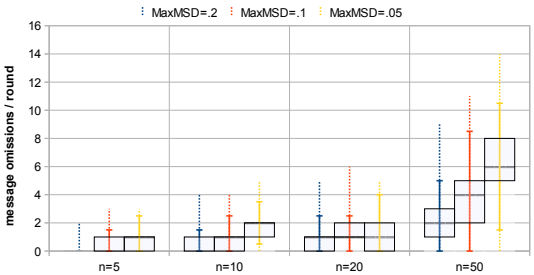
(c) Group spread diagram



(d) Clock adjustment diagram

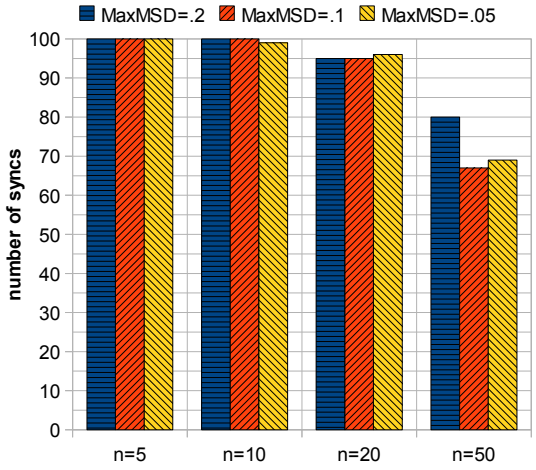


(e) Message collision diagram

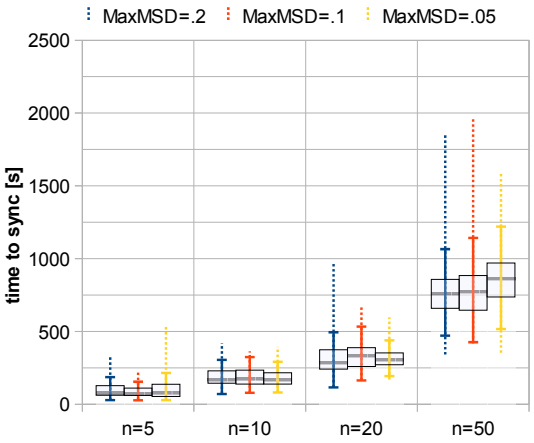


(f) Message omission diagram

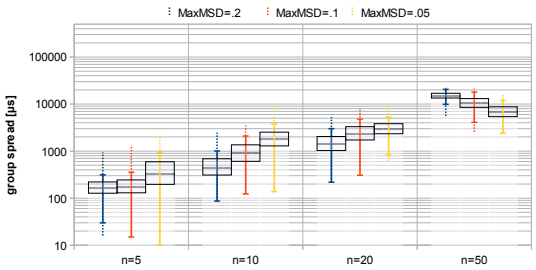
Figure D.21: Different quality aspects of algorithm version V2 in a fault-free ring topology.



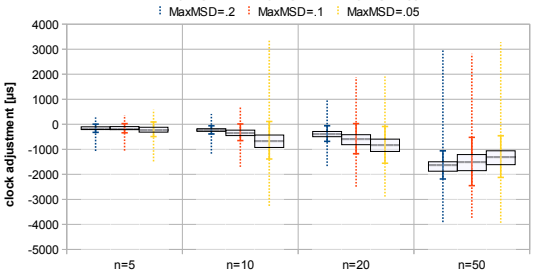
(a) Number of achieved syncs until 5000s in 100 randomized simulations



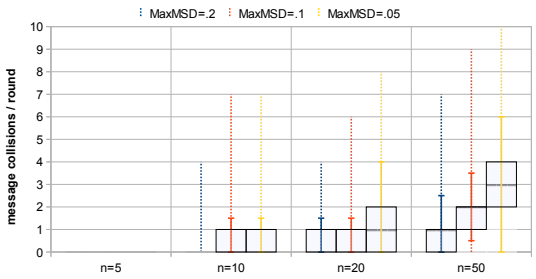
(b) Time to sync diagram (Randomized evaluation)



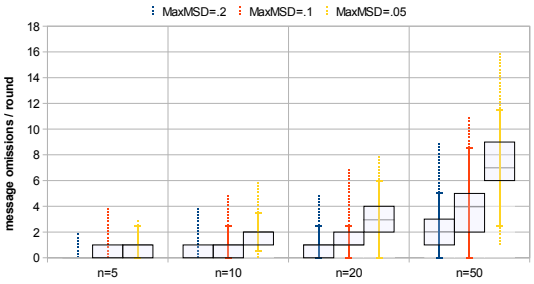
(c) Group spread diagram



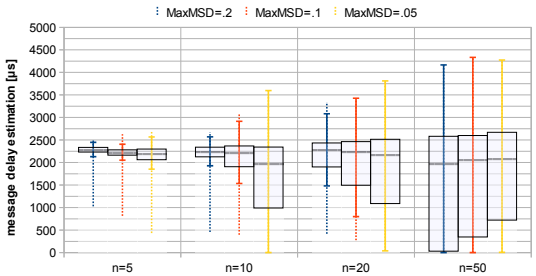
(d) Clock adjustment diagram



(e) Message collision diagram



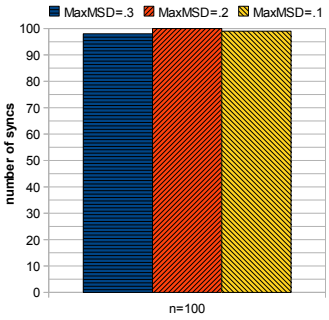
(f) Message omission diagram



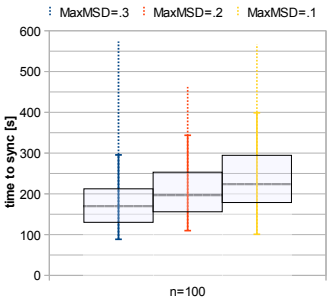
(g) Message delay estimation diagram

Figure D.22: Different quality aspects of algorithm version V8 in a fault-free ring topology.

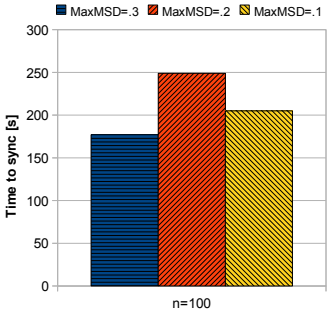
D.7 Fault-free Randomly-structured Multi-hop System



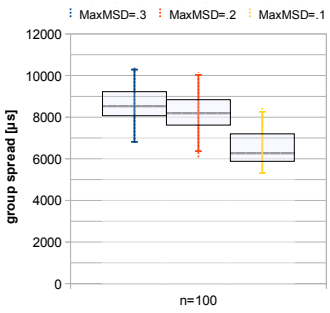
(a) Number of achieved syncs until 1000s in 100 randomized simulations



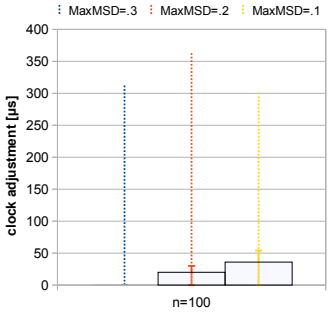
(b) Time to sync diagram (Randomized evaluation)



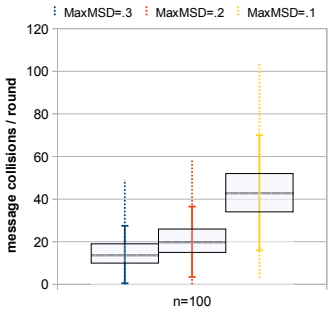
(c) Time to sync diagram (Prepared evaluation)



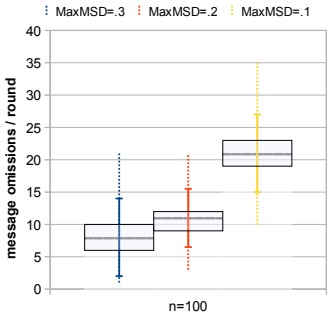
(d) Group spread diagram



(e) Clock adjustment diagram

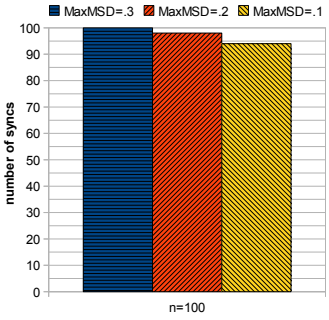


(f) Message collision diagram

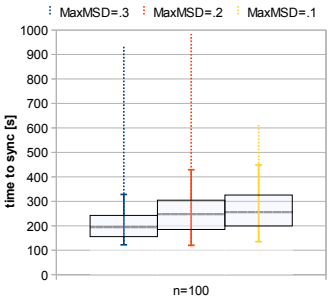


(g) Message omission diagram

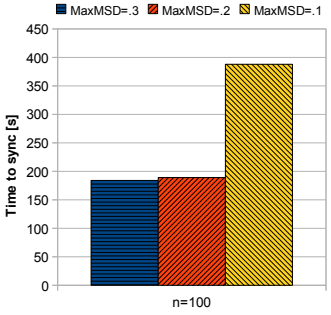
Figure D.23: Different quality aspects of algorithm version V1 in a fault-free random geometric topology.



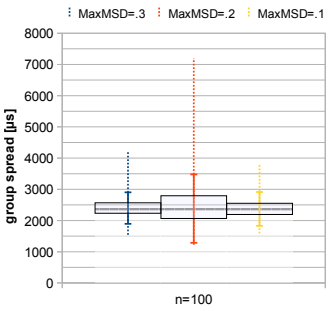
(a) Number of achieved syncs until 1000s in 100 randomized simulations



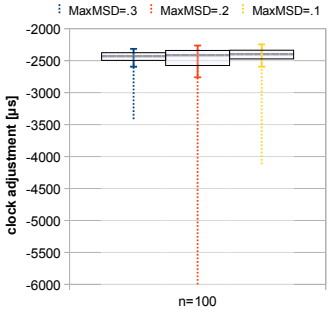
(b) Time to sync diagram (Randomized evaluation)



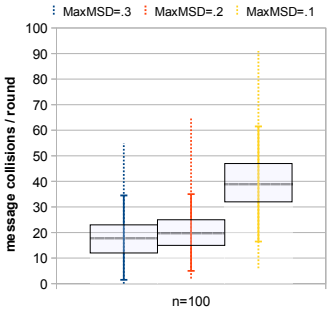
(c) Time to sync diagram (Prepared evaluation)



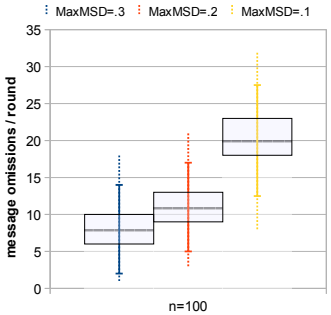
(d) Group spread diagram



(e) Clock adjustment diagram

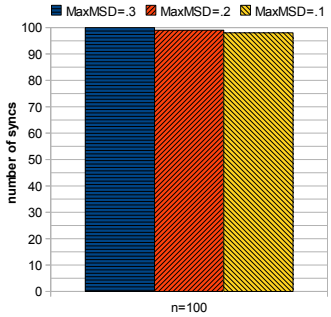


(f) Message collision diagram

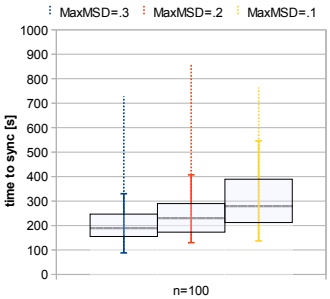


(g) Message omission diagram

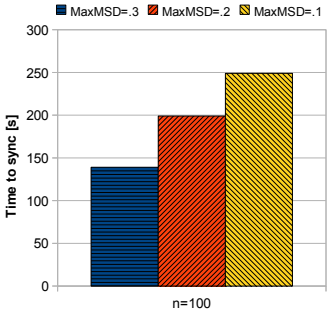
Figure D.24: Different quality aspects of algorithm version V2 in a fault-free random geometric topology.



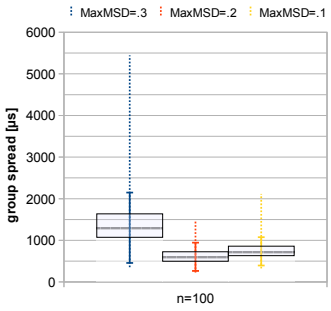
(a) Number of achieved syncs until 1000s in 100 randomized simulations



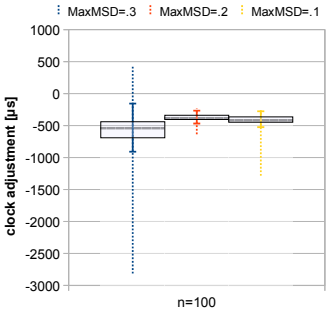
(b) Time to sync diagram (Randomized evaluation)



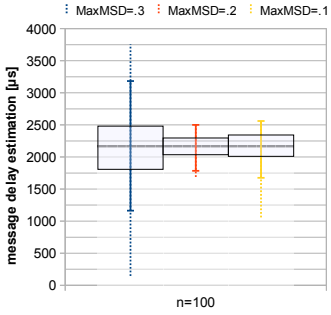
(c) Time to sync diagram (Prepared evaluation)



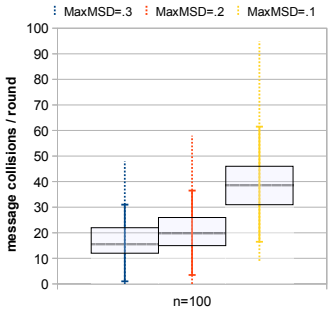
(d) Group spread diagram



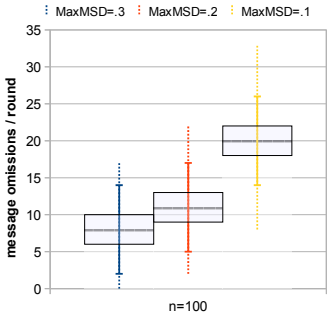
(e) Clock adjustment diagram



(f) Message delay estimation diagram



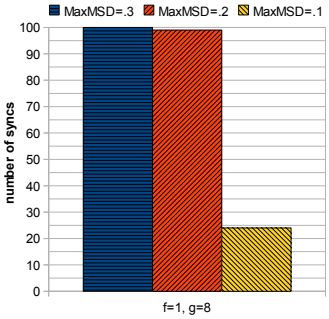
(g) Message collision diagram



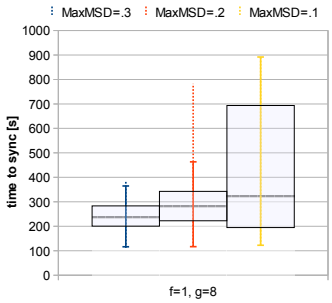
(h) Message omission diagram

Figure D.25: Different quality aspects of algorithm version V8 in a fault-free random geometric topology.

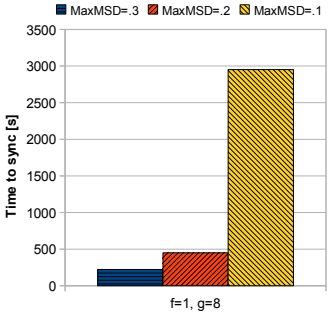
D.8 Coherent Grouped Multi-hop System



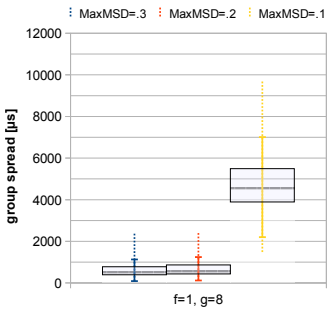
(a) Number of achieved syncs until 1000s in 100 randomized simulations



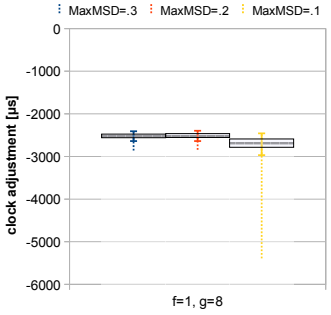
(b) Time to sync diagram (Randomized evaluation)



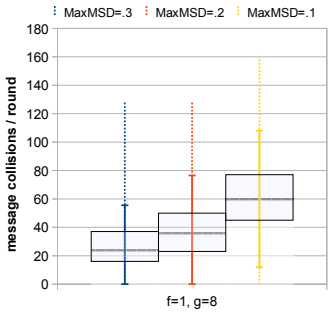
(c) Time to sync diagram (Prepared evaluation)



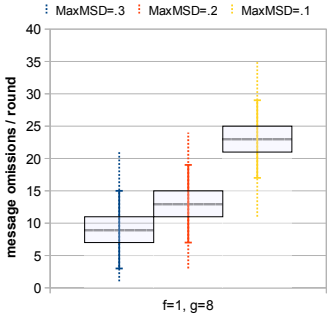
(d) Group spread diagram



(e) Clock adjustment diagram

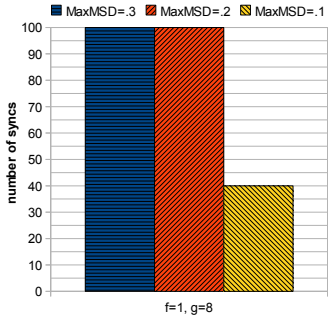


(f) Message collision diagram

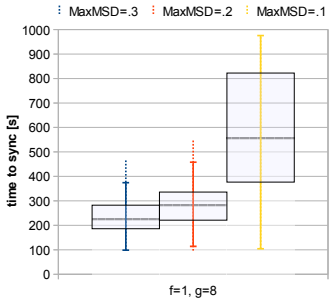


(g) Message omission diagram

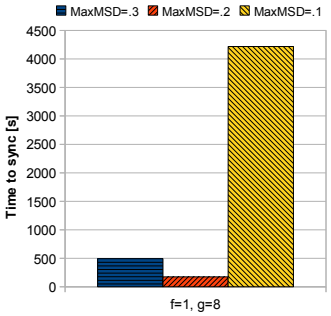
Figure D.26: Different quality aspects of algorithm version V2 in a grouped multi-hop network containing one erroneous node in each group with $g = 8$.



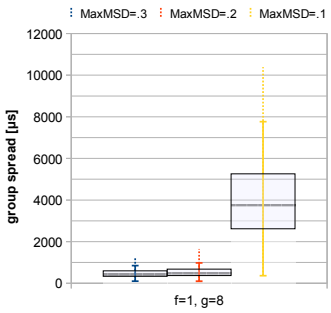
(a) Number of achieved syncs until 1000s in 100 randomized simulations



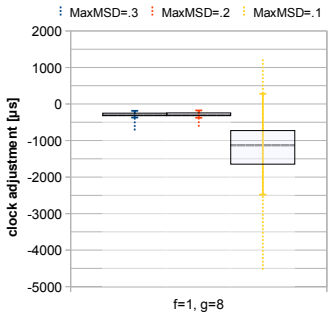
(b) Time to sync diagram (Randomized evaluation)



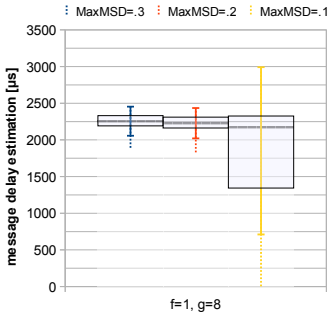
(c) Time to sync diagram (Prepared evaluation)



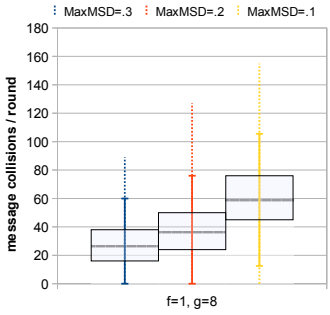
(d) Group spread diagram



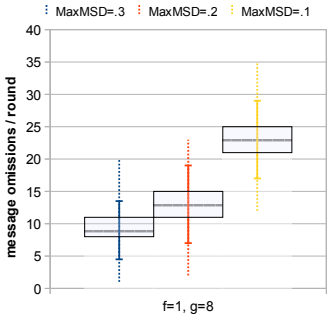
(e) Clock adjustment diagram



(f) Message delay estimation diagram



(g) Message collision diagram



(h) Message omission diagram

Figure D.27: Different quality aspects of algorithm version V8 in a grouped multi-hop network containing one erroneous node in each group with $g = 8$.