DISSERTATION

Investigation of Fault-Tolerant Multi-Cluster Clock Synchronization Strategies by Means of Simulation

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften
unter der Leitung von

O.Univ.-Prof. Dr.phil. Hermann Kopetz
Institut für Technische Informatik 182

eingereicht an der Technischen Universität Wien,
Fakultät für Technische Naturwissenschaften und Informatik

von

Alexander Hanzlik
Matr.-Nr. 8853162
Apostelgasse 39, 1030 Wien, Austria

Wien, im September 2004 ...A. Hanzlik.....

Investigation of Fault-Tolerant Multi-Cluster Clock Synchronization Strategies by Means of Simulation

Abstract

Distributed fault-tolerant real-time systems are increasingly deployed for safety-critical applications in automotive, aeronautic and process control domains. Time-triggered systems are becoming the technology of choice due to their deterministic behavior. A key issue in time-triggered systems is the establishment of a fault-tolerant global timebase among all nodes of a distributed application. Most such systems today consist of a single *cluster*, i.e. a set of nodes that execute a distributed application in a concurrent manner communicating over a dedicated communication medium by exchanging messages. It seems reasonable to build up large real-time systems from several clusters into so called *multi-cluster systems*. Such structures impose additional efforts with regard to inter-cluster communication and inter-cluster clock synchronization to bring the cluster times into agreement.

In this thesis we will investigate clock synchronization strategies by means of simulation using SIDERA, a simulation model for time-triggered systems based on the Time-Triggered Architecture (TTA) and the Time-Triggered Protocol (TTP). Various multi-cluster setups including fully hierarchical configurations and configurations containing feedback loops are investigated. Further, a clock synchronization algorithm is presented which provides tight synchronization even if a majority of nodes runs clocks with quartzes of low quality.

Simulationsbasierte Analyse fehlertoleranter Strategien zur Uhrensynchronisation in Multi-Cluster Systemen

Kurzfassung

Fehlertolerante verteilte Echtzeitsysteme halten vermehrt Einzug in sicherheitskritische Domänen der Automobilindustrie, der Luftfahrt sowie der Prozeßautomatisierung. Zeitgesteuerte Systeme sind aufgrund ihres deterministischen Verhaltens für Anwendungen dieser Art besonders geeignet. Ein wesentliches Charakteristikum zeitgesteuerter Systeme ist der Aufbau einer fehlertoleranten globalen Zeitbasis zwischen allen Knoten einer verteilten Applikation. Derzeit bestehen die meisten modernen zeitgesteuerten Systeme aus einem *Cluster*, also einer Menge von Knoten, die eine verteilte Applikation kooperativ ausführen und über ein Kommunikationsmedium Nachrichten austauschen. Ein naheliegender Ansatz zur Realisierung großer Systeme besteht im Aufbau sogenannter *Multi-Cluster Systeme* aus einzelnen Clustern. Derartige Strukturen erfordern zusätzlichen Kommunikationsaufwand zwischen den das System konstituierenden Clustern, speziell in Hinblick auf den Austausch von Zeitinformation, um die Synchronisation der globalen Zeiten innerhalb der einzelnen Cluster zu ermöglichen.

Inhalt der vorliegenden Dissertation ist die Untersuchung von Synchronisationsstrategien auf der Basis eines Simulationsmodells für zeitgesteuerte Systeme (SIDERA). SIDERA basiert auf der Zeitgesteuerten Architektur und dem Zeitgesteuerten Protokoll. Es werden sowohl hierarchische als auch zyklische Konfigurationen analysiert. Weiters wird ein Synchronisationsalgorithmus entwickelt, der eine enge Synchronisation zwischen einzelnen Knoten ermöglicht, selbst wenn die Mehrzahl der lokalen Uhren an diesen Knoten lediglich über Quarze minderer Qualität verfügt.

Acknowledgements

I thank Prof. Hermann Kopetz, the head of the institute, for giving me the opportunity to write a PhD thesis under his guidance besides my full-time job at SIEMENS Austria. He introduced me to the world of fault-tolerant distributed systems and supported me in improving my skills in the course of many interesting discussions.

I want to thank all colleagues at the Real Time Systems Group for a friendly working atmosphere.

I thank Astrit Ademaj for many hours of interesting discussions in the field of distributed systems as well as for the fun we had working together. I'm looking forward working with you on research projects to come.

Günter Bauer, thank you for patiently answering my questions when I was a newcomer at the institute and in almost all happening there.

Further, I wish to thank Michael Paulitsch for introducing me in the field of clock synchronization in distributed systems and for discussing implementation specific aspects of SIDERA (sometimes for several hours on the phone).

Wilfried Steiner brought startup algorithms to my knowledge. Further, he very thoroughly proof-read this thesis and gave a lot of helpful hints. Thank you very much.

I thank Leo Mayerhofer for technical support and for organizing and configuring hardware during early stages of this thesis.

Special thanks go to our secretary Maria Ochsenreiter for administrative support and for arranging meetings with Prof. Kopetz (which really is a tough job).

I thank my boss and friend Friedrich Wilhelm for his help and advice during a critical phase of my life and in my job, and for supporting my academic activities.

Mom, I thank you for support, your love and for being the best mother in the world.

Doris, thank you for being my partner and best friend and for sharing a great time with me.

Last, I thank Achilles Gaggia [1] for the creation of the modern-day espresso machine.

[1] In 1946, Gaggia invented a high pressure espresso machine by using a spring powered lever system. The first pump driven espresso machine was produced in 1960 by the Faema company.

This thesis is partly supported by the Austrian Science Fund FWF under contract number P16638.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Fault-tolerant distributed real-time systems are more and more used for the control of safety-critical applications in automotive (drive-by-wire), aeronautic (fly-by-wire) and process control (nuclear power plant monitoring systems) domains. Time-triggered systems ([Kop98]) are becoming the technology of choice due to their deterministic behavior. Such systems usually control objects in the environment by issuing control signals to the objects under control, based on calculations performed on input signals from these objects. Due to the distributed characteristics of such systems, there has to be some mechanism to coordinate activities and to provide a consistent view of the state of the system to all nodes of the distributed application.

In time-triggered systems like the TTA ([KB03]), SPIDER ([Min04]), SAFEbus ([HD93]) or Flexray [1], this is done by establishment of a fault-tolerant global timebase among the nodes of a cluster. A cluster consists of a set of spatially separated nodes, each maintaining a local clock used to trigger actions and to timestamp events observed in the system. The nodes execute a distributed application in a concurrent manner and communicate by sending messages over a dedicated communication network. For many applications (e.g. steer-by-wire, ABS) control systems consisting of a single cluster are sufficient. The establishment of a fault-tolerant global timebase within a single cluster is realized by bringing the clocks at the nodes into agreement. This is done by means of *internal clock synchronization* which is a well-investigated problem in distributed systems ([LL84], [HSSD84], [ST87], [RSB90], [CAS94], [AP98]).

A promising approach for the realization of large distributed real-time systems is *clustering*, i.e. to build such systems from single clusters into

[1] Flexray. http://www.flexray.com

multi-cluster systems. This imposes additional efforts with regard to communication and synchronization between the clusters to bring the internally synchronized cluster times into agreement to form a system-wide global timebase.

1.1 Objective of the Thesis

The objective of the thesis is the identification of measures and means that are suitable to provide fault-tolerant, stable and tight clock synchronization in time-triggered multi-cluster real-time systems. For our investigations we will use SIDERA, a simulation model for single-cluster and multi-cluster time-triggered systems based on the Time-Triggered Architecture TTA ([KB03]) and the Time-Triggered Protocol TTP ([TTT99]). SIDERA is a discrete-state simulation model implemented using MATLAB/Simulink [2] that has been especially designed for the investigations and experiments performed in the course of this thesis.

We rely on a simulation based approach due to the following reasons:

- Dedicated hardware is quite expensive. A TTP node is in the range of 1000 Euro, which makes the costs of experimental multi-cluster setups of reasonable size hardly affordable.
- An experimental configuration in hardware means considerable efforts in setup and cabling and impedes the comparison between different multi-cluster configurations.
- Gathering representative data in real-time execution can take much time.
- There is only limited control of the behavior of system components.
- A simulation based approach imposes almost no limitation with regard to system complexity. The only limiting factor is computational power.

1.2 Structure of the Thesis

The next section of this chapter gives an overview of the related work in the field of evaluation of clock synchronization algorithms by means of simulation.

[2] Mathworks. http://www.mathworks.com

Chapter 2 introduces the concepts the thesis is based on. First, some terms and their intended meaning throughout the thesis are introduced, followed by a short overview of the fundamental concepts of dependability. The following section is related to the concepts of time and space, the notion of physical time and to time standards and time sources. We then introduce the concept of physical clocks and local clocks and give an overview of the principles of internal, external and multi-cluster clock synchronization. The chapter closes with an overview of the Time-Triggered Architecture.

Chapter 3 presents SIDERA, a simulation model for fault-tolerant time-triggered systems, which will be used for the clock synchronization experiments presented in this thesis. We start with the description of the software environment and the principle of operation of SIDERA. Then, the internal structure of SIDERA is described in detail: we focus on the protocol services provided by the Time-Triggered Protocol and how they are implemented in the simulation model. Further, the simulation of the physical properties of a distributed system that are covered by SIDERA are described. The following section discusses the parameters that can be passed to the simulation model by means of a configuration file as well as the output that is generated by the model. The last section of this chapter is related to the verification of SIDERA by means of verification tests against a VHDL model of a TTP/C controller.

Chapter 4 describes methods and means that improve the quality of synchronization within a cluster. It introduces the notion of node *calibration* which aims at manipulation of the frequency of the local clocks such that they get into better agreement with an external time reference or with the internally synchronized cluster time. Static and dynamic calibration of the local clocks are described by formal analysis and evaluated in the course of experimental setups.

Chapter 5 is related to multi-cluster clock synchronization. We extend the concepts of dynamic clock calibration within a cluster introduced in Chapter 4 and provide a fault-tolerant clock synchronization algorithm for multi-cluster systems that is ideally suited for systems in which a majority of clocks have quartzes of low quality. This algorithm is based on a combination of clock-state and clock-rate correction. The second part of this chapter is devoted to the experimental evaluation of different multi-cluster architectures with regard to the achievable precision. We present both hierarchical configurations and configurations with feedback loops. We also compare the performance of our new clock synchronization algorithm combining clock-state and clock-rate correction with the performance of a multi-cluster clock synchronization algorithm based on clock-state correction only. Further, we compare

the blackout survivability of calibrated clusters and of non-calibrated clusters.

Chapter 6 concludes the thesis. The contribution of the presented work is summarized and an outlook of the future work is given.

1.3 Related work

[SWGS99] and [WGSS99] survey SimUTC, a framework for simulation of round-based clock synchronization algorithms in fault-tolerant distributed real-time systems, using the discrete-event simulation package C++ SIM ([LM94]). SimUTC has been developed in the course of the SynUTC [3] project which is devoted to establishing a time service for fault-tolerant distributed real-time systems. The toolkit incorporates either real network controllers or their simulated counterparts.

Cluster simulation ([Gal99], [GP99]) provides a cheap and useful technique to test single nodes of a distributed application in isolation without the need to setup the whole system. The idea is to simulate the target system for the node under test by means of one or more physical nodes connected to the test node via a dedicated logical line interface.

[Pal00] presents detailed investigations of communication properties of the Time-Triggered Protocol TTP/C based on a deterministic fault injection approach with regard to various kinds of faults on the communication medium and corresponding error detection latencies using TTP_{SIM}, a simulation environment for TTP/C ([PG98]).

[Bau99] investigates the performance and the limits of the clock synchronization algorithm used in the Time-Triggered Protocol TTP/C. The algorithm is analyzed using a VHDL simulation of the hardware the protocol is executed on. The system response (i.e. achieved synchrony within a cluster and cluster drift rate from real-time) to altered parameters is presented and discussed.

[Sch96] compares the performance of different clock synchronization algorithms and clock correction strategies with regard to achievable synchrony by means of simulation. [Sch95] deals with the simulation of multi-cluster clock synchronization strategies in the course of the ClockSync project ([Sch94a], [Sch94b]), which is concerned with the development of a large simulation model for the synchronization of clocks in a distributed real-time system.

[3] SYNUTC. http://www.auto.tuwien.ac.at/Projects/SynUTC/

[AP98] analyze the performance of several deterministic clock synchronization algorithms in the presence of clock crash, processor crash, timing, Byzantine, network omission and network performance failures. A simulation model consisting of n nodes connected through a point-to-point fully connected network is used for the analysis.

[dAB94] presents a software-based model for fault-tolerant clock synchronization in distributed UNIX environments and analyzes the performance of a software-based implementation according to variations in CPU and network load.

Chapter 2

Basic concepts

2.1 Terminology

This section defines some terms and their intended meaning as used throughout the thesis.

Node, Processor, Process. A node is a computational unit that executes a part of a distributed application. Each node maintains a local clock.

Cluster. A cluster is a set of spatially separated nodes that executes a distributed application in a concurrent manner. The nodes communicate via a dedicated communication network. The network is *fully connected* if there is a physical communication link between any two nodes and *partly connected* otherwise.

Gateway. A gateway connects two clusters. It consists of two nodes, each located in one of the interconnected clusters that communicate via a dedicated external communication link.

System. A system is defined to be a computational entity that interacts with its environment by receiving input signals from the environment and producing output signals to the environment. The system specification defines the intended behavior of a system.

User. A user is a human or another system that interacts with the given system at the service interface.

Service. The system *service* or *function* is the behavior of the system as perceived by its user(s). A system is *correct* if it provides its service according to the system specification.

2.2 Dependability

Systems that are used in safety-critical environments are usually expected to fulfill dependability requirements.

Dependability is defined as the basic trustworthiness of a computer system that allows people to rely on the service it delivers ([Lap92]). A systematic exposition of the concepts of dependability consists of three parts: the *threats* to, the *attributes* of, and the *means* by which dependability is attained ([ALR01]).

2.2.1 Threats

The threats to dependability are *faults*, *errors* and *failures*.

Fault. A fault is the adjudged or hypothesized cause of an error. A fault is *active* when it produces an error, otherwise it is *dormant*.

Error. An error is an unexpected problem internal to the system that, if altering the system service, may result in a system *failure*. An error is *detected* if its presence in the system is indicated by an error message or error signal that originates within the system ([ALR01]). Errors that are present but not detected are *latent* errors.

Failure. A system failure is the transition from correct service to incorrect service. The ways a system can fail are described by its *failure modes* ([ALR01]), which characterize failures according to the *domain* (value/timing failures), the *perception* by the system user(s) (consistent/inconsistent failures) and the *consequences* on the environment (minor/catastrophic failures). The *failure semantics* is the failure behavior of a system likely to be observed by its users ([Cri91]). A *failure model* is a way for precisely specifying how a system or a system component behaves when it fails ([MS92]):

- **Fail-stop failure**
  A component fails by ceasing execution without undergoing any incorrect state transition and its failure is detectable by other components ([SS83]).
- **Crash failure**
  Like fail-stop, without the guarantee of detectability.
- **Omission failure**
  A component fails by not responding to some input ([CASD85]).
- **Timing failure**
  A component gives a correct response, but either too early or too late. A *late* timing failure is also referred to as a *performance failure.*
- **Arbitrary failure**
  The components failure behavior is completely unspecified ([LSP82]). A component assumed to fail in this manner may perform unknown, inconsistent, or even malicious actions.

2.2.2 Attributes

The attributes of dependability are *reliability, availability, safety* and *security.*

Reliability is dependability with respect to the continuity of correct service. It is also a measure of the time to failure.

Availability is dependability with respect to the readiness for usage. It is a measure of correct service delivery with respect to the alternation of correct and incorrect service.

Safety is dependability with respect to the avoidance of catastrophic consequences on the user(s) and the environment of the system. It is a measure of continuous delivery of either proper service or improper service after a benign failure or a measure of the time to a catastrophic failure.

Security is the ability to prevent unauthorized access to information (confidentiality) and improper alterations of system service (integrity). Security is a *protective* term related to the provision of service to authorized users and the denial of service to unauthorized users ([Jon98]).

2.2.3 Means

The means to attain dependability are *fault prevention*, *fault tolerance*, *fault removal* and *fault forecasting* ([ALR01]).

Fault prevention is attained by quality control techniques employed during the design and manufacturing of hardware and software. They include structured programming, information hiding, modularization, etc., for software, and rigorous design rules for hardware. Operational physical faults are prevented by shielding, radiation hardening, etc., while interaction faults are prevented by training, rigorous procedures for maintenance, "foolproof" packages. Malicious faults are prevented by firewalls and similar defenses.

Fault tolerance is intended to preserve the delivery of correct service in the presence of active faults. It is generally implemented by error detection and subsequent recovery. A common method to achieve fault-tolerance is by running replicas of the same operation on several distinct processors.

Fault removal is performed both during the development phase and the operational phase of a system. Fault removal during the development phase of a system life-cycle consists of verification (checking the specification), diagnosis (identification of deviations from the specification) and correction of these deviations. Fault removal during the operational phase of a system is done by means of corrective maintenance (identification and removal of faults on the base of reported errors) or preventive maintenance (uncovering and removal of faults that might cause subsequent errors).

Fault forecasting is conducted by performing an evaluation of the system behavior with respect to fault occurrence or activation. Qualitative (ordinal) evaluation aims to identify, classify and rank the failure modes, or the event combinations (component failures or environmental conditions) that would lead to system failures; quantitative (probabilistic) evaluation aims to evaluate in terms of probabilities the extent to which some of the attributes of dependability are satisfied.

2.2.4 The Dependability Tree

Figure 2.1 shows the dependability tree which relates the concepts of dependability discussed in the previous sections.

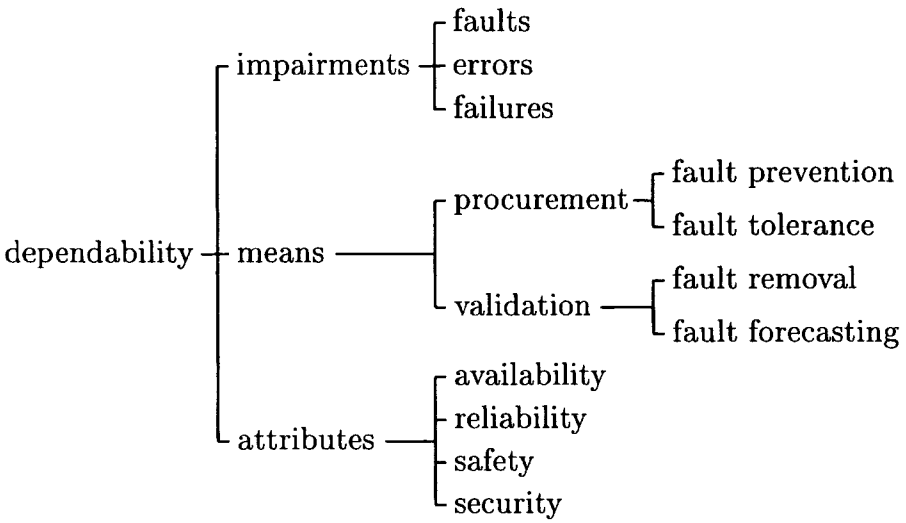

Figure 2.1: The Dependability tree [Lap95].

2.3 Time, Instants, Events

The notion of time is very familiar to humans and pervades our daily lives. Although the omnipresent characteristics of time, we may find it hard to express in words what time really *is*.

Many people, if asked to give a definition of time, may answer "I know well enough what it is, provided that nobody asks me about; but if I am asked what it is and I try to explain, I am baffled" ([Aug]).

Reasoning about the nature of time as well as the relation of space and time has a long tradition in the history of mankind.

2.3.1 Time and Space

For a long time men thought of time as something *periodical* strictly connected to natural events; therefore in many ancient cultures, the Greek and Roman classical world included, time was mainly conceived with a *cyclic* structure ([Sch94c]). Philosophers in the classical times gave time mainly a practical role, being connected to the changes evidenced by astronomical phenomena and by the problem of motion. Space was given a greater attention than time, the latter being considered as a disturbance in the conceptual systems. Aristotle thought of time and motion defining each other ([Ari91]).

He perceived the problem of differentiating time from movement since time cannot cease or change its speed like ordinary motions do. He also perceived that the uniform circular motion can be used to measure time.

The birth of modern Mechanics put time in a privileged, central position as the *independent variable* in the observation and description of motion and the formulation of the physical laws. Newton looked at time as a container of events, which homogenously flows independently of anything else: "Absolute true and mathematical time of itself, and from its own nature, flows equably without relation to anything external" ([New87]), considering time as an absolute mathematical entity which can also be called *duration.* Leibnitz, on the other hand, thought that space and time only represent relative order relations and that they do not possess any objective substantiality.

The discussion whether time were an absolute or relative entity was closed by the fundamental work of Einstein ([Ein55]), as one of the consequences of which is that time looses its privileged position as the independent variable for describing natural phenomena by introduction of a four coordinate space-time continuum used to express the physical laws.

Another consequence which is especially meaningful for distributed systems is the *relativity of the simultaneity* which means that we are able to decide whether events happen *simultaneously* (i.e. at the same instant) or not only if they happen under our direct perception which requires spatial proximity between the location where the events occur and the observer of the events. To learn from the occurrence of events at remote locations which cannot be directly observed, some information exchange is required which can be done only with a finite speed, the speed of light.

For illustration, consider the following example. Assume two observers at spatially separated physical locations A and B, referred to as obs_A and obs_B, each having a local clock. Upon perception of a *local event*, each observer sends a message to its counterpart at the remote location. Upon reception of a message, each observer learns from the occurrence of the *remote event.* Be $\Delta_{A,B} > 0$ the *message transmission time* between location A and B, assumed to be constant and known by both observers. We now focus on obs_A, the observer in A. Consider a local event e_A happening at A when obs_A's clock reads t. Assume that obs_A receives a message from B when its clock reads t'. obs_A concludes that

- e_A was *before* e_B if $t < t' - \Delta_{A,B}$
- e_A was *after* e_B if $t > t' - \Delta_{A,B}$
- e_A and e_B were *simultaneous* if $t = t' - \Delta_{A,B}$

Upon perception of each local event, both observers obs_A and obs_B have to wait for $\Delta_{A,B}$ time units to be able to decide whether some remote event happened simultaneously to the local event observed.

2.3.2 Physical time

A common view of time is that of a one-dimensional, directed *timeline* ([Wie14], [Rus36]) which is made up of an infinite set of *instants.*

An *instant* is a cut of the timeline and is defined (Russell 1914) as a set of *events*, any two of which are simultaneous and such that there is no other event which is simultaneous with them at all.

An *event* is said to be *at* a particular instant when it is a member of the set defining that instant. Any two events being at the same instant are said to be *concurrent.*

Temporal order of instants is defined by stipulating that one is earlier than another if there is some event at the former that is earlier than some event at the latter. If neither instant is earlier than the other, then they are simultaneous (identical) ([Pet02]). Instants are *totally ordered*, whereas events are only *partially ordered* (there can be many events happening at the same instant).

The continuum of real time can be modelled by a directed timeline consisting of an infinite set T of instants with the following properties ([Whi90], p. 208):

1. *T is a simply ordered set, that is, if p and q are any two instants, than either p is simultaneous with q, or p precedes q, or q precedes p, and these relations are mutually exclusive. Furthermore, if p precedes q and q precedes another instant r, then p precedes r, and q is said to be between p and r.*

2. *T is a dense set. This means, that, if p precedes r, there is at least one q which is between p and r.*

3. *T satisfies Dedekinds postulate, namely if T1 and T2 are any two non-empty parts of T such, that every instant of T belongs either to T1 or T2 and every instant of T1 precedes every instant of T2, then there is at least one instant t such that any instant earlier than t belongs to T1 and any instant later that t belongs to T2.*

2.3.3 Time measurement

is the determination of the size of an interval on the timeline delimited by two instants, the *start event* and the *terminating event* of the interval. This is done by dividing the timeline into *granules* (small intervals of equal size) and by *counting* the number of granules that "fit" into the time interval to measure. For any time interval to measure is made up of a finite number of granules, this method introduces a close relation between time and counting.

The size of a granule itself is derived from the frequency of some periodic action like a swinging pendulum or a caesium atomic clock. The use of the caesium atomic clock leads directly to the definition of the *physical second* by the International System of Units (SI).

2.3.4 Time standards and time sources

Time standards

The **International Atomic Time TAI** (Temps Atomique International) was specified in 1967 by the Bureau International de l'Heure (BIH) and is the base of the time and frequency standards of the world since 1972 ([Mil94]). It is a time standard that can be produced in a laboratory, but it is in agreement with the second derived from astronomical observations. TAI defines the *physical second* as the duration of 9 192 631 770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium 133 atom. It is a strictly monotonic timebase without discontinuities.

The **Universal Time Coordinated (UTC)** is based on astronomic observations of the rotation of the earth relative to the sun and is the base of "wall-clock time". There is a known offset between UTC and local wall-clock time due to timezones and daylight saving times. Because of a slight irregularity of the rotation of the earth around the sun, the duration of the UTC second changes slightly over time, deviating from the physical second provided by TAI. This deviation is corrected by occasionally inserting a *leap second* into UTC to maintain synchrony between the UTC and astronomical phenomena, like day and night. Due to the occasional insertion of a leap second UTC is not *chronoscopic*, i.e. it is a timebase with discontinuities.

Time sources

provide access to time standards.

The **Global Positioning System (GPS)** operated, funded and controlled by the U. S. Department of Defense is a satellite-based system that provides accurate location and timing data worldwide. Provision is done via specially coded satellite signals that can be processed in a GPS receiver, enabling the receiver to compute position, velocity, and time. GPS provides access to UTC with an accuracy better than 15 nanoseconds. It is an accepted and widely used time source for military and civil applications ([Dan97]).

The **GLObal NAvigation Satellite System (GLONASS)** is (similar to GPS) a satellite-based navigation system that enables global-wide positioning, velocity measuring, and timing information. The GLONASS system is managed by the Russian Space Forces for the Russian Federation Government. The time of GLONASS satellites is synchronized to UTC (with a constant offset to provide chronoscopic behavior) with approximately 15 nanoseconds ([Leb98]).

Galileo is the planned European satellite navigation and time transferring system and is a project launched by the European Union. Unlike GPS and GLONASS being not maintained by national defence authorities, it shall provide guarantees of service, better availability in urban areas and equal quality of service both for civil and military needs. Galileo is planned to become fully operational in 2008.

Terrestrial radio stations provide time information that can be used by time-code receivers in several regions of the world. Receivers, however, are subject to occasional gross errors due to propagation and equipment failures ([Mil94]). They allow for accuracies of timing information with respect to UTC in the range of 50 milliseconds up to 10 seconds ([Lic97]).

2.4 Clocks

2.4.1 Physical clocks

Physical clock. A *physical clock* is a device for time measurement that contains a *counter* and a *physical oscillation mechanism* that periodically generates an event to increase the counter ([Kop97], p. 48). The periodic event is called the *microtick* of the clock. The duration between two consecutive microticks is the *granularity* of the clock. Physical clocks are also referred to as *hardware clocks.*

Reference clock. Assume an *omniscient external observer* who can observe all events that are of interest in a given context (relativistic effects are

disregarded). This observer possesses a *unique reference clock z* with a very high granularity compared to any other clock within the observed system. Reference clock z is assumed to be in perfect agreement with International Atomic Time TAI. Whenever the omniscient observer perceives the occurrence of an event e, it will timestamp this event with the current state of the reference clock. $z(e)$ is the state of the reference clock at occurrence of event e. The *duration* between two events e_1 and e_2 is measured by counting the microticks of the reference clock that occur in the interval between $z(e_1)$ and $z(e_2)$. The *granularity* g^k of a clock k is given by the nominal number n^k of microticks of the reference clock z between two microticks of this clock k.

In the following, the state of reference clock z will be referred to as **realtime**.

Clock drift. The *drift* of a physical clock k between microtick i and microtick $i+1$ is the frequency ratio between this clock k and the reference clock at the instant of microtick i. The drift is determined by measuring the duration of a granule of clock k with the reference clock z and dividing it by the nominal number n^k of reference clock microticks in a granule:

$$drift_i^k = \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k} \tag{2.1}$$

The drift of a physical clock consists of a *systematic* and a *stochastic* part.

Systematic drift. The systematic drift of crystal oscillators is a constant deviation of the frequency from the specified nominal value. It is influenced by the operating environment (especially by variations in temperature) as well as by aging of the crystal oscillator.

Stochastic drift. In addition to the systematic drift, crystal oscillators deviate randomly within a certain range. The reason for this behavior cannot be discovered. The systematic part of hardware clocks is approximately constant and approximately 100 times larger than the stochastic part ([Sch88]). The deviations causing the stochastic part of the clock drift are assumed to follow a symmetric distribution function in a given period of time ([Pau02]).

Drift rate. Because a good clock has a drift very close to 1, for notational convenience the notion of a *drift rate* ρ_i^k is introduced as

$$\rho_i^k = abs(\frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k} - 1) \tag{2.2}$$

The drift rate of a clock k determines the deviation of clock k from reference clock z in seconds per second. Positive values for ρ_i^k indicate that clock i is running slower and negative values for ρ_i^k indicate that clock i is running faster than reference clock z. A perfect clock will have a drift rate of 0. Real clocks have a varying drift rate that is influenced e.g. by environmental conditions or aging of the crystal. Within specified environmental parameters, the drift rate of a resonator is bounded by the *maximum drift rate* ρ_{max}^k (typically in the range of 10^{-2} to 10^{-7} sec/sec, depending on the quality and price of the resonator) which is guaranteed by the manufacturer and documented in the data sheet of the resonator. A typical value for the drift rate of clocks used in modern computers is in the order of 10^{-6} sec/sec ([CF94], [BHHN00], [CAS94]).

2.4.2 Local clocks

Local clock. A *local clock* is a device for time measurement that contains a *counter*, a physical clock and a mechanism that periodically generates an event to increase the counter. The counter of the local clock can be modified such that the speed of the local clock can be increased or decreased by application of an *adjustment value* or *clock state correction term* (see Section 2.5). The state of local clock C_i at realtime t is defined as the state of its hardware clock H_i and the value of the clock state correction term $CORR_i$ at realtime t:

$$LT_{C_i}(t) = H_i(t) + CORR_i(t) \tag{2.3}$$

$LT_{C_i}(t)$ is referred to as the **local time** of local clock C_i at realtime t ([LL84]).

Clocks whose state can dynamically be altered by application of adjustment values are also termed **virtual clocks** ([ST87], [CF85], [Sch86]) or **logical clocks** ([BHHN00], [CAS94], [RSB90]).

2.5 Clock synchronization

Synchronization is the action of making different processes in a computer network or different parts of a circuit or different clocks to agree on a same time reading. In the context of multiprocessor and distributed systems, synchronization ensures that operations occur in the logically correct order, and

it allows the establishment of causal implications between events in different computational units ([Sch94c]).

Clock synchronization algorithms aim at bringing a set of distributed, spatially separated clocks into agreement. *Internal clock synchronization* is the action of bringing all clocks within a set into closer agreement to each other while *external clock synchronization* aims at bringing a clock or a set of clocks into agreement with a reference clock. Clock synchronization algorithms usually do not synchronize hardware clocks. Instead, they try to attain synchrony among a set of *local clocks* by determination and application of proper *adjustment values* (see Section 2.4.2).

2.5.1 Internal clock synchronization

Internal clock synchronization aims at establishing synchrony among a set of clocks such that the maximum deviation between any two clocks can be bounded by a known and constant value. In the following, the term *cluster time* denotes the internally synchronized *local times* at the nodes within a cluster. The state of each node's local clock represents this node's view of cluster time. The *cluster drift rate* is the drift rate of cluster time against realtime.

Objectives

Typical **objectives** of internal clock synchronization algorithms are

- **Bounded internal deviation**
  At any time the deviation between any two correct clocks C_i and C_j can be bounded by a constant Π ([LL84],[HSSD84], [FC95]):

  $$|C_i(t) - C_j(t)| \leq \Pi \tag{2.4}$$

  Π is called the *precision* ([Kop97], p.50) and is a measure for the tightness of synchronization within an ensemble of clocks.

- **Bounded cluster drift rate**
  Any synchronized clock C_i should read times that are within a linear envelope of real-time ([CAS94]). That is, there should exist a constant β such that for any clock C_i and any realtime t:

  $$X + \frac{t}{(1+\beta)} < C_i(t) < X' + (1+\beta)t, \tag{2.5}$$

where X and X' are constants which depend on the initial conditions of the clock synchronization algorithm execution.

A set of clocks that meets the *bounded internal deviation* objective is said to be *internally synchronized* with precision Π.

Requirements

Typical **requirements** to meet the objectives of internal clock synchronization are

- **Bounded transmission delay**
  There is a known upper bound t_{del} on the time t required for the transmission of a message. Clock synchronization requires the exchange of clock values among the nodes of a cluster my means of messages. This requirement ensures that any clock C_i can retrieve the state of any other clock C_j within a known and upper bound in time.

- **Bounded hardware clock drift rate**
  The drift rate of all hardware clocks H_i with respect to realtime can be bounded by a constant ρ:

$$\forall s, t, s \leq t : (1 - \rho)(t - s) \leq H_i(t) - H_i(s) \leq (1 + \rho)(t - s) \quad (2.6)$$

Assumptions

Typical **assumptions** that clock synchronization algorithms go on are

- **Initial synchronization**
  Some algorithms require a bound on the initial deviation of any two clocks ([LMS85], [LL84]).

- **Bounded number of faulty clocks**
  Algorithms that make no assumptions about the *failure semantics* of clocks (see Section 2.2.1) usually guarantee the *bounded internal deviation* objective under the assumption that at most one third of the clocks are faulty at the same time (e.g. [LL84], [BHHN00], [ST87]). This is a consequence of the fact that (in the absence of authentication), synchronization can be achieved only if fewer than a third of the clocks in the system are *arbitrary* faulty ([LSP82], [PSL80]). If

the failure semantics can be restricted such that only crash failures and clock reading failures can occur, a majority of non-faulty nodes is sufficient to ensure the bounded internal deviation objective ([CF94]). [MS85] presents a clock synchronization algorithm that guarantees the bounded internal deviation objective if fewer than 1/3 of the processors are faulty; otherwise, the processors either detect that too many faults have occurred or precision diverges bounded, referred to as *graceful degradation* - reasonable and predictable behavior - as long as no more than 2/3 of the processors are faulty. *Self-stabilizing* clock synchronization algorithms try to reestablish a stable system state and the bounded internal deviation objective if a majority of processors becomes faulty. The idea of *self-stabilization* has first been addressed by Dijkstra ([Dij74]). It is related to the problem of bringing a system from any so called *illegitimate* state to a *legitimate* state within finite time. Regarding clock synchronization, a system with a majority of faulty processors is in an illegitimate state (the bounded internal deviation objective is not fulfilled). [DW95] presents two self-stabilizing protocols for synchronized bounded clocks in the presence of arbitrary (Byzantine) processor faults. However, the expected stabilization time of both protocols is exponential in the number of faulty processors which limits their practical use.

Principle of operation

Internal clock synchronization is the periodic activity of determination and application of a *clock state correction term* for each local clock to achieve agreement among an ensemble of clocks. The action of applying a clock state correction term to a local clock is referred to as *clock adjustment.* The point in time when clock adjustment starts is referred to as *synchronization instant.* The realtime interval between two synchronization instants is called *synchronization interval.*

Internal clock synchronization is a process that comprises three steps. Every node periodically

1. **Reads the values of the other clocks.**
   Due to the presence of possibly varying communication delays and due to the existence of clock drifts, getting an exact knowledge of a remote clock value is not feasible. Thus, only estimates of the remote clock values can be acquired ([AP98]). Therefore, the process of reading a remote clock is also known as *remote clock estimation.* The *clock*

reading error is the deviation between the *real* state and the *estimated* state of the remote clock at the time the estimate becomes available.

2. **Determines a *clock state correction term* for its clock**
by invocation of a *convergence function* based on a set of remote clock estimates. In the following, $f(p, x_1, ..., x_n)$ denotes a convergence function f invoked at processor p based on the remote clock readings $x_1, ..., x_n$. f denotes the maximum number of tolerated faulty clocks. Popular convergence functions are:

   - *Interactive convergence function* f_e
   Also known as *egocentric average* ([LMS85]), f_e returns the average of all arguments x_1 through x_n where x_j $(1 \leq x_j \leq n)$ is kept intact if it is no more than ω apart from x_p (processor p's own clock reading) and replaced by x_p otherwise. A lower bound for ω is the achievable precision of the clocks to avoid *partitioning* of the set of clocks into disjoint subsets running faster or slower than other subsets. No sorting mechanism is required to eliminate faulty clock readings resulting in low complexity of clock correction.

   - *Fast convergence function* f_{fc}
   The Fast convergence function f_{fc} ([MS85]) returns the average of all arguments x_1 to x_n that are within ω of at least $n - f$ other clock readings. f_{fc} yields high quality precision with the price of high complexity because of the need to determine for each clock reading x_i the deviation to all other $n - 1$ clock readings.

   - *Fault-tolerant midpoint function* f_{ftm}
   The Fault-tolerant midpoint function f_{ftm} ([LL84]) returns the midpoint of the range of values spanned by arguments x_1 to x_n after discarding the f highest and the f lowest values. f_{ftm} is based on the assumption that faulty clocks are running either too fast or too slow, and that good clocks lie in-between. f_{ftm} has higher complexity than f_e due to the necessity of sorting the remote clock readings.

   - *Differential Fault-tolerant midpoint function* f_{dftm}
   f_{dftm} is an extension of the Fault-tolerant midpoint function ([FC97]). f_{dftm} has been proven to be optimal with regard to precision achievable for logical clocks and is defined as follows:

$$f_{dftm}(p, x_1, ..., x_n) = \frac{min(T - \Theta, x_l) + max(T + \Theta, x_u)}{2} \quad (2.7)$$

where $x_l = x_{h_{f+1}}, x_u = x_{h_{n-f}}$
with $x_{h_1} \leq x_{h_2} \leq x_{h_n}, h_p \neq h_q : 1 \leq h_p, h_q \leq n$

where T is p's logical time at invocation time of f_{dftm} and Θ is the maximum error induced by the *remote clock estimation* method.

3. **Applies the clock state correction term to its clock.**
The *clock adjustment policy* defines how the clock state correction term is applied to the local clocks.

   - *Clock state correction (discrete adjustment)*
   The clock state correction term is applied *at once* at each synchronization instant. The local clock is set back of forth according to the sign of the clock state correction term and is running free till the next synchronization instant. Another possibility is to spread the application of the clock state correction term over a longer period of time within the synchronization interval or over the whole synchronization interval. A typical implementation of that mechanism is to divide the synchronization interval into smaller intervals of equal length and to correct one local clock tick per interval until the clock state correction term is exhausted ([TTT99]). This strategy is also referred to as *clock amortization.* It is shown in [SC90] that adding continuous clock amortization to an existing clock synchronization algorithm need not affect the precision of that algorithm if the amortization phase (i.e. the duration of the "piece-wise" application of the clock state correction term) is no longer than the synchronization interval.
   - *Clock rate correction (continuous adjustment)*
   Clock *rate correction* aims at manipulation of the frequency of clocks such that synchronism among an ensemble of clocks is attained by continuous monitoring and manipulation of the frequencies of the local oscillators (e.g. by manipulation of the input voltage of the resonator), an approach usually used in hardware based, continuous-update algorithms ([RSB90]).
   - *Combined approaches*
   [Sch96] discusses various combinations of discrete and continuous clock adjustment. An approach for fault-tolerant clock state correction and clock rate correction can be found in [SW99]. In Chapter 4 we will present a fault-tolerant clock synchronization algorithm that combines distributed clock state correction with central clock rate correction.

2.5.2 External clock synchronization

External clock synchronization aims at establishing synchrony between a set of clocks and an external time reference which can be a time standard (see Section 2.3) or any other external time source that is trusted in. The external time reference is provided by one or more reference clocks referred to as *reference time servers* ([CF85]). It is the goal of external clock synchronization to synchronize a set of local clocks to an external time reference such that the maximum deviation between any local clock and any reference time server can be bounded by a known and constant value.

Objectives

Typical **objectives** of external clock synchronization algorithms are

- **Bounded external deviation**
  At any time the deviation between any correct clock C_i and a reference time server R can be bounded by a constant A:

$$|C_i(t) - R(t)| \leq A \tag{2.8}$$

  A is called the *accuracy* ([Kop97], p.50) and is a measure for the tightness of synchronization between a local clock and a reference time server.

A set of clocks that fulfills the *bounded external deviation* objective is said to be *externally synchronized* to reference time server R with accuracy A. A set of clocks externally synchronized with accuracy A is also *internally synchronized* with *precision* 2A:

$$\forall i,j : |C_i(t) - R(t)| \leq A \wedge |C_j(t) - R(t)| \leq A \Rightarrow |C_i(t) - C_j(t)| \leq 2 \times A \tag{2.9}$$

However, the converse is not true: an initially externally synchronized set of clocks may eventually drift apart from reference time server R if never resynchronized to R.

Principle of operation

External clock synchronization is the periodic activity of determination and application of an *external clock state correction term* for each local clock to achieve agreement between the local clocks and an external time reference.

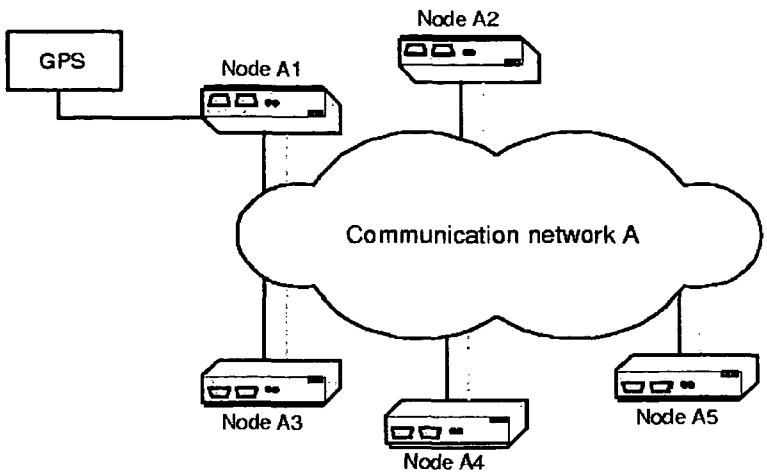

Figure 2.2: External clock synchronization - principle of operation

Time master node. A *time master node* ([BP00b]) is a node which has access to a reference time server. The time master node periodically accesses the reference time server and distributes the clock reading obtained from it to the other nodes within its cluster.

Figure 2.2 shows the principle of operation. In this configuration a GPS receiver (see Section 2.3.4) is used as a reference time server. Node A1 is a time master node.

Like internal clock synchronization, external clock synchronization is a process that comprises three steps. Every clock periodically

1. **Reads the values of the reference time server(s)**
   which are provided by the time master node.

2. **Determines an *external clock state correction term* for its clock.**
   In case of more than one clock reading from reference time servers, the external clock state correction term is determined by invocation of a *convergence function* (see Section 2.5.1) based on a set of remote clock estimates obtained from the reference time servers.

3. **Applies the external clock state correction term to its clock**
   according to its *clock adjustment policy* (see Section 2.5.1).

Requirements

Typical **requirements** to meet the objectives of external clock synchronization are

- **Bounded drift rate of the reference time server R**
  A non-faulty reference time server R synchronized to a time standard always fulfills this requirement. However, R may be faulty or synchronized to an external time reference that violates the bounded drift rate requirement.
- **Bounded hardware clock drift rate of the local clocks**
  See Section 2.5.1.
- **Bounded transmission delay between R and a time master node**
  There is a known upper bound t_{del} on the time t required for the transmission of a message between a reference time server R and a time master node.
- **Bounded transmission delay between a time master node and the other nodes**
  There is a known upper bound t_{del} on the time t required for the transmission of a message between a time master node and any other node. External clock synchronization requires the transmission of clock values between the reference time server and the local clocks. This and the last requirement ensure that any clock C_i can retrieve the state of a reference time server R within a known and upper bound in time and that the clock reading error (the error in estimating the state of the reference time server) can be bounded (see Section 2.5.1).

Assumptions

Typical **assumptions** that external clock synchronization algorithms go on are

- **Bounded number of faulty reference time servers**
  For fault-tolerant external clock synchronization it is necessary to provide more than one reference time server. [CF85] derives lower and upper bounds for the required number of reference time servers dependant on their failure semantics. It is shown that F+1 reference time

servers are sufficient if only *crash* failures can occur or if the reference time servers are *detectable faulty* (i.e. they provide a detectable wrong reading when being accessed). It is also shown that 2F+1 reference time servers are necessary and sufficient to tolerate up to F *arbitrary* failures. Generally, there is no correct external clock synchronization algorithm capable of masking F faulty reference time servers if the total number of reference time servers is not greater than 2F because the set of externally synchronized local clocks may be partitioned if there is no majority of non-faulty reference time servers ([FC97]).

2.5.3 Multi-cluster clock synchronization

Multi-cluster clock synchronization aims at establishing and maintaining synchrony among a set of clusters and is a combination of *internal* and *external* clock synchronization. All clusters within a multi-cluster system establish internally synchronized *cluster times* which are in turn synchronized to each other by means of external clock synchronization.

Objectives

Typical **objectives** of multi-cluster clock synchronization algorithms are

- **Bounded deviation**
  At any time the deviation between any two correct clocks C_i and C_j within the multi-cluster system can be bounded by a constant Π.
- **Bounded system drift rate**
  Any synchronized clock C_i within the multi-cluster system should read times that are within a linear envelope of real-time.

Principle of operation

Gateway. Clusters are connected via *gateways*. A gateway consists of two nodes, each of which belonging to one of the two clusters and communicating with each other via a dedicated communication link.

Figure 2.3 shows a configuration consisting of two clusters A and B with a gateway connecting them. Cluster A and cluster B are internally synchronized. Additionally, cluster B is externally synchronized to cluster A's cluster

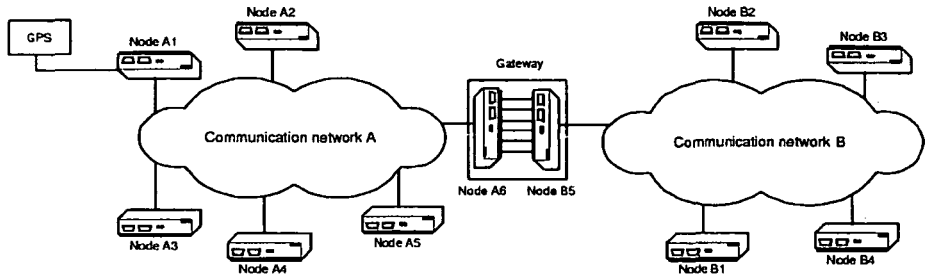

Figure 2.3: Multi-cluster clock synchronization - principle of operation

time. Node A6 serves as a *reference time server*, providing its view of the internally synchronized cluster time in cluster A to node B5, which acts as a *time master node* for cluster B (see Section 2.5.2).

2.6 The Time-Triggered Architecture

2.6.1 Overview

The Time-Triggered Architecture (TTA) provides a computing infrastructure for the design and implementation of dependable, distributed systems ([KB03]). The TTA is designed to meet the requirements of *hard real-time systems*. Hard real-time systems are highly dependable systems upon which the environment imposes stringent timing requirements which must be fulfilled under all operational conditions. The miss of a single *deadline* by such a system (i.e. the system fails to provide a result correct both in the domains of value and time) may cause catastrophic consequences to the environment or may endanger human lives. This property separates hard real-time systems from *soft real-time systems*, which are deployed in environments where occasional violations of deadlines have less or no critical consequences.

Figure 2.4 depicts a *cluster* of the TTA. It consists of a set of *Fault-Tolerant Units* (FTUs) that communicate over a communication network using the Time-Triggered Protocol TTP ([TTT99]). A FTU can consist of one, two or more *nodes* that operate in replica determinism. A set of replicated nodes is replica determinate if all the members of this set have the same externally visible state, and produce the same output messages at points in time that are at most an interval of d realtime units apart ([Pol94]). A FTU provides the specified service without delay even after the failure of a node. The *host* part of a node executes the host application, based on the *TTP*

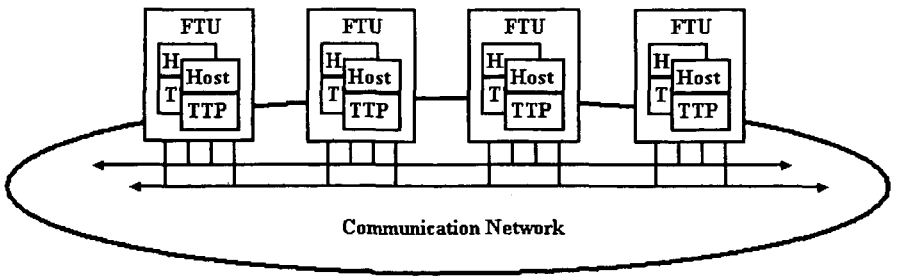

Figure 2.4: TTA cluster

part which handles protocol services (startup, clock synchronization, etc.) provided to the host application. The access to the communication network is controlled by a cyclic time-division multiple access (TDMA) scheme derived from a global notion of time.

2.6.2 The Time-Triggered Model of Computation

The TTA is based on the *Time-Triggered Model of Computation* (TT-Model), a design methodology for the analysis andrepresentation of large hard real-time systems ([Kop98]). The TT-Model consists of four building blocks from which large systems can be built by repetitive use. These are

1. **Interfaces** which provide access to functions of *subsystems* and which are boundaries between subsystems,

2. a **Communication system** which connects interfaces,

3. **Host computers** that read data from one or more interfaces, process data and write data into one or more interfaces and

4. **Transducers** which connect real-time entities in the environment to interfaces and vice versa.

It is assumed that each one of these building blocks has access to a globally synchronized time base of sufficient precision. The establishment of a fault-tolerant global timebase is a key issue in the TTA and is the topic of the following section.

2.6.3 The Sparse Time Model

Dense Time

In a *dense* time model, time progresses along a dense timeline and events can occur at any *instant* on this timeline (see Section 2.3.2).

Assume that all nodes timestamp any event they observe with their local times (i.e. the state of their local clocks at the time the event occurs). In distributed systems, clocks cannot be perfectly synchronized due to clock drifts (see Section 2.4.1) and variations in message transmission delay ([DHS84], [AP98]). As a consequence, the following scenario is always possible for any two events e_1 and e_2 and any two nodes $node_1$ and $node_2$:

- $node_1$ observes e_1 and e_2 at the same tick of its local clock
- $node_2$ observes e_1, its local time proceeds, $node_2$ observes e_2

Consequently, $node_1$ considers events e_1 and e_2 to have happened *at the same time* whereas $node_2$ considers event e_2 to have happened *after* event e_1: the view of the precedence of events is not consistent at the two nodes.

This example shows that a consistent ordering of events is impossible if events are allowed to happen *anytime* and are timestamped with the granularity of the local clocks.

Global time

In the TTA, the nodes of a cluster establish a *global timebase* by means of internal clock synchronization (see Section 2.5.1).

The TTA utilizes the concept of *microticks* and *macroticks* ([Kop97]). Microticks correspond to the local oscillator ticks at each node (see Section 2.4.1), while macroticks represent the global notion of time. Each node generates a macrotick by selecting a number of microticks and synchronizes its macrotick by dynamically increasing or decreasing the number of microticks per macrotick ([Pal00]), according to the clock state correction term that is delivered periodically by the clock synchronization algorithm (see Section 2.5.1). All nodes adjust their local clocks at the same point in global time. The internally synchronized global time proceeds in units of macroticks. The macrotick counter at each node represents this node's view (or *approximation*) of global time.

A global time base is called *reasonable*, if all local implementations of the global time satisfy the condition

$$g > \Pi, \tag{2.10}$$

the reasonableness condition for the global granularity g ([Kop97], p.52) which ensures that the synchronization error is bounded to less than the duration between two macroticks.

Sparse Time

In the *sparse* time model, dense time is partitioned into alternating intervals of *activity* and *silence* as shown in Figure 2.5.

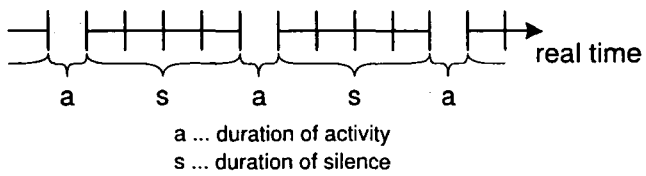

Figure 2.5: Sparse time ([KO02])

All events that occur within an interval of activity are considered to happen at the same time ([Kop92]).

In a distributed system based on a sparse time model, a consistent, total ordering of events is always possible if

- the duration of an activity interval is at most the precision of the clock synchronization and
- the duration of a silence interval is at least four times the duration of an activity interval ([KO02]).

The availability of a sparse global timebase can be used to deterministically solve the mutual exclusion problem when accessing shared resources in a distributed system ([Ade03]). The access to the communication network is organized by means of a *communication schedule* which is determined before runtime. A node is allowed to send a message within a priori known time intervals assigned to this node allowing for collision-free access to the communication network.

Chapter 3

SIDERA

This chapter introduces SIDERA, a simulation model for safety-critical fault-tolerant real-time systems, inspired by the Time-Triggered Architecture TTA and the Time-Triggered Protocol TTP. SIDERA is an acronym for *SI*mulation model for *DE*pendable *R*ealtime *A*rchitectures.

The chapter is structured as follows: we start with an overview, describing the software environment and the principle of operation. Then, the internal structure of SIDERA is given; we describe the TTP protocol services that are covered by SIDERA as well as the simulated physical properties of the systems under investigation (i.e. possible communication system topologies, message transmission delays and the occurrence of faults). The chapter closes with a description of the external structure of SIDERA, containing the structure of the configuration file and the output that is generated by the simulation model.

3.1 Overview

3.1.1 Software environment

SIDERA has been developed using MATLAB/Simulink [1]. Simulink is a software package for modelling, simulating and analyzing dynamical systems. It supports linear and nonlinear systems, modelled in continuous time, sampled time, or a hybrid of the two. For modelling, Simulink provides a graphical user interface for building models as block diagrams. Once created, Simulink models can be run in the MATLAB environment, which is convenient to use,

[1] Mathworks. http://www.mathworks.com

but not suitable for simulation models with lots of calculations per simulation step because simulation time very soon becomes unacceptable (e.g. 12 hours simulation time for 0.001 seconds realtime). Furthermore, to use SIDERA one would need to have MATLAB installed and knowledge about how to use this tool.
Fortunately, MATLAB provides a mean to create stand-alone applications - the real-time-workshop (RTW), which transforms a Simulink model (which uses S-functions [2] written in C/C++) to an executable running on the platform for which it has been compiled.
SIDERA is a Win32 application which has been tested under the operating systems Windows [3] 2000 and WindowsXP.

3.1.2 Principle of operation

A SIDERA system consists of an arbitrary number of clusters. The relationship between the clusters is defined via a *cluster map* which is part of the SIDERA configuration file. Each cluster consists of up to 64 nodes (which equals the maximum number of slots that can be handled by the Time-Triggered Protocol TTP ([TTT99])).
Figure 3.1 shows the principle of operation of SIDERA. At simulation start a configuration file is read and the simulation runs according to the contents of the configuration file. The data produced during simulation is written to log files which can be evaluated when simulation has completed. The following sections deal with the contents and the format of the configuration file and the simulation log files.

Figure 3.1: SIDERA - principle of operation

[2] user-programmed functions
[3] Windows is a registered trademark of the Microsoft Corporation.

3.2 Internal structure

SIDERA focuses on the simulation of the following services that are provided by the time triggered protocol TTP:

- Local time
- Communication
- Global timebase
- Membership
- Protocol error handling
- Startup and reintegration

The simulation covers the functional aspects of the TTP protocol services. Physical parameters of simulation are

- Communication system topology
- Transmission delays
- Occurrence of faults

3.2.1 Protocol services

Local time

All nodes establish *local time* (see Section 2.4.2) as shown in Figure 3.2.

Each node maintains a quartz oscillator which is subject to *drift*. The systematic and the stochastic part of the drift (see Section 2.4.1) can be adjusted individually for each node and have an impact on the duration of the *oscillator tick*. The oscillator tick is the input for the node's *microtick generator* that produces a microtick each OSC oscillator ticks. The generation of the *macrotick* is based on the microtick and the *microtick-macrotick conversion factor* $MMCF$ that determines the number of microticks per macrotick. $MMCF$ consists of an integer and a fractional part. The generation of the macrotick additionally depends on the current value of the *clock state correction term* $CSCT$, which is periodically delivered by the clock synchronization algorithm. The macrotick counter of a node represents the node's view of global time (see Section 2.6.3).

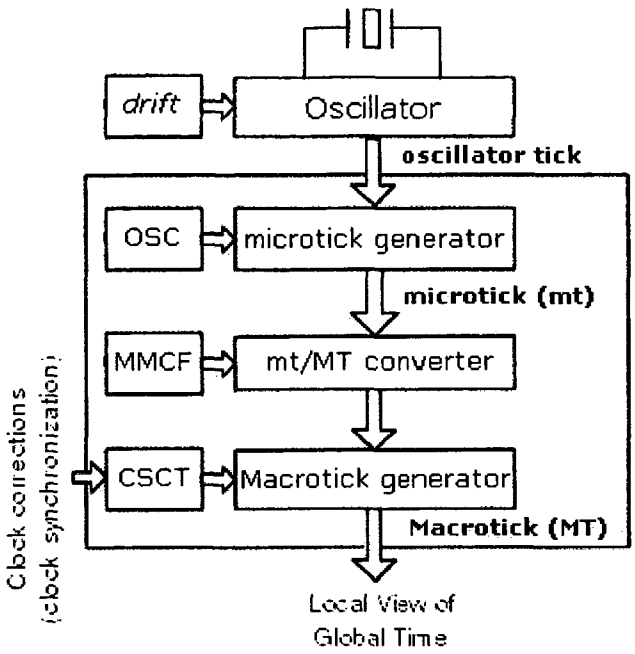

Figure 3.2: Local time

Communication

The access to the communication medium is based on a Time Division Multiple Access (TDMA) strategy. Realtime is divided into *slots*. Slots are assigned to nodes. A list of slot entries forms a *communication schedule*. Figure 3.3 shows the principle of operation.

Each node traverses the communication schedule in a cyclic manner. A node enters a slot if its macrotick counter reaches *slot_start_time* (Table 3.2). If *nodeID* (Table 3.1) is equal to *LogicalSenderName*, the node is a sender in the current slot and sends a message when its macrotick counter reaches *msg_send_time*, else the node is a receiver. Each node is assigned one slot in which it is allowed to send. The sequence of slots in which each node sends at most one message forms a *TDMA round*.

Global timebase

The central element in the TTA is the global notion of time (see Section 2.6.3). The nodes of a cluster execute a distributed fault-tolerant clock

nodeID	node identifier
OSC	oscillator ticks per microtick
MMCF	microticks per macrotick
sys_drift	systematic drift rate in sec/sec
stoch_drift	stochastic drift rate in sec/sec
gateway_node	gateway node flag
time_master_node	time master node flag
free_running_MT_int	free running macroticks internal clock synchronization
free_running_MT_ext	free running macroticks external clock synchronization
CF	coldstart allowed flag
cold_start_max	maximum number of frames allowed to be sent in state COLD START
distance_to_medium	distance to communication medium

Table 3.1: Node specific offline parameters

synchronization algorithm and establish a global timebase by periodic adjustments of their local clocks according to the clock state correction term delivered by the clock synchronization algorithm.

Time difference capturing. A node has to know the deviation of its local clock to the clocks of the other nodes to keep synchronized. Time difference capturing is the process of estimating the deviation of a local clock to a remote clock. In a slot with the SYF flag set (Table 3.2), all nodes use the *observed* arrival time of an incoming message for the estimation of the deviation of their local clocks from the sender's clock. For the communication schedule

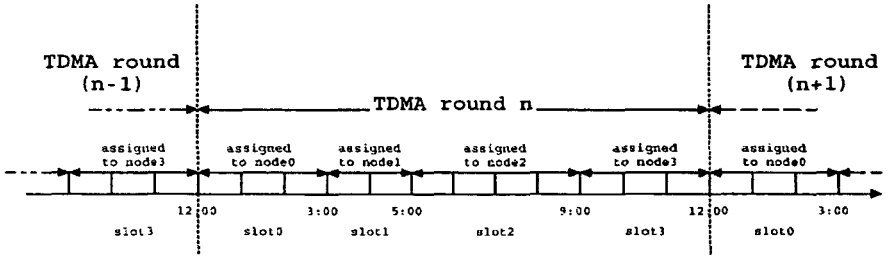

Figure 3.3: Communication medium access strategy

slot_start_time	start time of slot in macroticks
LogicalSenderName	nodeID of the sender
msg_send_time	message send time in macroticks
CS	clock synchronization flag
SYF	time difference capturing flag

Table 3.2: Slot entry

is available at all nodes, the receivers know the *expected* arrival time (stored in the slot entry, Table 3.2). To get the deviation from the sender's clock, all receivers determine the difference between the observed arrival time and the expected arrival time in terms of local microticks. The result is stored in *deltas*, a push-down stack of depth four that is used for the calculation of the clock state correction term. Each time an estimate is added to *deltas*, the oldest estimate is discarded.

deltas[4]	time difference capturing stack
slot_number	current slot number in communication schedule
corr_term_int	current internal clock state correction term
corr_term_ext	current external clock state correction term
frame_ack_counter	frame acknowledge counter
frame_invalid_counter	invalid frame counter
frame_fail_counter	failed frame counter
iframe_counter	frames sent in state COLD START
c_state	current C-State of controller

Table 3.3: Node specific runtime parameters

Internal clock synchronization. At the end of a slot with the CS flag set (Table 3.2), all nodes calculate an internal clock state correction term *corr_term_int* on the base of the estimates contained in *deltas* according to the fault-tolerant average (FTA) algorithm ([Kop97], p.62): the minimum and the maximum estimates are discarded, the internal clock state correction term becomes the average of the two remaining estimates.

$$corr\_term\_int = \lfloor \frac{(\sum_{i=1}^{4} deltas_i) - min(deltas_i) - max(deltas_i)}{2} \rfloor^4 \quad (3.1)$$

Positive values of *corr_term_int* indicate that the local clock is running fast, negative values that the local clock is running slow against cluster time.

Application of the internal clock state correction term. All nodes periodically adjust the number of microticks per macrotick according to *corr_term_int*. Every *free_running_MT_int* (Table 3.1) macroticks the duration of the current macrotick is extended/shortened by one microtick (according to the sign of *corr_term_int*) until *corr_term_int* is exhausted. The local clocks run free afterwards until the next synchronization instant marked in the communication schedule.

External clock synchronization. The global time of a cluster is externally synchronized to an external time reference by means of a *time master node* in this cluster which has access to a *reference time server* (see Section 2.5.2). A node with the *time_master_node* flag set periodically calculates the deviation from external reference time to local time in seconds and distributes this deviation to the other nodes at its *msg_send_time* (see Table 3.2) in the *delta_ext* field of the message (see Table 3.4).
Upon reception of a message from a time master node all nodes calculate an external clock state correction term *corr_term_ext* (Table 3.3) in terms of local microticks.

Application of the external clock state correction term. All nodes periodically adjust the number of microticks per macrotick according to *corr_term_ext*. Every *free_running_MT_ext* (Table 3.1) macroticks the duration of the current macrotick is extended/shortened by one microtick (according to the sign of *corr_term_ext*) until *corr_term_ext* is exhausted. The local clocks run free afterwards until the next external synchronization instant (i.e. until reception of the next message from a time master node).

Membership service

A membership service provides each node with a consistent view of the operational state of the other nodes within its cluster. The membership problem is a well-known and investigated problem in distributed systems. [HS95] describes the abstract properties of membership. [BP00a] deal with membership and clique avoidance mechanisms in the TTA. A formal verification of the TTP group membership algorithm is given in [Pfe00].

Message format. An active node sends a message (*frame*) each TDMA round containing the following information:

Time	Time in macroticks
MEDL_entry	Current slot number in communication schedule
Membership	Membership vector
delta_ext	deviation from reference time server provided by time master node

Table 3.4: Message format

The message entry *delta_ext* is used by the time master node only. All other nodes set this entry to 0 when sending their messages.

Membership. The membership vector is a bitfield containing one bit entry for each node. Active nodes are set to 1 and inactive nodes are set to 0. The membership vector at a node represents this node's view of the state of the ensemble of nodes. Each sender considers itself as operational and sets its corresponding bit in the *Membership* entry of the message it sends. Each receiver considers a sender as operational (and sets the sender's bit in the local membership vector) when it receives a *valid* frame during the communication schedule slot assigned to the sender. A frame is *valid* at the receiver if sender and receiver agree in *Time* (with a maximum deviation of one macrotick), *MEDL_entry* and *Membership*, else *invalid*. A receiver considers a sender as failed (and clears the sender's bit in the local membership vector) if it receives an *invalid* frame or no frame at all (*null frame*) during the slot of the sender.

Clique avoidance. The *clique avoidance* algorithm shall prevent the cluster from being partitioned into disjoint subsets, so-called *cliques*. Cliques are sets of nodes that consider all nodes within their clique as operational and all other nodes as failed. To avoid cluster partitioning, each node checks once per TDMA round whether it is in agreement with a *majority* of the nodes within its cluster. This check is based on a set of counters and works as follows:

Each node maintains a *frame_ack_counter*, a *frame_invalid_counter* and a *frame_fail_counter* (see Table 3.3). The node increments the *frame_ack_counter* when it receives a valid frame, the *frame_invalid_counter* when it receives an invalid frame and the *frame_fail_counter* if it receives no frame

at all in the current slot. When a node enters its sending slot (i.e. once per TDMA round), it checks the counters. If *frame_ack_counter* is bigger than the sum of *frame_fail_counter* and *frame_invalid_counter*, it clears the counters and remains active. Else the node is in disagreement with the majority of the cluster and quits operation.

Protocol error handling

A node quits operation upon detection of a protocol error. The following protocol errors are detected and handled:

Clock synchronization error. When the clock state correction term exceeds half the duration of a macrotick, the node detects a synchronization error.

Acknowledgement error. The clique avoidance logic has detected that the node is in disagreement with the majority of the cluster.

Communication system blackout. The node detects that no other node has sent a frame during the last TDMA round which is the case if

$$frame\_ack\_counter + frame\_invalid\_counter = 0 \tag{3.2}$$

(see Table 3.3).

Startup and reintegration

SIDERA provides a *startup service* for cluster startup (after initial power-on of the cluster and for cluster startup after a communication system blackout, see Section 3.2.1) and a *reintegration service* for the reintegration of nodes that have become inactive due to the detection of a protocol error.

[Lön99] analyzes different startup algorithms for TDMA communication within a group of computers connected via a broadcast bus. [SP01] presents a startup algorithm for synchronous distributed fault-tolerant systems. Formal analysis of a TTP-like startup algorithm can be found in [SRSP04].

Timeouts. The following node-specific timeouts are of importance for correct operation of the node in all operational states.

- **Startup timeout**
  The startup timeout $\tau_i^{startup}$ of a node which is assigned TDMA slot i is equal to the sum of the durations of all slots prior to slot i.

$$\tau_i^{startup} = 0 \quad if\ i = 0 \tag{3.3}$$

$$\tau_i^{startup} = \sum_{j=1}^{i} \tau_{j-1}^{slot} \quad if\ i > 0 \tag{3.4}$$

  τ_j^{slot} is the duration of the slot assigned to node j.

- **Cold start timeout**
  The cold start timeout of a node $\tau_i^{coldstart}$ is the sum of its startup timeout $\tau_i^{startup}$ and the duration of a single TDMA round τ^{round}.

$$\tau_i^{coldstart} = \tau_i^{startup} + \tau^{round} \tag{3.5}$$

- **Listen timeout**
  The listen timeout of a node τ_i^{listen} is the sum of its startup timeout $\tau_i^{startup}$ and the duration of two TDMA rounds τ^{round}.

$$\tau_i^{listen} = \tau_i^{startup} + 2 \times \tau^{round} \tag{3.6}$$

  This choice for the listen timeout ensures that the longest cold start timeout is shorter than the shortest listen timeout ([TTT99]).

Node states and state transitions. Figure 3.4 shows the state and state transition model of a node.

- **FREEZE**
  A node transits to the FREEZE state
  - after power-on of the cluster (start of simulation) or
  - upon detection of a protocol error (see Section 3.2.1).

  The node initializes its internal data structures, starts the listen timeout (see Section 3.2.1) and transits to the LISTEN state.

- **LISTEN**
  A node transits to the LISTEN state

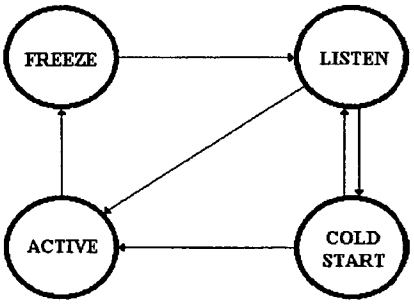

Figure 3.4: Node states and state transitions

- after initialization of its internal data structures.

Upon expiration of the listen timeout the node restarts the listen timeout and remains in state LISTEN if

- no valid frame was received from any other node and
- the conditions for entering COLD START state are not fulfilled.

- **COLD START**
  A node transits to the COLD START state upon expiration of the listen timeout if

  - the node is allowed to enter COLD START state (CF flag is set, see Table 3.1) and
  - *iframe_counter* (see Table 3.3) is less than *cold_start_max* (Table 3.1).

  The node broadcasts a frame, increments *iframe_counter* and starts the cold start timeout.

  Upon expiration of the cold start timeout the node increments *iframe_counter*, broadcasts another frame and remains in state COLD START if

  - no valid frame was received from any other node and
  - *iframe_counter* is less than *cold_start_max*.

  If the node is not allowed to broadcast another frame, it starts the listen timeout and transits to state LISTEN.

- **ACTIVE**
  A node transits to the ACTIVE state
  - from state LISTEN
    upon reception of a valid frame (see Section 3.2.1).
  - from state COLD START
    upon reception of a valid frame. The *iframe_counter* (see Table 3.3) is cleared.

3.2.2 Communication system topology

SIDERA provides simulation of bus and star topologies. It is also possible to define *guardians*. A guardian is a unit over which the communication medium is accessed. The guardian shall prohibit faulty nodes from sending outside their sending slot in the communication schedule. In a bus architecture, the *bus guardian* is a part of the node itself. In a star topology, all nodes access the communication medium via a dedicated *guardian node* which blocks all messages that are faulty or untimely from the guardian's point of view. The bus guardian approach and the central guardian approach in the TTA are described in [Tem99] and [BKS03], respectively.

3.2.3 Transmission delay

The *transmission delay* is the realtime it takes for a message to travel through the communication medium from a sender to a receiver. The transmission delay depends on various factors like the physical properties of the communication medium, the access strategy to the communication medium, the physical distance between sender and receiver etc. In the TTA, due to the collision-free TDMA communication medium access scheme, the transmission delay only depends on the physical properties of the communication medium and the distance between the sender and the receiver of a message.

For notational convenience, we introduce the notion of a communication medium with zero transmission delay. For each node, the *distance_to_medium* (see Table 3.1) in meters can individually be defined. Each node is connected to the medium via a *communication link* of *distance_to_medium* length. The communication links are characterized by a *signal_speed* in meters per second which defines the time it takes for a signal to pass through the communication link. The signal speed accounts for physical properties of the links and is common to all communication links in a cluster.

Be δ_i the transmission delay between node i and the communication medium:

$$\delta_i = distance\_to\_medium_i \times signal\_speed \tag{3.7}$$

In the following, $\Delta_{i,j}^{trans}$ denotes the transmission delay between node i and node j.

Each node "receives" its own message without delay:

$$\forall i,j,\ i=j : \Delta_{i,j}^{trans} = 0 \tag{3.8}$$

Bus topology

For a bus topology, transmission delay $\Delta_{i,j}^{trans}$ between node i and node j is equal to

$$\forall i,j,|i-j| = 1 : \Delta_{i,j}^{trans} = \delta_i + \delta_j \tag{3.9}$$

$$\forall i,j,|i-j| > 1 : \Delta_{i,j}^{trans} = \delta_i + \delta_j + 2 \times \sum_{k=i+1}^{j-1} \delta_k \tag{3.10}$$

Equation 3.9 describes the transmission delay between "neighboring" nodes (i.e. there is no node "between" node i and node j which means that node i is a successor or a predecessor to node j, respectively). Equation 3.10 describes the transmission delay between node i and node j with at least one node k "in-between". Figure 3.5 shows the principle of operation. Assume that node i sends a message. It follows from Equation 3.9 that node k receives the message with a delay of

$$\Delta_{i,k}^{trans} = \delta_i + \delta_k \tag{3.11}$$

The message from node i passes node k and arrives at node j with a delay of

$$\Delta_{k,j}^{trans} = \delta_k + \delta_j \tag{3.12}$$

The transmission delay between node i and node k follows from Equations 3.11 and 3.12 as

$$\Delta_{i,k}^{trans} + \Delta_{k,j}^{trans} = \delta_i + \delta_j + 2 \times \delta_k \tag{3.13}$$

which is equivalent to Equation 3.10 with $|i-j| = 2$.

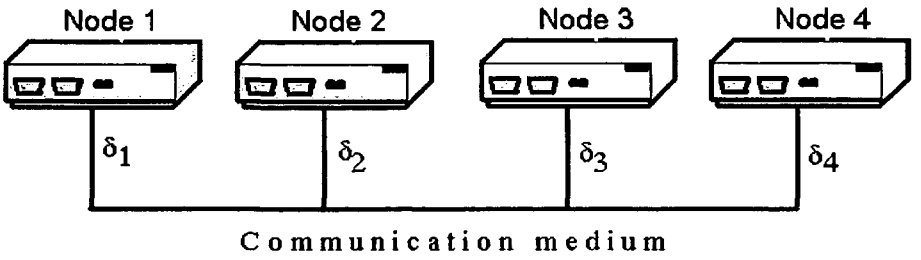

Figure 3.5: Transmission delay - bus topology

Star topology

For a star topology, the transmission delay $\Delta_{i,j}^{trans}$ between node i and node j is equal to

$$\Delta_{i,j}^{trans} = \delta_i + \delta_j + \delta_{guardian} \tag{3.14}$$

$\delta_{guardian}$ is the "switching time" which accounts for the delay induced by the message correctness check performed by the central guardian. Figure 3.6 shows the principle of operation.

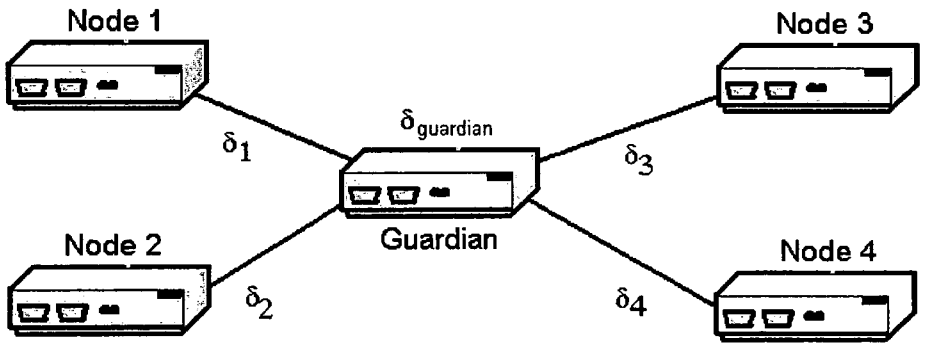

Figure 3.6: Transmission delay - star topology

3.2.4 Fault injection

SIDERA provides a fault injection module that allows to test the stability of the systems under investigation in the presence of faulty nodes and node failures. A node can exhibit incorrect behavior with regard to

- Crash failures
- Transmission faults
- Clock drift rate faults

A detailed description of the fault injection module is given in Section 3.3.1. Fault injection is necessary for the investigation of the stability of a system in the presence of faults. [Ade03] deals with the assessment of error detection mechanisms in the TTA by means of fault injection. The assumption coverage under different failure modes in the TTA is investigated in [BKP95]. [KBP01] is about the toleration of arbitrary node failures in the TTA.

3.3 External structure

3.3.1 The configuration file

The simulation is started by invoking the executable *sidera.exe*. The configuration file *model_param.dat* has to exist in the current working directory. There are parameters that are *mandatory* for the simulation and that have to exist in the configuration file; *optional* parameters may be omitted. In the following sections, all parameters of the configuration file are described. Mandatory parameters are marked with *M* and optional parameters are marked with *O*. Detailed information about the structure of the configuration file can be found in Appendix A.

System specific parameters

System specific parameters affect all clusters and all nodes within the system.

1. *sim_time* (M)
   The simulation time in terms of seconds.
2. *num_clusters* (M)
   The number of clusters within the system.
3. *ext_corr_rate_max* (O)
   In a multi-cluster system, this entry defines the maximum external correction rate in seconds per second, i.e. the maximum rate at which

the cluster time of a slave cluster is drawn towards the cluster time of a master cluster (see Section 2.5.3). It should be in the order of $10^{-4}\frac{sec}{sec}$ to ensure that the drift of cluster time is bounded and that internal clock synchronization is not affected ([KKMS95], [BP00b]). This entry becomes mandatory if *num_clusters* (see 2) is greater than 1.

4. *sample_rate_mt_log* (O)
   The update rate of the microtick log file in terms of seconds. If 0, microtick logging is disabled.

5. *sample_rate_ict_log* (O)
   The update rate of the internal clock state correction term log file in terms of seconds. If 0, internal clock state correction term logging is disabled.

6. *sample_rate_ect_log* (O)
   The update rate of the external clock state correction term log file in terms of seconds. If 0, external clock state correction term logging is disabled.

7. *sample_rate_MT_log* (O)
   The update rate of the macrotick log file in terms of seconds. If 0, macrotick logging is disabled.

8. *sample_rate_user_log* (O)
   This entry is reserved for development purposes and should be set to 0, if defined.

9. *cluster_map*
   In a multi-cluster system, the cluster map defines the relationship between the clusters and the flow of timing information. It consists of one entry

   - *num_cluster_map_entries* **(O)**
     The number of entries the cluster map consists of. 0 if no cluster map is needed.

   and *num_cluster_map_entries* pairs of

   - *master* (M)
     The number of the cluster which is a master to the cluster defined in entry *slave*.

- *slave* (M)
  The number of the cluster which is a slave to the cluster defined in entry *master*.

entries. Each pair of *master* and *slave* entries increments *num_cluster_map_entries* by one. A multi-cluster system with no cluster map defined consists of *num_clusters* (see 2) free running clusters.

Example: The following example shows a cluster map for a system consisting of three clusters. Cluster 0 is a master to cluster 1 which in turn is a master to cluster 2.

```
//Cluster_Map
    2       //num_cluster_map_entries
    0       //master
    1       //slave
    1       //master
    2       //slave
```

Cluster specific parameters

Cluster specific parameters affect all nodes of a cluster.

1. *num_nodes* (M)
   The number of nodes within the cluster.

2. *slot_len_ref* (M)
   The slot length of a reference cluster configuration in macroticks. In the current implementation of SIDERA, all nodes of a cluster must have equal slot length.

3. *slot_len_sim* (M)
   The slot length that shall be used for the simulation of the reference cluster configuration in macroticks. By choosing *slot_len_sim* smaller than *slot_len_ref*, simulation time is shortened by the factor $\frac{slot\_len\_sim}{slot\_len\_ref}$ compared to the runtime of the original configuration (one second runtime in the "real world" equals $\frac{slot\_len\_sim}{slot\_len\_ref}$ seconds simulation time). The motivation for choosing a shorter simulation slot length than in the original cluster configuration (shorter simulation time, smaller simulation log files) is described in Section 3.4.1.

4. *delta_0* (O)
   If not 0, *delta_0* determines the systematic drift rates of the nodes of the cluster. *delta_0* is uniformly distributed over all nodes of the cluster within the interval $[-\frac{\Delta_0}{2}, +\frac{\Delta_0}{2}]$ according to Equation 3.15. Be δ_i^{sim} the systematic drift rate of node i.

$$\delta_i^{sim} = (-\frac{\Delta_0}{2} + \frac{\Delta_0}{num\_nodes - 1}) \times \frac{slot\_len\_ref}{slot\_len\_sim} \tag{3.15}$$

5. *symm_capturing* (O)
   If 0, the time difference capturing mechanism is asymmetric, else symmetric.

6. *recover_from_freeze* (O)
   If not 0, all nodes of the cluster remain in state FREEZE once they have entered it, e.g. due to a protocol error. This setting is useful for investigation of stability of cluster designs with regard to clock synchronization.

7. *self_calibration* (O)
   If not 0, all nodes of the cluster calibrate their local clocks on the base of their internal clock state correction terms that have been observed in the first *self_calibration* TDMA rounds after the node has entered state ACTIVE or after the node has triggered recalibration of its local clock. A node performs self-calibration whenever it transits to state ACTIVE. The maximum number of TDMA rounds available for self calibration is 100 in the current implementation of SIDERA.

8. *use_master_clock* (O)
   If not 0, entry *master_clock_node* is the id of the node that shall be used as the rate master node. If 0, no rate master node exists.

9. *master_clock_node* (O)
   If *use_master_clock* (see 8) is not 0, this is the id of the node that shall be used as the rate master node. If not defined and *master_clock_node* is not 0, node number 0 becomes the rate master node. A rate master node never calibrates its local clock. All other nodes calibrate their local clocks against the clock rate of the rate master node. The idea behind the rate master node concept is described in Chapter 4 and in [KAH04].

10. *recalibration_threshold* (O)
    If not 0, this entry defines the maximum clock state correction term

that is tolerated by all nodes. If the absolute value of the clock state correction term at a node exceeds *recalibration_threshold*, the node recalibrates its local clock (unless it is a rate master node). The length of the interval is *self_calibration* TDMA rounds (see 7).

11. *central_guardian* (O)
    If not 0, entry *guardian_node* (see 12) is the id of the node that shall be used as the central guardian. If 0, no central guardian exists.

12. *guardian_node* (O)
    If *central_guardian* (see 11) is not 0, this is the id of the node that shall be used as the central guardian. If not defined and *central_guardian* is not 0, node number 0 becomes the central guardian.

13. *div_ext_corr_term* (O)
    If not 0, the external clock state correction term is divided by two before it is applied. This setting is useful for multi-cluster configurations with feedback loops where cluster times tend to oscillate as shown in the course of simulation experiments in Section 5.2.5.

14. *time_flooding* (O)
    If not 0, the cluster performs no cluster startup until there is a valid time information available from the gateway node the time master node of this cluster is connected to. This setting is useful for the investigation of multi-cluster startup scenarios given in [SPHH04].

15. *no_cluster_startup* (O)
    If not 0, no cluster startup is performed. All nodes immediately enter state ACTIVE after start of simulation.

16. *no_sync_nodes* (O)
    If not 0, the local clocks of the nodes in the cluster are running free, no clock synchronization is performed.

17. *signal_speed* (O)
    If not 0, this entry defines the signal speed through the communication links in meters per second; it reflects the physical properties of the communication links and is used for the calculation of the transmission delay between sender and receiver of a message (see Section 3.2.3).

18. *prop_delay_corr* (O)
    If not 0, the transmission delay induced by *signal_speed* is corrected by the nodes at message reception time. That means that the estimate

of the message send time is corrected at the receiver according to the transmission delay between sender and receiver (see Section 3.2.3).

19. *spread_corr_int* (O)
If 0, a microtick is corrected each macrotick until the internal clock state correction term is exhausted. If not 0, then a microtick is corrected each $\frac{spread\_corr\_int \times slot\_len\_sim}{\frac{\Pi}{2}}$ macroticks until the internal clock state correction term is exhausted. Π is the precision of the global timebase in terms of local microticks (see Section 2.5.1).

20. *spread_corr_ext* (O)
If 0, a microtick is corrected each macrotick until the external clock state correction term is exhausted. If not 0, then a microtick is corrected each $\frac{spread\_corr\_ext \times slot\_len\_sim}{\frac{\Pi}{2}}$ macroticks until the external clock state correction term is exhausted. Π is the precision of the global timebase in terms of local microticks (see Section 2.5.1).

21. Message descriptor list (MEDL)
The MEDL defines the communication schedule of the cluster. It consists of one entry

    - *num_slots* (M)
    The number of entries the MEDL consists of.

    and *num_slots* slot entries consisting of

    - *LogicalSenderName* (M)
    The id of the node which is allowed to send in this slot (the *owner*).
    - *CS* (O)
    If not 0, all nodes calculate an internal clock state correction term for their local clocks at the end of this slot.
    - *SYF* (O)
    If not 0, all nodes use the difference between expected and observed arrival time for the estimation of the deviation of their local clocks from the sender's clock in this slot.

    entries. Each set of *LogicalSenderName*, *CS* and *SYF* entries increments *num_slots* by one.

Example: The following example shows a MEDL for a cluster consisting of four nodes.

```
//MEDL
    4       //num_slots
    //slot_0
    0       //LogicalSenderName
    0       //CS
    1       //SYF
    //slot_1
    1       //LogicalSenderName
    0       //CS
    1       //SYF
    //slot_2
    2       //LogicalSenderName
    0       //CS
    1       //SYF
    //slot_3
    3       //LogicalSenderName
    1       //CS
    1       //SYF
```

Node specific parameters

Node specific parameters affect the node for which they are defined. A node parameter block consists of the following entries:

1. *MMCF* (M)
   The number of microticks per macrotick. Integer and fractional values are allowed.

   Examples: 20, 19.664

2. *OSC* (M)
   The number of oscillator ticks per microtick.

3. *gateway* (O)
   If not 0, this node is a gateway node.

4. *time_master_node* (O)
   If not 0, this node is a time master node.

5. *coldstart_flag* (O)
   If not 0, this node is allowed to enter state COLDSTART.

6. *max_coldstart_frames* (O)
   If not 0, this is the number of coldstart frames the node is allowed to send in state COLDSTART. Only taken into account if *coldstart_flag* is not 0. A detailed description of the nodes state and state transition model is given in Section 3.2.1.

7. *sys_drift* (O)
   The systematic drift of the node in seconds per second. Only taken into account if *delta_0* is 0. Otherwise, this entry is overwritten with a calculated systematic drift value according to equation 3.15. If *delta_0* is 0, this entry becomes mandatory.

8. *distance_to_medium* (O)
   If not 0, this entry defines the distance between the node and the communication medium in meters. This entry is used for the calculation of the transmission delay between a sender and a receiver (see Section 3.2.3).

Example: The following example shows the node specific parameters for a cluster consisting of four nodes. Assume that *delta_0* is set to 0 in the cluster specific parameters.

```
//Node_0
    20          //MMCF
    10          //OSC
    1           //coldstart_flag
    1           //max_coldstart_frames
    -0.000004   //sys_drift
//Node_1
    20          //MMCF
    10          //OSC
    -0.000002   //sys_drift
//Node_2
    20          //MMCF
    10          //OSC
    +0.000001   //sys_drift
//Node_3
    20          //MMCF
    10          //OSC
    +0.00002    //sys_drift
```

Fault injection parameters

SIDERA allows the simulation of node faults and node failures. A node can exhibit incorrect behavior with regard to

- Crash failures
  A nodes transits to state FREEZE once or periodically.
- Transmission faults
  A node sends an invalid message once or periodically.
- Clock drift rate faults
  A node changes the drift rate of its local clock once or periodically.

1. **Crash failures**
   Crash failure parameters consist of the following entries:
   - *freeze_at_slot* **(O)**
     If not 0, the node enters state FREEZE at this slot.
   - *freeze_duration* **(O/M)**
     The number of slots the node stays in state FREEZE.
   - *freeze_num* **(O/M)**
     The node enters state FREEZE for *freeze_num* times.
   - *freeze_repeat_rate* **(O/M)**
     The node enters state FREEZE every *freeze_repeat_rate* slots.

   If *freeze_at_slot* is not 0, all crash failure parameters are mandatory.

   Example: The following example shows sample crash failure parameters. Node 1 enters state FREEZE in slot 100, slot 150 and slot 200 for a duration of 10 slots.

```
//Node_1
    20          //MMCF
    10          //OSC
    1           //coldstart_flag
    1           //max_coldstart_frames
    -0.000004   //sys_drift
        //fault_injection
        100     //freeze_at_slot
```

```
10      //freeze_duration
3       //freeze_num
50      //freeze_repeat_rate
```

2. **Transmission faults**
   Transmission fault parameters consist of the following entries:

   - *faulty_msg_in_round* (O)
     If not 0, the node sends an invalid message in this TDMA round.
   - *faulty_msg_num* (O/M)
     The number of invalid messages the node will send.
   - *faulty_msg_repeat_rate* (O/M)
     The node sends an invalid message every *faulty_msg_repeat_rate* TDMA round.

   If *faulty_msg_in_round* is not 0, all transmission fault parameters are mandatory.

   Example: The following example shows sample message fault parameters. Node 1 sends a faulty message in round 100, round 150 and round 200.

```
//Node_1
    20          //MMCF
    10          //OSC
    1           //coldstart_flag
    1           //max_coldstart_frames
    -0.000004   //sys_drift
        //fault_injection
        100     //faulty_msg_in_round
        3       //faulty_msg_num
        50      //faulty_msg_repeat_rate
```

3. **Clock drift rate faults**
   Clock drift rate fault parameters consist of the following entries:

   - *change_drift_at_slot* (O)
     If not 0, the node changes its drift rate in this slot. This is done by adding δ_{drift} to the current systematic drift rate of the node.

- *change_drift_factor* **(O/M)**
  This entry determines the amount and the direction of the drift rate change.

$$\delta_{drift} = sys\_drift \times change\_drift\_factor \tag{3.16}$$

  sys_drift is the original value of the systematic drift as defined in the configuration file.
- *change_drift_num* **(O/M)**
  The systematic drift is changed by δ_{drift} for $change\_drift\_num$ times.
- *change_drift_repeat_rate* **(O/M)**
  The systematic drift is changed by δ_{drift} every $change\_drift\_repeat\_rate$ slots.

If $change\_drift\_at\_slot$ is not 0, all clock drift rate fault parameters are mandatory.
To change the systematic drift value from a value $drift_{start}$ to a value $drift_{end}$, the value for $change\_drift\_factor$ has to be calculated according to

$$change\_drift\_factor = -\frac{drift_{end} - drift_{start}}{drift_{start} \times change\_drift\_num} \tag{3.17}$$

If $change\_drift\_factor$ is constant, $change\_drift\_repeat\_rate$ determines the duration of the drift rate change.

Example: The following example shows sample clock drift rate fault parameters. Starting at slot 10, node 1 changes its systematic drift rate from -5×10^{-5} sec/sec to $+5 \times 10^{-5}$ sec/sec within a duration of 80 slots. $change\_drift\_factor$ was calculated according to Equation 3.17.

```
//Node_1
    20          //MMCF
    10          //OSC
    1           //coldstart_flag
    1           //max_coldstart_frames
    -0.00005    //sys_drift
        //fault_injection
        8       //change_drift_at_slot
        -0.1    //change_drift_factor
        20      //change_drift_num
        4       //change_drift_repeat_rate
```

3.3.2 Model output

SIDERA produces logfiles for analysis of the simulation runs. The log files can be divided into the following parts:

- **Online log information**
  is available during a simulation run. It is written to *stdout*, and can thus be redirected to a file for documentation and evaluation purposes.
- **Offline log information**
  is not available until the completion of a simulation run.

Online log information

Online log information can be divided into the following groups:

1. **State information**
   Each node logs its state at its message send time (usually once per TDMA round). The node state information looks like the example below

```
R#0002  [0,1]  [ -3 +11  +7  +4] [I:+5] [P:16] [E:+0] [P:16]
               [M:3F] [T:0162]
```

   and consists of the following entries:

   - TDMA round (R#0002)
     The TDMA round in which the node is sending.
   - Node identifier ([0,1])
     A node is uniquely identified within a system by a pair of cluster and node number. In the above example the log information is from node 1 in cluster 0.
   - Time difference capturing stack ([-3 +11 +7 +4])
     The time difference capturing stack contains the estimated deviations of the local clock from the clocks of the last four sending nodes (which have sent in slots with the SYF flag set) in terms of microticks.

- Internal clock state correction term ([I:+5])
  The current value of the internal clock state correction term in terms of microticks.
- Precision ([P:16])
  The current cluster precision in terms of microticks (i.e. the current maximum deviation of two node's local clocks within the cluster).
- External clock state correction term ([E:+0])
  The current value of the external clock state correction term in terms of microticks. For the example above is taken from a simulation run with a single-cluster system, the external correction term is always 0.
- System precision ([P:16])
  The current system precision in terms of microticks (i.e. the current maximum deviation of two node's local clocks within the whole system). For the example above is taken from a simulation run with a single-cluster system, system precision is always equal to cluster precision.
- Membership vector ([M:3F])
  The membership vector representing the operational state of all nodes of the cluster the node belongs to.
- Global time ([T:0162])
  The node's current view of the internally synchronized cluster time in terms of macroticks.

2. **State change information**
   Each time a node undergoes a state change, it generates a log entry. The node state change information looks like the example below

```
[0,1]   FREEZE -> LISTEN [T:1]
```

and consists of the following entries:

- Node identifier ([0,1])
  A pair of cluster and node number. In the above example the log information is from node 1 in cluster 0.
- Old state (FREEZE)
  The state the node transits from.

- New state (LISTEN)
  The state the node transits to.
- Global time ([T:1])
  The node's current view of the internally synchronized cluster time in terms of macroticks.

3. **Protocol error information**
   Each time a node encounters a protocol error, it generates a log entry. The protocol error information looks like the example below

```
[0,1]   PROTOCOL ERROR: acknowledgement
[0,1]   ACTIVE -> FREEZE [T:300]
```

   and consists of the following entries:

   - Node identifier ([0,1])
     A pair of cluster and node number. In the above example the log information is from node 1 in cluster 0.
   - Protocol error (acknowledgement)
     The protocol error the node has encountered.

   After the detection of a protocol error, a node transits to state FREEZE causing a node state change information log entry (see 2).

4. **Fault information**
   Each time a node encounters a fault, it generates a log entry. The fault information looks like the example below

```
[0,1]   FAULT: clock drift (10 more times)
        -0.0005 -> +0.0001 [T:1200]
```

   and consists of the following entries:

   - Node identifier ([0,1])
     A pair of cluster and node number. In the above example the log information is from node 1 in cluster 0.

- Fault (clock drift (10 more times) -0.0005 -> +0.0001)
  The fault the node has encountered and the remaining number of faults of this kind to occur during simulation. In case of a clock drift fault, the systematic drift rate of the node before and after the fault is also logged.
- Global time ([T:1200])
  The node's current view of the internally synchronized cluster time in terms of macroticks.

5. **Simulation results**
   When simulation has completed, the results are logged. They look like the example below

```
RESULTS: \Test_1_A

-------------------------------------------------
Cluster#   Node drift rates   Precision   Drift rate
-------------------------------------------------
   0
           +8.65e-005
           +8.73e-005
           +8.88e-005
           +9.18e-005
           +9.33e-005
           +9.47e-005
                              17.30       +9.04e-005
-------------------------------------------------
System                        17.30       +9.04e-005
-------------------------------------------------
```

and consist of the following entries:

- Test case (*Test_1_A*)
  The name of the current working directory which contains the configuration file *model_param.dat*.
- Node drift rates (+8.65e-005 ... +9.47e-005)
  The drift rates of the different nodes of the cluster in terms of second per second.
- Precision (17.30)
  The precision that has been observed in terms of microticks.

- Drift rate (+9.04e-005)
  The cluster drift rate (i.e. the drift of cluster time against simulation time) that has been observed in seconds per second.

Offline log information

Offline log information is written into files in MATLAB raw format. For the evaluation of these files the MATLAB environment or any tool that is able to handle files in MATLAB raw format is necessary. The offline log information file format is given in Appendix B.

Offline log information can be divided into the following groups:

- **Microtick log information** (*microticks.mat*)
  If *sample_rate_mt_log* is not 0, the microtick counters of all nodes are logged with the specified sampling rate.

- **Internal correction term log information** (*corr_terms_int.mat*)
  If *sample_rate_ict_log* is not 0, the internal correction terms of all nodes are logged with the specified sampling rate.

- **External correction term log information** (*corr_terms_ext.mat*)
  If *sample_rate_ect_log* is not 0, the external correction terms of all nodes are logged with the specified sampling rate.

- **Macrotick log information** (*MACROTICKS.mat*)
  If *sample_rate_MT_log* is not 0, the macrotick counters of all nodes are logged with the specified sampling rate.

In the MATLAB environment, the following script can be used to generate figures from the simulation logfiles.

```
function y = my_plot (varargin)
% example: my_plot ('microticks')
variable = load (varargin{1});
dim = size (variable.data);
rows=dim(1);
columns = dim(2);
plot (variable.data (1, 1:columns),
      variable.data (2:rows, 1:columns));
```

3.4 Model verification

The topic of this section is the verification of SIDERA by means of reference tests against a VHDL model of a TTPC/C1 controller. [Bau99] contains a detailed description of the five test cases as well as a discussion of the results delivered by the VHDL simulation.

3.4.1 Considerations and prerequisites

Reduction of slot length

SIDERA produces considerable amounts of data. A prerequisite for proper evaluation of the test runs is the logging of the local times at all nodes of a cluster. The simulation logfile consists of *samples*. For our reference tests, a sample contains the simulation time and the local times. For a cluster with eight nodes, one sample has a size of 72 bytes. If the maximum sampling rate is specified, a sample is added to the logfile each tick of the simulation time (i.e. all 5×10^{-9} seconds, that are 2×10^8 samples per second simulation time).

All reference tests have a duration of 200 slots with a slot length of 186 macroticks. With this configuration, SIDERA produces a logfile of about half a GB size ($5,3568 \times 10^8$ bytes) if only the local times are recorded. Simulation logfile size can grow to 2 GB size and more if other parameters of interest are recorded additionally (e.g. the clock state correction terms over simulation time). To gain test results from reasonable amounts of data, the slot length used for the reference tests with SIDERA was reduced from 186 macroticks (*slot_len_ref*, see Section 3.3.1) to 12 macroticks (*slot_len_sim*, see Section 3.3.1). The reduction of the slot length has to be considered in the calculation of the systematic drift rates of the nodes δ_i^{sim}.

Application of systematic cluster drift rate Δ_0

The cluster drift rate value Δ_0 is uniformly distributed over all nodes within the interval $[-\frac{\Delta_0}{2}, +\frac{\Delta_0}{2}]$. Table 3.5 shows the assigned systematic drift values δ_i^{sim} for the reference test configurations consisting of six nodes according to Equation 3.15.

The factor $\frac{slot\_len\_ref}{slot\_len\_sim}$ accounts for the different slot lengths and ensures that the impact of the systematic drift on the local clocks is equal in both models.

Node 0	$\delta_0^{sim} = -\frac{\Delta_0}{2} \times \frac{slot\_len\_ref}{slot\_len\_sim}$
Node 1	$\delta_1^{sim} = (-\frac{\Delta_0}{2} + \frac{\Delta_0}{5}) \times \frac{slot\_len\_ref}{slot\_len\_sim}$
Node 2	$\delta_2^{sim} = (-\frac{\Delta_0}{2} + 2 \times \frac{\Delta_0}{5}) \times \frac{slot\_len\_ref}{slot\_len\_sim}$
Node 3	$\delta_3^{sim} = (-\frac{\Delta_0}{2} + 3 \times \frac{\Delta_0}{5}) \times \frac{slot\_len\_ref}{slot\_len\_sim}$
Node 4	$\delta_4^{sim} = (-\frac{\Delta_0}{2} + 4 \times \frac{\Delta_0}{5}) \times \frac{slot\_len\_ref}{slot\_len\_sim}$
Node 5	$\delta_5^{sim} = +\frac{\Delta_0}{2} \times \frac{slot\_len\_ref}{slot\_len\_sim}$

Table 3.5: Node drift rates

Precision measurement

Precision is measured in terms of microticks. This only makes sense if all nodes of the cluster have the same microtick duration (which is the case in the reference tests). In case of differing microtick durations at the nodes, the local times at the different nodes would have to be calculated in terms of seconds; precision would have to be given in terms of seconds on the base of the transformed local times.

3.4.2 Reference test setup and results

Number of nodes	6
Microtick duration	50×10^{-9} seconds
Microticks per macrotick	20
Macrotick duration	10^{-6} seconds
Slot length	12 macroticks
Free running macroticks	0
TDMA round length	72 macroticks
Simulation time	200 slots

Table 3.6: Reference test common parameters

Tables 3.8 and 3.9 show the results of the reference tests. Precision and cluster drift rate are given for each test run and compared to the results that have been achieved with the VHDL reference model. The precision values available from the VHDL tests are precise, whereas the cluster drift rates measured in the VHDL tests are approximate values.

The cluster drift rate describes the drift of cluster time from simulation time in seconds per second. It is calculated from the median microtick

Test	Δ_0	CS flags	SYF flags
1	0.0005	Slot 0, 6, 12, ...	Every slot
2	0.0005	Slot 0, 4, 8, ...	Every slot
3	0.0005	Every slot	Every slot
4	0.0005	Every slot	Slot 1, 2, 3, 4
5	0.0001	Slot 0, 4, 8, ...	Every slot

Table 3.7: Reference test case specific parameters

counter at the end of simulation time. Positive values indicate that cluster time is late against realtime, negative values that cluster time is fast against realtime.

Test 4 with SIDERA becomes unstable after a few slots. Node 5 detects a clock synchronization error (see Section 3.2.1) in slot number 22 and fails. Nodes 0, 2, 3 and 4 fail in slot 29, 31, 62 and 81, respectively, due to an acknowledgement error (see Section 3.2.1). Finally, node 1 fails due to the detection of a communication system blackout (3.2.1).

Test 4 with the VHDL model shows the following behavior: Node 5 fails in slot 15 and node 4 in slot 25 due to a clock synchronization error. Nodes 2, 0 and 1 fail in slot 31, 35 and 66, respectively, due to an acknowledgement error. Finally, node 3 fails due to the detection of a communication system blackout.

Test	SIDERA	Reference
1	16.5322	16.34825
2	12.5801	12.9465
3	7.9785	8.230
5	3.5469	3.33615

Table 3.8: Precision (microticks)

Figure 3.7 shows the course of cluster precision for Reference Test 1 during a simulation time of 200 slots. After start of the simulation, the cluster startup phase takes place (see Section 3.2.1). After expiration of its listen timeout, Node 0 sends a coldstart frame that the remaining nodes integrate on. During the cluster startup phase, no clock synchronization takes place - the local clocks are running free (leftmost peak in Figure 3.7). In slot 12, all nodes are active and start internal clock synchronization. In Test 1, the cluster establishes a global timebase with a precision of 17 microticks.

Test	SIDERA	Reference
1	+7.4266	+8
2	-6.8087	-6
3	-18.172	-18
5	-3.7391	-5

Table 3.9: Cluster drift rate $\times 10^{-5}$ sec/sec

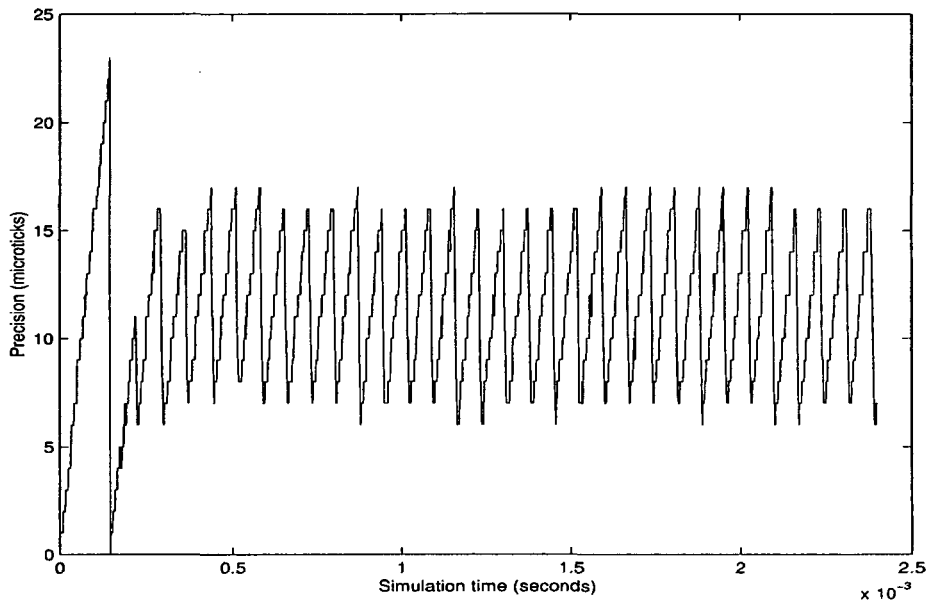

Figure 3.7: Reference Test 1 - Cluster precision

The course of cluster precision for reference test cases 2, 3 and 5 can be found in Appendix C.

SIDERA follows the behavior of the VHDL reference model with a maximum deviation in precision of 0,3664 microticks (Test 2) and a maximum deviation in cluster drift rate of $1,2609 \times 10^{-5}$ seconds (Test 5). In Test 4 SIDERA also behaves very similar to the VHDL reference model: one node after the other fails due to the detection of a protocol error within a duration of 84 slots.

Chapter 4

Cluster tuning

This chapter describes methods and means that improve the quality of synchronization within a cluster. It introduces the notion of node *calibration* which aims at manipulation of the frequencies of the local clocks such that they get into better agreement with an external time reference or with the internally synchronized cluster time. Static and dynamic calibration of the local clocks are described by formal analysis and evaluated in the course of experimental setups.

4.1 Cluster calibration - principle of operation

This section shows how the impact of clock drift on internal clock synchronization can be minimized.

4.1.1 Local clock properties

As described in Section 3.2.1, each node generates a macrotick after a given number of microticks. For the global time within a cluster proceeds in units of macroticks, the duration of the macrotick has to be the same for all nodes within a cluster. The microtick duration may be different at the nodes, because the nodes might have oscillators with different frequencies. However, we assume that within a cluster all nodes i use oscillators with the same nominal frequency f_i^{nom}. Consequently, the nominal microtick durations are equal at all nodes of a cluster which in turn means that the granularities of the approximate global times are about the same in all the nodes ([KO87]).

Table 4.1 sums up the oscillator frequencies and the duration of the microtick and the macrotick, respectively, which are common to all nodes.

f^{nom}	20MHz
microtick duration	50×10^{-9} seconds $(= \frac{1}{f^{nom}})$
Macrotick duration	10^{-6} seconds

Table 4.1: Local clock properties

4.1.2 Local clock parameters

The local clock parameter of node i $MMCF_i^{nom}$ defines the number of microticks per macrotick for node i, based on the node's nominal frequency f_i^{nom}. Each node i increases its macrotick counter each $MMCF_i^{nom}$ local microticks. $MMCF_i^{nom}$ consists of an integer and a fractional part. In a cluster with k nodes, for all nodes i the local clock parameters are

$$MMCF_i^{nom} = 20.00, \quad 1 \leq i \leq k \tag{4.1}$$

for the local clock properties defined in Table 4.1.

4.1.3 The impact of clock drift

For the following considerations, we assume that the drift rate ρ_i of all clocks i is constant within a given time interval T:

$$\forall t, 0 \leq t \leq T : \rho_i(t) = const = \rho_i \tag{4.2}$$

We furthermore assume that clock i and reference clock z have the same nominal frequency

$$\forall i, 1 \leq i \leq k : f_i = f_z \tag{4.3}$$

and therefore use the same local clock parameters:

$$\forall i, 1 \leq i \leq k : MMCF_i = MMCF_z \tag{4.4}$$

k is the number of nodes in the cluster.

The microtick counter $mt_i(t)$ of node i at time t is equal to

$$mt_i(t) = \lfloor t \times f_z \times (1 - \rho_i) \rfloor \tag{4.5}$$

The microtick counter at reference clock z (which has a drift rate of 0) at time t is equal to

$$mt_z(t) = \lfloor t \times f_z \rfloor \tag{4.6}$$

The macrotick counter $MT_i(t)$ of node i at time t is equal to

$$MT_i(t) = \lfloor \frac{mt_i(t)}{MMCF_z} \rfloor \tag{4.7}$$

The macrotick counter at reference clock z at time t is equal to

$$MT_z(t) = \lfloor \frac{mt_z(t)}{MMCF_z} \rfloor \tag{4.8}$$

The deviation between the microtick counters at clock i and reference clock z at time t follows from Equation 4.5 and Equation 4.6 as

$$mt_i(t) - mt_z(t) = -\lfloor (t \times f_z \times \rho_i) \rfloor \tag{4.9}$$

and the deviation between the macrotick counters at clock i and reference clock z at time t follows from Equation 4.7 and Equation 4.8 as

$$MT_i(t) - MT_z(t) = -\lfloor \frac{t \times f_z \times \rho_i}{MMCF_z} \rfloor \tag{4.10}$$

Equation 4.9 shows that the microtick of clock i drifts apart from the microtick of reference clock z proportional to the value of ρ_i. The same follows for the macrotick counter due to Equation 4.10.

4.1.4 Clock calibration

For the following considerations we assume that we have no possibility to interact with the oscillators on a physical level to accelerate or decelerate the local clocks (e.g. by changing the input voltages of the oscillators).

We can't do anything against the *drift* of the microtick counter of clock i. That means, that we cannot change the frequency f_i to account for drift rate ρ_i. However, we can prevent $MT_i(t)$ to drift apart from reference clock z.

Calibration based on the clock drift rate

Calibration of clock i means that we want to bring the macrotick counter of clock i and the macrotick counter of reference clock z at time t into agreement:

$$MT_i(t) = MT_z(t) \Rightarrow \frac{mt_i(t)}{MMCF_z} = \frac{mt_z(t)}{MMCF_z} \tag{4.11}$$

Equation 4.11 is true if clock i is a perfect clock

$$\forall t, 0 \leq t \leq T : \rho_i(t) = 0 \tag{4.12}$$

or if $MMCF_z$ is replaced by $MMCF_i$ at clock i such that $MMCF_i$ compensates drift rate ρ_i:

$$\frac{mt_i(t)}{MMCF_i} = \frac{mt_z(t)}{MMCF_z} \Rightarrow MMCF_i = MMCF_z \times (1 - \rho_i) \tag{4.13}$$

If the drift rate of clock i is known, the clock can be calibrated according to Equation 4.13.

Calibration based on the microtick counter

Clock i can also be calibrated using the microtick counters of clock i and reference clock z at time t. Again, we want to bring the macrotick counters of clock i and reference clock z at time t into agreement:

$$MT_i(t) = MT_z(t) \Rightarrow \frac{mt_i(t)}{MMCF_i} = \frac{mt_z(t)}{MMCF_z} \tag{4.14}$$

Solving for $MMCF_i$ we get

$$MMCF_i = \frac{mt_i(t)}{mt_z(t)} \times MMCF_z \tag{4.15}$$

The deviation of the microtick counters of clock i and reference clock z at time t is a measure for the drift rate of clock i against reference clock z.

The impact of calibration on free running clocks

As described in Section 4.1.2, the MMCF consists of an integer and a fractional part. Since any single macrotick is made up of an integer number of microticks, the fractional part is used to periodically prolong a macrotick by one microtick. A typical implementation of this mechanism will simply add up the fraction value at each macrotick until it overflows, at which point in time the current macrotick is expanded by one microtick ([TTT99]).
Be $t_i^{MT_k}$ the point in time when non-calibrated clock i generates macrotick k:

$$t_i^{MT_k} = \frac{k \times MMCF_z}{f_z \times (1 - \rho_i)} \tag{4.16}$$

Reference clock z generates macrotick k at

$$t_z^{MT_k} = \frac{k \times MMCF_z}{f_z} \tag{4.17}$$

It follows that

$$t_i^{MT_k} - t_z^{MT_k} = \frac{MMCF_z}{f_z} \times k \times (\frac{1}{1 - \rho_i} - 1) \tag{4.18}$$

Be $\mathcal{R}_{i,z}^{MT}$ the rate at which the macrotick generation times of clock i and reference clock z drift apart per macrotick of reference clock z. If follows from Equation 4.18 that

$$\mathcal{R}_{i,z}^{MT} = \frac{1}{1 - \rho_i} - 1 \tag{4.19}$$

Calibrating clock i according to Equation 4.13 we get

$$t_i^{MT_k} = \frac{k \times MMCF_i}{f_z \times (1 - \rho_i)} = \frac{k \times MMCF_z \times (1 - \rho_i)}{f_z \times (1 - \rho_i)} = t_z^{MT_k} \tag{4.20}$$

and therefore

$$\mathcal{R}_{i,z}^{MT} = 0 \tag{4.21}$$

$$\forall k : t_{MT_k}^i - t_{MT_k}^z = 0 \tag{4.22}$$

Formally, calibration of clock i against reference clock z leads to perfect agreement concerning the macrotick generation times: both clocks generate all macroticks k at exactly the same points in time. In theory, if clock i is calibrated against reference clock z, there is no need for periodical resynchronization of clock i to clock z because the two clocks are in perfect agreement at all times. In reality, periodical resynchronization of clock i to clock z is necessary for the following reasons:

- The drift rate of clock i against reference clock z cannot be measured with infinite precision (it would take infinite time to do so).
- The measured drift rate of clock i against reference clock z cannot be accounted for with infinite precision. Due to the binary representation of the fractional part of the MMCF, there will be digitalization errors.
- A prerequisite for our considerations is the assumption that the drift rate of clock i is constant during a given time interval T (see Equation 4.2). Clock i may drift apart from reference clock z if running free longer than T.

We therefore have to reconsider condition 4.22 for calibrated clock i:

$$\forall k : t_i^{MT_k} - t_z^{MT_k} = k \times \mathcal{R}_{i,z}^{MT}, \quad 0 < |\mathcal{R}_{i,z}^{MT}| \ll |\frac{1}{1-\rho_i} - 1| \tag{4.23}$$

Equation 4.23 shows that calibration brings clock i and reference clock z into much better agreement; however, clock i will eventually drift apart from reference clock z if never resynchronized to clock z due to the fact that calibration cannot account perfectly for drift rate ρ_i.

The impact of calibration on synchronized clocks

For the following considerations, we define the *local time* (see Section 2.4.2) at node i at time t as

$$LT_i(t) = MT_i(t) \times MMCF_z + mt_i(t) - mt_i(t_i^{MT_i(t)}) \tag{4.24}$$

The term $mt_i(t) - mt_i(t_i^{MT_i(t)})$ in Equation 4.24 equals the number of local microticks that has elapsed since the generation of the last macrotick at node i. Clock i is *synchronized* to clock z if clock i periodically brings its local time into agreement with the local time of clock z. Clock i synchronizes to clock z by periodically applying a *clock state correction term* $adj_i(t)$ to its local time:

$$adj_i(t) = LT_i(t) - LT_z(t) \tag{4.25}$$

Be k the number of nodes within a cluster. All nodes run local clocks with the same nominal frequency and therefore the same granularity.

Precision. We define the *precision* $\pi(t)$ of the cluster at time t to be the maximum deviation between any two node's local times at time t.

$$\forall i, 1 \leq i \leq k : \pi(t) = max|LT_i(t)) - min(LT_j(t)| \tag{4.26}$$

We define the *overall precision* Π of the cluster during time interval T to be the maximum precision observed within that time interval T:

$$\forall t, 0 \leq t \leq T : \Pi = max(\pi(t)) \tag{4.27}$$

Lemma: Precision between clock i and reference clock z improves if clock i is calibrated against reference clock z.

Proof: In the following, subset $_{cal}$ denotes calibrated clock i. Be $LT_{i_{cal}}(t)$ the local time of calibrated clock i. We have to show that

$$\forall t, 0 \leq t \leq T : |LT_{i_{cal}}(t) - LT_z(t)| < |LT_i(t) - LT_z(t)| \tag{4.28}$$

Calibration of clock i against reference clock z brings the macrotick counters of both clocks into agreement. It follows that

$$|MT_{i_{cal}}(t) - MT_z(t)| \leq |MT_i(t) - MT_z(t)| \tag{4.29}$$

If clock i is running slow against reference clock z, then the microtick counter at clock i proceeds slower than the microtick counter of reference clock z. It follows that

$$mt_i(t) - mt_i(t_i^{MT_i(t)}) < mt_z(t) - mt_z(t_z^{MT_z(t)}) \tag{4.30}$$

$$t_i^{MT_i(t)} > t_z^{MT_z(t)} \tag{4.31}$$

We know from Equation 4.20 that calibration of clock i against reference clock z brings the macrotick generation times into agreement. There is no impact on the microtick duration of clock i. It follows that

$$t_{i_{cal}}^{MT_{i_{cal}}(t)} < t_i^{MT_i(t)} \tag{4.32}$$

$$\Rightarrow mt_i(t) - mt_i(t_{i_{cal}}^{MT_i(t)}) > mt_i(t) - mt_i(t_i^{MT_i(t)}) \tag{4.33}$$

$$\Rightarrow |LT_{i_{cal}}(t) - LT_z(t)| < |LT_i(t) - LT_z(t)| \tag{4.34}$$

If clock i is running fast against reference clock z, then the microtick counter at clock i proceeds faster than the microtick counter of reference clock z. It follows that

$$mt_i(t) - mt_i(t_i^{MT_i(t)}) > mt_z(t) - mt_z(t_z^{MT_z(t)}) \tag{4.35}$$

$$t_i^{MT_i(t)} < t_z^{MT_z(t)} \tag{4.36}$$

After calibration of clock i we get

$$t_{i_{cal}}^{MT_{i_{cal}}(t)} > t_i^{MT_i(t)} \tag{4.37}$$

$$\Rightarrow mt_i(t) - mt_i(t_{i_{cal}}^{MT_i(t)}) < mt_i(t) - mt_i(t_i^{MT_i(t)}) \tag{4.38}$$

$$\Rightarrow |LT_{i_{cal}}(t) - LT_z(t)| < |LT_i(t) - LT_z(t)| \tag{4.39}$$

Figure 4.1 illustrates the principle of operation: at time t, precision is 10 microticks between clock z and clock i and 5 microticks between clock z and clock j. By bringing the macrotick generation times into agreement such that

$$t(MT_i) \approx t(MT_z) \approx t(MT_j) \tag{4.40}$$

precision improves to 3 microticks between clock z and clock i and to 2 microticks between clock z and clock j. Moreover, precision improves from 15 microticks to 5 microticks between clock i and clock j.

4.2 Cluster calibration against an external time reference

One possibility to improve cluster precision is to calibrate all nodes of a cluster against an external reference clock. Experimental results concerning the elimination of the impacts of clock drift by calibration of all nodes of a cluster against physical time using a GPS receiver can be found in [BP00b]. It is shown that the rate at which the local clocks drift apart from reference time remarkably decreases after calibration.The reduction of the rate at which the fastest and the slowest node drift apart usually leads to smaller clock state correction terms, if the nodes are synchronized, which improves cluster precision.

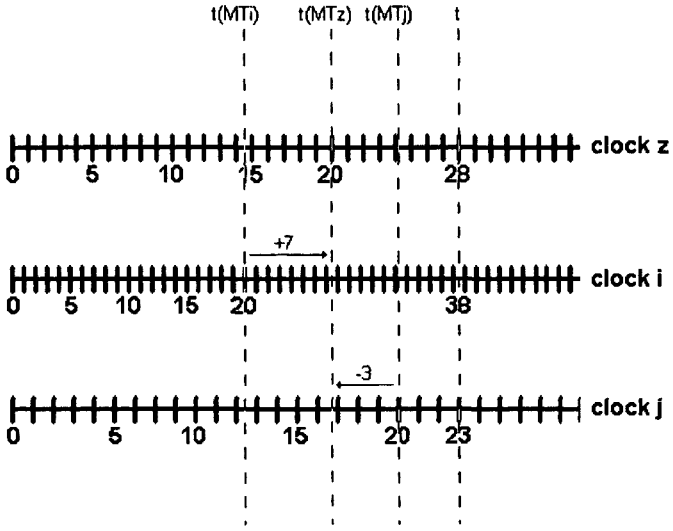

Figure 4.1: Calibration - principle of operation

4.3 Cluster calibration against global time

Another possibility to improve cluster precision is calibration of all nodes against global time. Global time is the internally synchronized cluster time - the common notion of time of all nodes within a cluster (see Section 2.6.3). As described in Section 2.5.1, all nodes read the local clocks of the other nodes and calculate a clock state correction term (in terms of local microticks) to bring their local clocks into better agreement with each other.

Be $adj_i(t_j)$ the clock state correction term that node i applies to its local clock at time t_j. Node i's view of global time at time t_j is equal to

$$GT_i(t_j) = LT_i(t_j) - adj_i(t_j) \tag{4.41}$$

Be $corr_i(t)$ the sum of all adjustments node i has applied to its local clock till time t:

$$corr_i(t) = \sum_{j=1}^{k} adj_i(t_j), \quad \forall t_j, 1 \leq j \leq k : t_j \leq t \tag{4.42}$$

Let $LT_i^{free}(t)$ be the local time at node i if no synchronization would have taken place. $LT_i^{free}(t)$ is equal to node's i local time at time t plus the sum of all correction terms that have been applied to clock i till time t:

$$LT_i^{free}(t) = LT_i(t) + corr_i(t) \tag{4.43}$$

Term $corr_i(t)$ is a measure for the drift rate of clock i against global time. We define $MT_i^{free}(t)$ and $MT_i(t)$, respectively, as

$$MT_i^{free}(t) = \frac{LT_i^{free}(t)}{MMCF_i} \tag{4.44}$$

$$MT_i(t) = \frac{LT_i(t)}{MMCF_z} \tag{4.45}$$

We want to calibrate clock i by bringing $MT_i^{free}(t)$ and $MT_i(t)$ into agreement:

$$\frac{LT_i^{free}(t)}{MMCF_i} = \frac{LT_i(t)}{MMCF_z} \tag{4.46}$$

Solving for $MMCF_i$, we get

$$MMCF_i = \frac{MMCF_z \times LT_i^{free}(t)}{LT_i(t)} = \frac{LT_i(t) + corr_i(t)}{LT_i(t)} \tag{4.47}$$

4.4 Cluster calibration against a rate master node

We extend the approach presented in Section 4.3 by adding the notion of a *rate master node*. A rate master node is a node within the cluster, preferably one with a quartz oscillator of high quality and therefore low drift rate. The rate master node never calibrates its local clock:

$$MMCF_{master} = const \tag{4.48}$$

All other nodes i calibrate their local clocks against global time and the rate master node's clock. These nodes are in the following referred to as *time keeping nodes*.

4.4.1 Calibration based on explicit timing information from the rate master node

The time keeping nodes calibrate their local clocks on the base of explicit timing information periodically provided by the rate master node by means of a message sent from the rate master node to the time keeping nodes.

Be $adj_{master}(t_j)$ the clock state correction term that the rate master node applies to its local clock at time t_j. The rate master node's view of global time at time t_j is equal to

$$GT_{master}(t_j) = LT_{master}(t_j) - adj_{master}(t_j) \tag{4.49}$$

Node i's view of global time at time t_j is equal to

$$GT_i(t_j) = LT_i(t_j) - adj_i(t_j) + adj_{master}(t_j) \tag{4.50}$$

Be $corr_{master}(t)$ the sum of all adjustments the rate master node has applied to its local clock till time t

$$corr_{master}(t) = \sum_{j=1}^{k} adj_{master}(t_j), \quad \forall t_j, 1 \leq j \leq k : t_j \leq t \tag{4.51}$$

and be $corr_i(t)$ the sum of all adjustments node i has applied to its local clock till time t:

$$corr_i(t) = \sum_{j=1}^{k} adj_i(t_j), \quad \forall t_j, 1 \leq j \leq k : t_j \leq t \tag{4.52}$$

At time t, the rate master node distributes the sum of its adjustments to the time keeping nodes. Let $LT_i^{free}(t)$ be the local time at node i if no synchronization would have taken place. $LT_i^{free}(t)$ is equal to node's i local time at time t plus the sum of all correction terms that have been applied to clock i till time t minus the sum of all correction terms the rate master node has applied to its local clock till time t:

$$LT_i^{free}(t) = LT_i(t) + corr_i(t) - corr_{master}(t) \tag{4.53}$$

Term $corr_i(t) - corr_{master}(t)$ is a measure for the drift rate of clock i against global time.

Applying $LT_i^{free}(t)$ to Equation 4.47, we get

$$MMCF_i = \frac{MMCF_z \times LT_i^{free}(t)}{LT_i(t)} = \frac{LT_i(t) + corr_i(t) - corr_{master}(t)}{LT_i(t)} \tag{4.54}$$

4.4.2 Calibration based on implicit timing information from the rate master node

The time keeping nodes calibrate their local clocks on the base of implicit timing information periodically provided by the rate master node by means of time difference capturing. Each time keeping node uses its estimated deviation from the rate master node provided by the time difference capturing mechanism (see Section 3.2.1) to calibrate its local clock.

Node i's view of global time at time t_j is equal to

$$GT_i(t_j) = LT_i(t_j) - adj_i(t_j) + delta_{master_i}(t_j) \tag{4.55}$$

Be $DELTA_i(t)$ the sum of all time difference capturing values from the rate master node that node i has observed till time t:

$$DELTA_i(t) = \sum_{j=1}^{k} delta_{master_i}(t_j), \quad \forall t_j, 1 \leq j \leq k : t_j \leq t \tag{4.56}$$

and be $corr_i(t)$ the sum of all adjustments node i has applied to its local clock till time t:

$$corr_i(t) = \sum_{j=1}^{k} adj_i(t_j), \quad \forall t_j, 1 \leq j \leq k : t_j \leq t \tag{4.57}$$

At time t, all time keeping nodes calibrate their local clocks. Let $LT_i^{free}(t)$ be the local time at node i if no synchronization would have taken place. $LT_i^{free}(t)$ is equal to node's i local time at time t plus the sum of all correction terms that have been applied to clock i till time t plus the sum of all time difference capturing values observed from the rate master node till time t:

$$LT_i^{free}(t) = LT_i(t) + corr_i(t) + DELTA_i(t) \tag{4.58}$$

Term $corr_i(t) + DELTA_i(t)$ is a measure for the drift rate of clock i against global time.

Applying $LT_i^{free}(t)$ to Equation 4.47, we get

$$MMCF_i = \frac{MMCF_z \times LT_i^{free}(t)}{LT_i(t)} = \frac{LT_i(t) + corr_i(t) + DELTA_i(t)}{LT_i(t)} \tag{4.59}$$

4.4.3 Stability considerations

Establishing a global timebase by a combination of a distributed clock state correction algorithm and a central clock rate correction algorithm as lined out in the last sections remarkably improves cluster precision as will be shown in the course of the simulation experiments in Section 4.5.

However, adding a central rate master node introduces a single point of failure into the system. In case of a failure of the rate master node [1], the central clock rate correction algorithm also fails. The time keeping nodes will continue to execute the distributed clock state correction algorithm and remain synchronized until reintegration of the rate master node upon which the central clock rate correction mechanism becomes available again.

If a system has to tolerate a permanent failure of a rate master node, a shadow rate master node has to be provided.

4.5 Experimental evaluation

The topic of this section is the experimental evaluation of the analysis of the last section. For the experiments, we use SIDERA, a simulation model for the Time-Triggered Architecture (see Chapter 3).

4.5.1 Objectives of the tests

The objectives of the tests presented in this section are

- Verification of the results provided by the formal analysis presented in the last sections
- Examination of the behavior of a calibrated cluster in the presence of clock drift faults
- Examination of the impacts on the cluster drift rate if a rate master node is deployed

4.5.2 Test cluster configuration

For the tests we use the configuration of a representative hardware cluster. The test cluster consists of four nodes. All nodes start with a nominal $MMCF_z$ of 20.00. Table 4.2 shows the configuration of the test cluster.

[1] We assume that the rate master node has fail-stop failure semantics (see Section 2.2.1).

Simulation time	0.01 seconds (= 400 slots)
$MMCF_0 = MMCF_1 = MMCF_2 = MMCF_3 = MMCF_z$	20.00
Drift rate node 0	$-4.27246094 \times 10^{-6}$
Drift rate node 1	$-1.89208984 \times 10^{-5}$
Drift rate node 2	$+5.34057617 \times 10^{-6}$
Drift rate node 3	$-2.59399414 \times 10^{-5}$

Table 4.2: Test cluster configuration

4.5.3 Tests setup and results

Test 1

We let the four nodes run free to determine the drift offsets [2] between the nodes during simulation time. Figure 4.2 shows the deviation of the local times (node times) from reference clock z. The x-axis denotes the progression of node time at reference clock z (which has a drift rate of 0). Nodes 0, 1 and 3 have negative drift rates (i.e. they are running fast against reference clock z), Node 2 has a positive drift rate (i.e. it is running slow). The maximum drift offset between node 3 (fastest) and node 2 (slowest) is about 626 microticks.

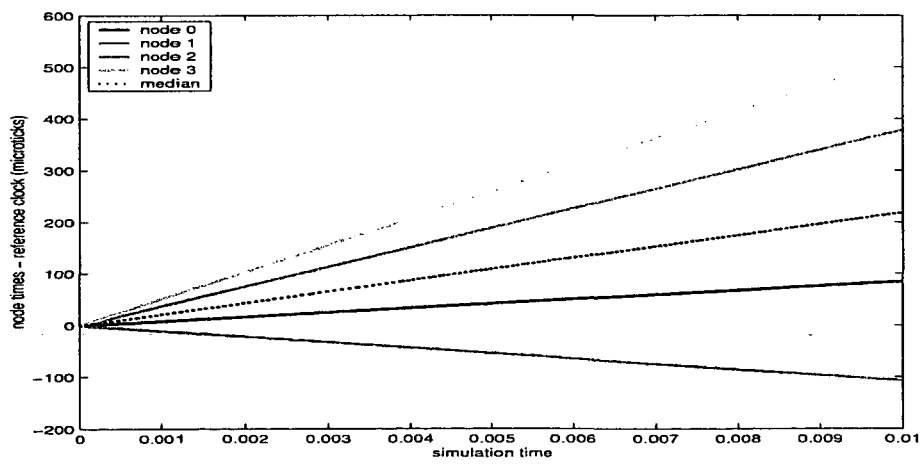

Figure 4.2: Test 1 - Free running nodes

[2] If two clocks with different clock drift rates are initialized and running free for a sufficient period of time, their microtick counters will eventually differ.

Test 2

We let the four nodes run free with calibrated MMCFs. The nodes are calibrated according to Equation 4.13. Table 4.3 shows the local clock parameters after calibration.

$MMCF_z$	20.00
$MMCF_0$	20.00854492188
$MMCF_1$	20.0378417968
$MMCF_2$	19.98931884766
$MMCF_3$	20.0518798828

Table 4.3: Test 2 - Calibrated cluster

Figure 4.3 shows the deviation of the node times from reference clock z. The maximum drift offset between node 3 (fastest) and node 2 (slowest) improves from 626 microticks to about 2 microticks for the calibrated nodes.

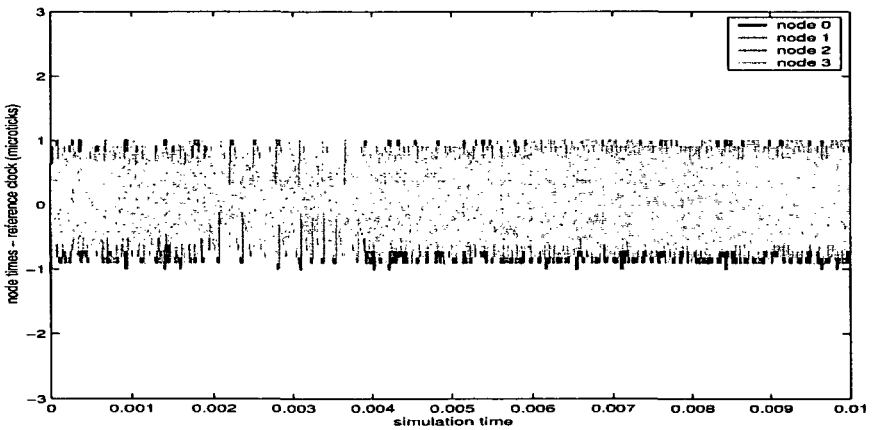

Figure 4.3: Test 2 - Free running nodes

Test 3

The test cluster is internally synchronized, the nodes are not calibrated (i.e. all nodes use the nominal $MMCF_z = 20.00$). Figure 4.4 shows the course of cluster precision during simulation time. The non-calibrated test cluster reaches a precision of 12 microticks. Figure 4.5 shows the drift of

synchronized cluster time against simulation time in terms of microticks. Cluster time drifts apart from simulation time with a rate of about 286 microticks/second, which is close to the median drift rate of the nodes (see Figure 4.2).

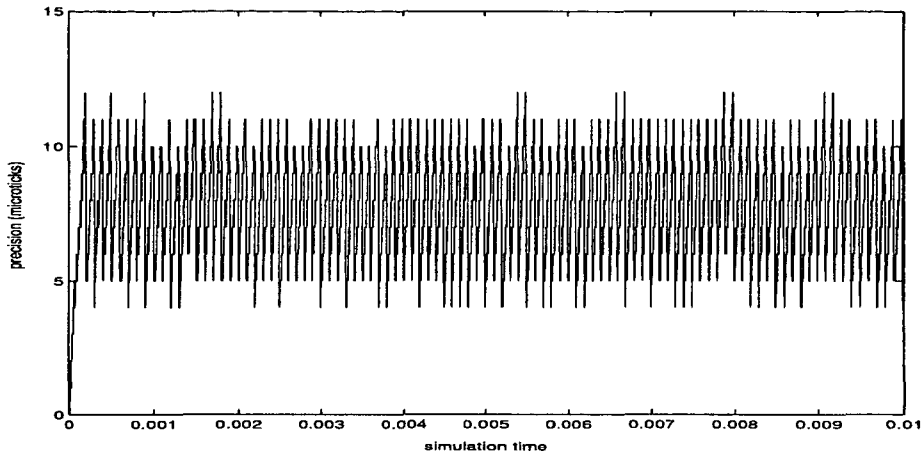

Figure 4.4: Test 3 - Cluster precision

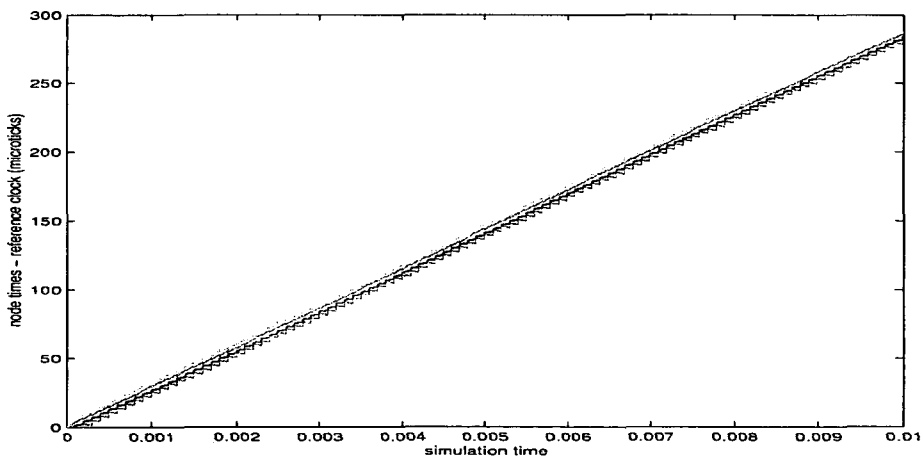

Figure 4.5: Test 3 - Cluster drift rate

Test 4

The test cluster is internally synchronized. All nodes start with a nominal $MMCF_z = 20.00$. After cluster startup, each node measures the deviation

of its local clock from global time for a duration of 3 TDMA rounds and thereafter calculates a new MMCF according to Equation 4.47. Table 4.4 shows the local clock parameters of the nodes after calibration against global time.

$MMCF_z$	20.00
$MMCF_0$	20.025
$MMCF_1$	20.010
$MMCF_2$	19.975
$MMCF_3$	19.955

Table 4.4: Test 4 - MMCFs after calibration against global time

Figure 4.6 shows the course of cluster precision during simulation time. Cluster precision remarkably improves from 12 microticks (see Figure 4.4) to 3 microticks after calibration of the nodes. Figure 4.7 shows that after calibration, the absolute values of the clock state correction terms remarkably decrease.

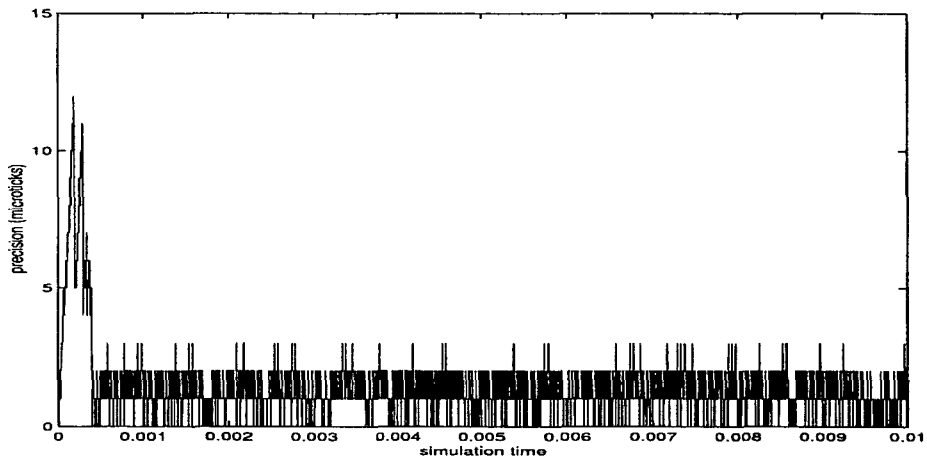

Figure 4.6: Test 4 - Cluster precision

Test 5

We drop the restriction that the drift rates of all nodes remain constant over simulation time (see Equation 4.2). Test 5 is based on Test 4. The only difference is that the drift rate of node 2 (which is the slowest node) changes

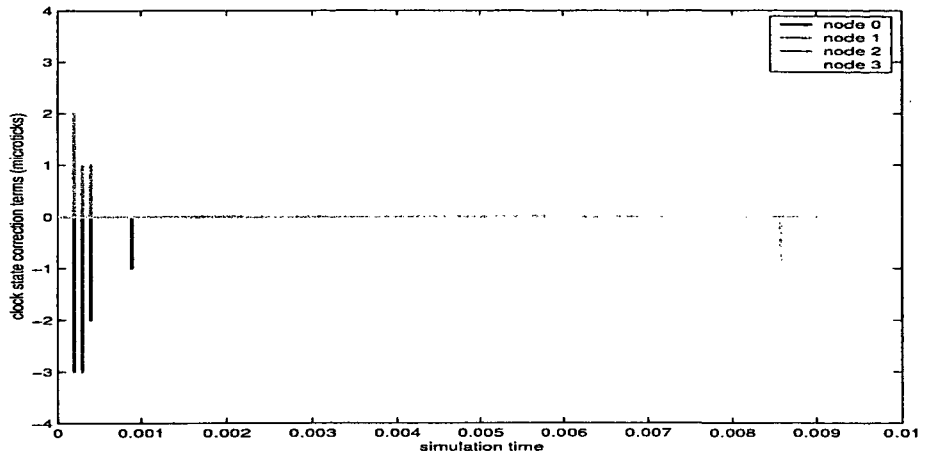

Figure 4.7: Test 4 - Clock state correction terms

with a rate of $+2,4 \times 10^{-6}$ seconds per TDMA round, starting in TDMA round 20 (i.e. the clock of node 2 becomes slower).

The *recalibration threshold* triggers the recalibration of a node against global time: if the clock state correction term of a node exceeds the recalibration threshold (i.e. the difference between this node's local time and global time is bigger than the recalibration threshold), the node recalibrates its local clock against global time. The recalibration threshold for all nodes is 8 microticks.

Figure 4.8 shows the course of cluster precision during simulation time. After cluster startup, all nodes calibrate their local clocks against global time for a duration of 10 TDMA rounds. The calibration phase ends at 0,0011 seconds simulation time from which point in time cluster precision improves from 12 microticks to 2 microticks. The clock of node 2 starts slowing down at 0,0023 seconds simulation time, causing cluster precision to deteriorate. At about 0,004 seconds node 2 detects that its clock state correction term exceeds the recalibration threshold and starts a recalibration phase of 10 TDMA rounds. Recalibration is complete at about 0,0044 seconds simulation time from which point in time cluster precision improves from 18 microticks to 3 microticks.

Figure 4.9 shows the drift of synchronized cluster time against simulation time in terms of microticks. It can be seen that node 2 drifts apart from cluster time between 0,0023 and 0,004 seconds simulation time and that node 2 gets into agreement with the other nodes after recalibration.

Figure 4.10 shows the correction terms the nodes have applied to their

local clocks during simulation time. It can be seen that the absolute values of the clock state correction terms of node 2 become bigger as node 2 drifts apart from global time between 0,0023 and 0,004 seconds simulation time. After recalibration, node 2 is in almost perfect agreement with the other nodes causing its clock state correction terms to become minimal.

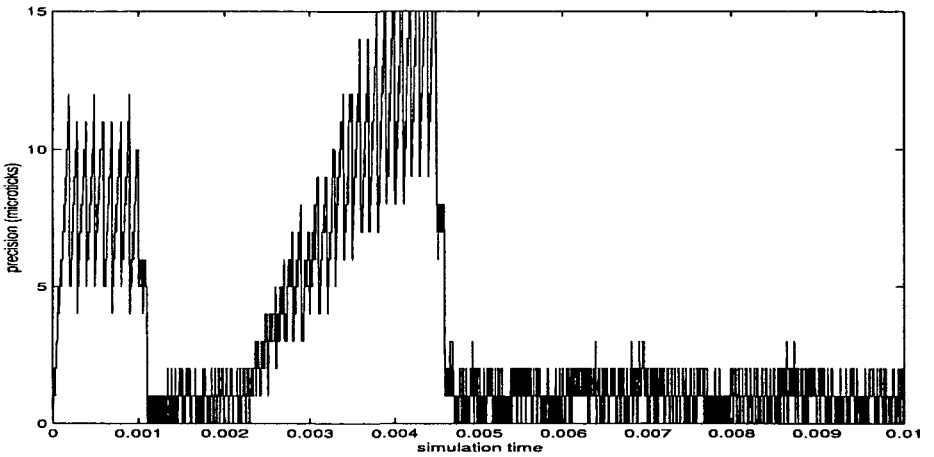

Figure 4.8: Test 5 - Cluster precision

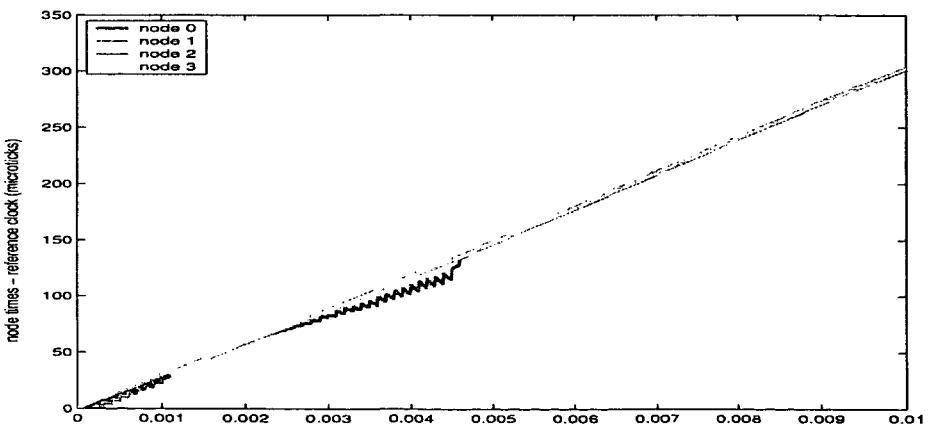

Figure 4.9: Test 5 - Cluster drift rate

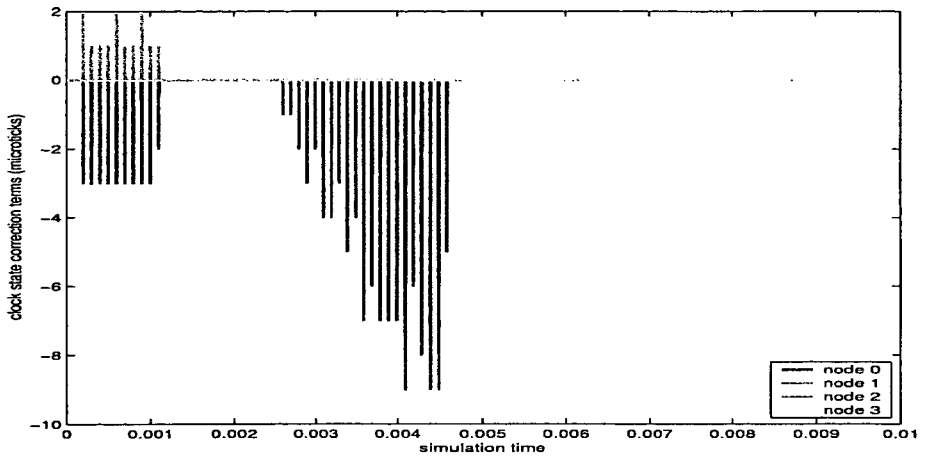

Figure 4.10: Test 5 - Clock state correction terms

Test 6

We have the test cluster internally synchronized and calibrated against global time and a rate master node. As described in Section 4.4 the rate master node periodically distributes the sum of its clock state correction terms $corr_{master}$ to the time keeping nodes which calibrate their local clocks on the base of $corr_{master}$ and the sum of their own clock state correction terms $corr_i$ according to Equation 4.54.

We perform four test runs with four different rate master nodes. Figure 4.11 shows the precision of the test cluster. After initial calibration, cluster precision does not exceed 3 microticks, independent of the choice of the rate master node. Figures 4.12 - 4.15 show the drift of the synchronized cluster times against simulation time in terms of microticks for the four different rate master nodes. If compared to Figure 4.2 it can be seen that in all four test cases the cluster drift rate is determined by the drift rate of the rate master node.

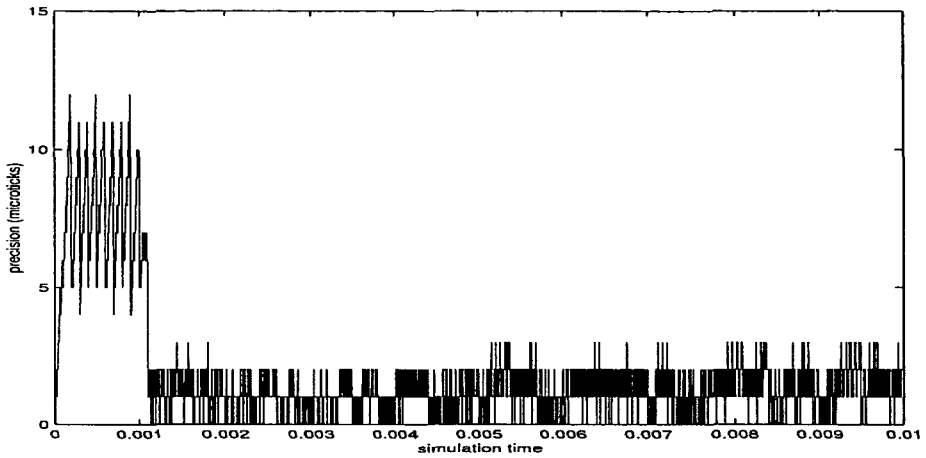

Figure 4.11: Test 6 - Cluster precision

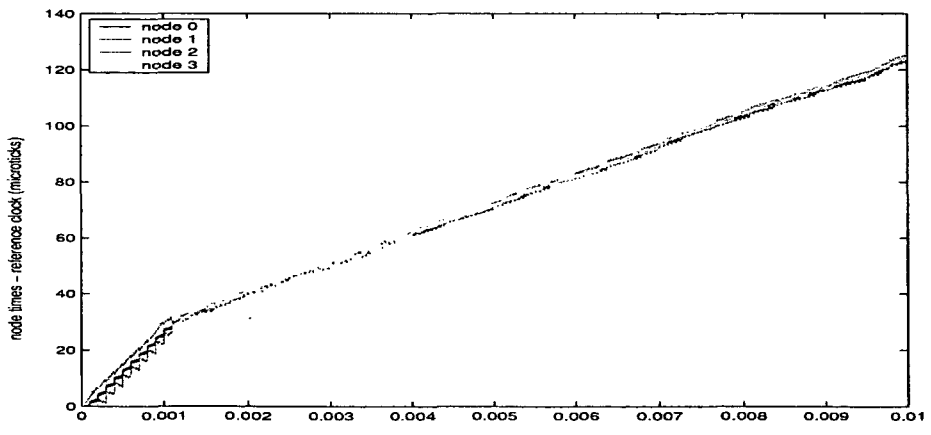

Figure 4.12: Test 6 - Cluster drift rate with rate master node 0

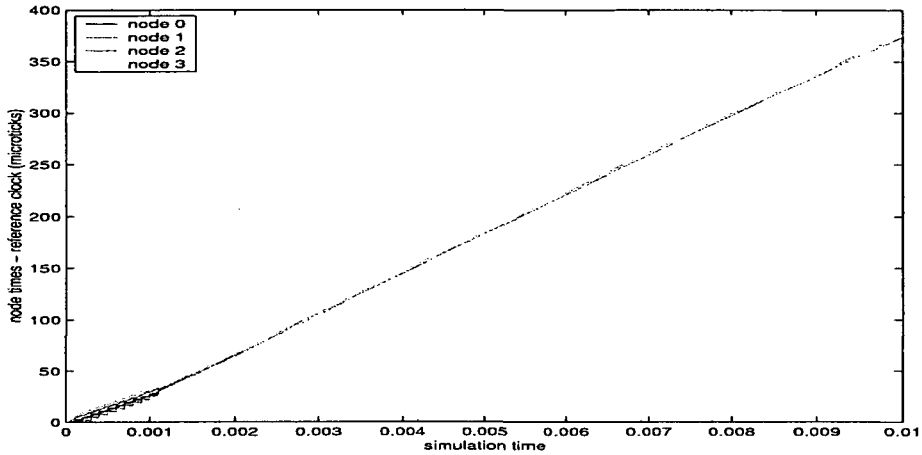

Figure 4.13: Test 6 - Cluster drift rate with rate master node 1

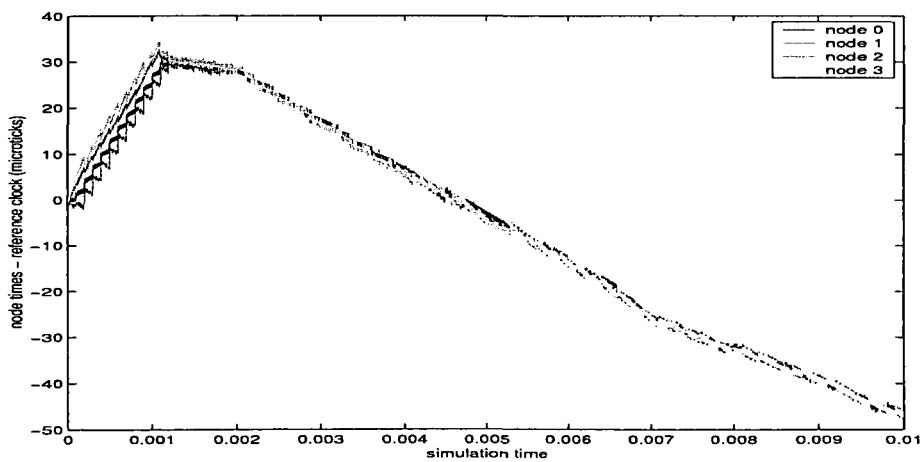

Figure 4.14: Test 6 - Cluster drift rate with rate master node 2

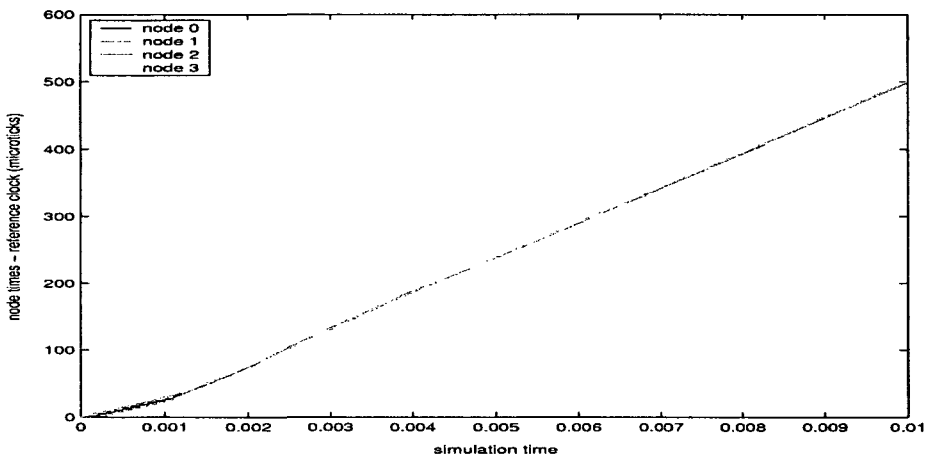

Figure 4.15: Test 6 - Cluster drift rate with rate master node 3

Chapter 5

Multi-cluster clock synchronization

This chapter is related to multi-cluster clock synchronization. We extend the concepts of dynamic clock calibration within a cluster introduced in Chapter 4 and provide a fault-tolerant clock synchronization algorithm for multi-cluster systems that makes use of this approach.

The second part of this chapter is devoted to the experimental evaluation of different multi-cluster architectures with regard to the achievable precision. We present both hierarchical configurations and configurations with feedback loops. Further, we compare the performance of our new clock synchronization algorithm combining clock state and clock rate correction with the performance of a multi-cluster clock synchronization algorithm based on clock state correction as described in Section 2.5.2.

5.1 A new approach

This section elaborates on the extension of the combination of clock state correction and clock rate correction for single-cluster systems presented in Chapter 4 to multi-cluster systems.

5.1.1 External clock synchronization

It is the objective of multi-cluster clock synchronization to establish a common time base in a multi-cluster system. As pointed out in Section 2.5.3, multi-cluster clock synchronization is a combination of internal and external

clock synchronization. Detailed analysis of the combination of clock state correction and clock rate correction to improve the quality of internal clock synchronization in single-cluster systems is given in Chapter 4. We now extend this approach to multi-cluster clock synchronization.

For this purpose we analyze the external synchronization between the clusters shown in Figure 5.1. This figure depicts a configuration of two clusters interconnected via a gateway (see Section 2.5.3). A gateway contains two communication controllers, one for each cluster it is connected to, and a host computer that can access the registers of both controllers and transfer data and control signals between them. We assume that in Figure 5.1 time flows from left to right, i.e. that the right cluster (cluster B) follows the time established in the left cluster (cluster A) and that the time is relayed to cluster B by the gateway. The right communication controller of the gateway (the time master node of cluster B, see Section 2.5.2) thus initializes cluster B and performs the rate master function for cluster B. In order to eliminate any feedback mechanism, the clock rate of the right communication controller is only controlled by the host computer of the left communication controller (which acts as a reference time server for cluster B, see Section 2.5.2) but not by the rate correction mechanism in the right cluster. However, the right communication controller participates as an equal member in the fault-tolerant clock state correction of cluster B.

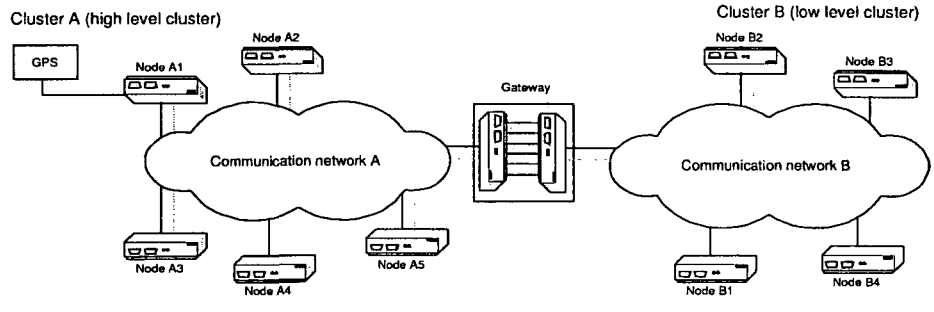

Figure 5.1: Multi-cluster clock synchronization

5.1.2 Combining clock state correction and clock rate correction

The reference time server in the gateway (left communication controller in Figure 5.1) controls the rate of the time master node in the gateway (right communication controller in Figure 5.1) in such a manner that

- over the long term, the rate of the time master node in the gateway is in agreement with the rate of the reference time server in the gateway. Since this rate is in agreement with the rate master of cluster A, both clusters progress at about the same clock rate.

- if the clock state in the time master node deviates from the clock state of the reference time server, then the host computer of the time master node performs a slight change of its rate such that an agreement between these two clock state values is attained within an upper bound in time. The value of this rate change must be below an a priori established threshold. This threshold establishes an upper bound for the clock rate error of cluster B.

5.1.3 Integrating internal and external clock synchronization

The time keeping nodes of the right cluster are not aware of this external synchronization procedure and do not contain any explicit mechanism for external clock synchronization. Such a mechanism is not needed because the time keeping nodes calibrate their local clocks on the base of implicit timing information from the rate master node as described in Section 4.4.2.

Instead of applying an external clock state correction term that would have explicitly to be communicated by the rate master node by means of a message, the time keeping nodes keep in track of the rate master node by using the time difference capturing values from the rate master node that they periodically calculate in course of execution of the internal clock synchronization algorithm (see Section 3.2.1). The internally synchronized cluster time in cluster B follows the clock state and clock rate of the reference time server in cluster A, which is the objective of external clock synchronization.

5.1.4 Blackout survivability

It is the goal of external clock synchronization to bring the internally synchronized cluster time into agreement with an external reference time. This is done by means of time master nodes which are connected to reference time servers and which provide timing information for all nodes within their clusters. The reference time server may be a time source (see Section 2.3.4) or a node in another cluster providing its cluster time as an external reference time.

However, the external reference time may not always be available. A loss of the external reference time signal is in the following referred to as a *blackout*. A blackout can be caused by

- **a failure of the reference time server**
- **a failure of the time master node**
- **a failure of the source providing the external reference time**
  A satellite based time source may be unavailable from time to time.
- **unavailability of the reference time signal due to environmental conditions**
  A satellite based time service may not be available e.g. in tunnels or underground car parks.

A blackout due to failures of one reference time server or of one time master node can be avoided by replication. However, nothing can be done against a failure of the time source or in case of unavailability of the reference time signal due to environmental conditions.

Therefore, a blackout is a scenario to reckon with which must be handled properly by the system under consideration. An externally synchronized system is only internally synchronized for the duration of the blackout and may arbitrarily drift apart from the external reference time. When the reference time signal becomes available again, the system has to be resynchronized to this reference time signal.

This resychronization process can become costly if the amount of system drift during the blackout phase is high. It may be necessary to initialize the local clocks within the system and to restart the synchronization process. During this initialization phase, the system service is not available.

With *blackout survivability* we denote the ability of a system to "survive" short external synchronization gaps without the necessity to initialize the whole system after the end of a blackout. In the following, we assume that the system under consideration consists of a single cluster externally synchronized to a receiver providing access to a time source (e.g. a GPS receiver).

In Section 4.5 we present experiments with free running clocks calibrated against a reference clock. It is shown that free running calibrated clocks drift apart from the reference clock with a much smaller rate than free running non-calibrated clocks. This result should also apply to a cluster which

is calibrated against GPS time and which looses external synchronization: the calibrated cluster time should drift apart from GPS time with a much smaller rate than a non-calibrated cluster. Consequently, the deviation between cluster time and GPS time should be remarkably smaller at the end of the blackout which in turn facilitates resynchronization of cluster time to GPS time.

We revisit the blackout survivability problem in Section 5.2.6 where we perform experiments with a calibrated and a non-calibrated cluster.

5.2 Experimental evaluation

The topic of this section is the experimental evaluation of multi-cluster clock synchronization algorithms by means of simulation runs using SIDERA (see Chapter 3).

We will perform experiments with systems consisting of two and three clusters. The experiments cover both hierarchical configurations and configurations with feedback loops.

5.2.1 Objectives of the tests

The objectives of the tests presented in this section are

- Analysis of the performance of our new algorithm for multi-cluster clock synchronization
- Comparison of the performance of our new algorithm to a multi-cluster clock synchronization algorithm based on clock state correction
- Examination of multi-cluster systems with feedback loops
- Examination of the blackout survivability of calibrated clusters

5.2.2 Structure of the tests

This section elaborates on the structure of the tests and detailly describes the internal structure of *calibrated* and *non-calibrated* clusters.

- **Multi-cluster clock synchronization based on clock state correction**

  - **Internal clock synchronization**
    The nodes execute a distributed clock synchronization algorithm in course of which they periodically determine an internal clock state correction term. Each node applies the internal clock state correction term to its local clock by means of state correction. The achievable precision (see Section 2.5.1) depends on the drift rates of the local clocks. In the course of the following experiments, such a cluster will be denoted as a **non-calibrated cluster**.
  - **External clock synchronization**
    The time master node (see Section 2.5.2) periodically determines the deviation of its local clock from the local clock of the reference time server (see Section 2.5.2) and communicates this deviation to the other nodes within its cluster by means of messages. The time master node and the other nodes determine an external clock state correction term which they apply to their local clocks by means of state correction. The achievable accuracy (see Section 2.5.2) depends on the precision of the externally synchronized non-calibrated cluster as well as on the drift rates of the reference time server and the time master node.

- **Multi-cluster clock synchronization based on clock state correction and clock rate correction**
  - **Internal clock synchronization**
    In each cluster there is at least one rate master node, the other nodes are time keeping nodes (see Section 4.4). The nodes execute a distributed clock synchronization algorithm in course of which they periodically determine an internal clock state correction term. Each node applies the internal clock state correction term to its local clock by means of state correction. Additionally, all time keeping nodes periodically adjust the rate of their local clocks on the base of implicit timing information from the rate master node as described in Section 4.4.2. The rate master node serves as a reference for the clock rate within its cluster. The achievable precision (see Section 2.5.1) depends on how good the drift rates of the local clocks can be brought into agreement (which cannot perfectly be done, see Equation 4.23). In the course of the following experiments, such a cluster will be denoted as a **calibrated cluster**.
  - **External clock synchronization**
    In an externally synchronized cluster, the time master node (see

Section 2.5.2) always serves as a rate master node (see Section 4.4). The time master node periodically determines the deviation of its local clock from the local clock of the reference time server (see Section 2.5.2) and slightly changes the rate of its local clock such that an agreement with the clock state of the reference time server will be attained within an upper bound in time. For the time keeping nodes follow the rate of the rate master node (see above), there is no need for any explicit external clock synchronization mechanism. The time keeping nodes are not aware of that they are externally synchronized (see Section 5.1.1). The achievable accuracy (see Section 2.5.2) depends on the precision of the externally synchronized calibrated cluster as well as on how good the drift rates of the local clocks of the reference time server and the time master node can be brought into agreement.

5.2.3 Test setup

For the experiments, we use cluster configurations from the model verification tests performed with SIDERA (see Section 3.4). Table 5.1 shows the simulation parameters common to all test runs. Table 5.2 shows the cluster-specific simulation parameters, the precision and the cluster drift rate that have been observed for the different clusters in the model verification tests.

Number of nodes	6
Microtick duration	50×10^{-9} seconds
Microticks per macrotick	20
Macrotick duration	10^{-6} seconds
Slot length	12 macroticks
Free running macroticks	0
TDMA round length	72 macroticks
Simulation time	200 slots

Table 5.1: Common parameters

5.2.4 Hierarchical configurations

We start with the investigation of hierarchical system configurations. Hierarchical means that the flow of timing information is unidirectional. A cluster that serves as an external time reference for another cluster is neither

Cluster	Δ_0	CS flags	SYF flags	Precision (microticks)	Drift rate $\times 10^{-5}$ s/s
1	0.0001	Slot 0, 4, 8, ...	Every slot	4	-5
2	0.0005	Slot 0, 4, 8, ...	Every slot	12	-6
3	0.0005	Slot 0, 6, 12, ...	Every slot	17	+8

Table 5.2: Cluster specific parameters

implicitly nor explicitly externally synchronized to the cluster time of that cluster.

Test 1

We connect Cluster 1 and Cluster 2 in a hierarchical manner such that Cluster 2 is externally synchronized to the cluster time of Cluster 1. The gateway consists of Node 3 in Cluster 1 and Node 3 in Cluster 2. Timing information flows from Cluster 1 to Cluster 2. Figure 5.2 shows the system configuration.

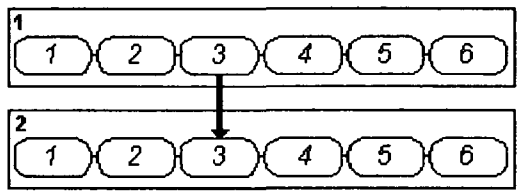

Figure 5.2: Test 1 - System configuration

Figure 5.3 shows the system precision in case that Cluster 1 and Cluster 2 are non-calibrated clusters. In this case, the system reaches a precision of 13 microticks.

Figure 5.4 shows the system precision in case that Cluster 1 and Cluster 2 are calibrated clusters. Compared to Figure 5.3, system precision improves from 13 microticks to 5 microticks.

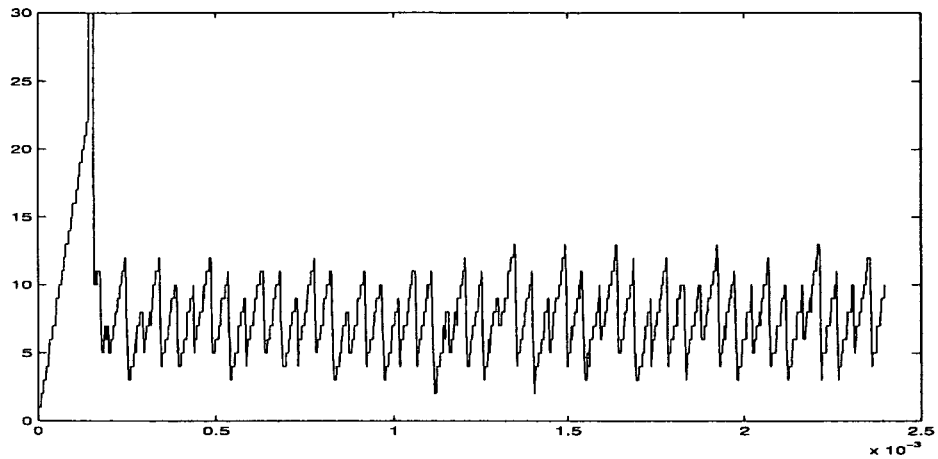

Figure 5.3: Test 1 system precision - non-calibrated clusters

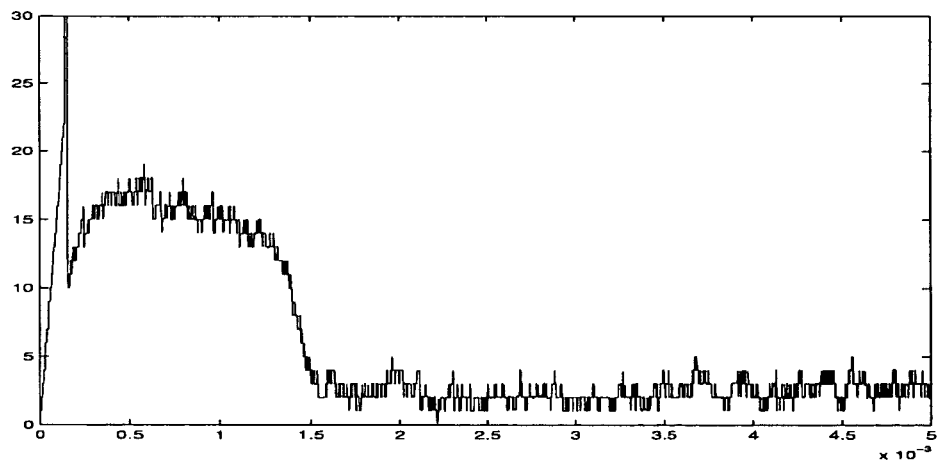

Figure 5.4: Test 1 system precision - calibrated clusters

Test 2

We add Cluster 3 to the configuration used for Test 1. Cluster 3 is externally synchronized to the cluster time of Cluster 2 which in turn is externally synchronized to the cluster time of Cluster 1. The gateway between Cluster2 and Cluster 3 consists of Node 3 in Cluster 2 and Node 3 in Cluster 3. Timing information flows from Cluster 1 to Cluster 2 and from Cluster 2 to Cluster 3. Figure 5.5 shows the system configuration.

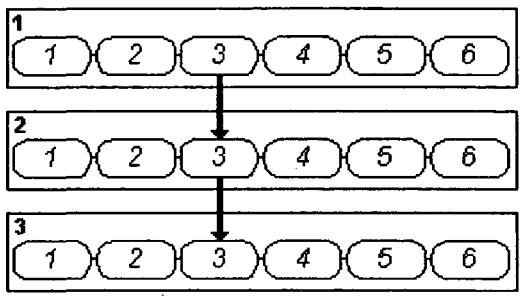

Figure 5.5: Test 2 - system configuration

Figure 5.6 shows the system precision in case that Cluster 1, Cluster 2 and Cluster 3 are non-calibrated clusters. In this case, the system reaches a precision of 17 microticks.

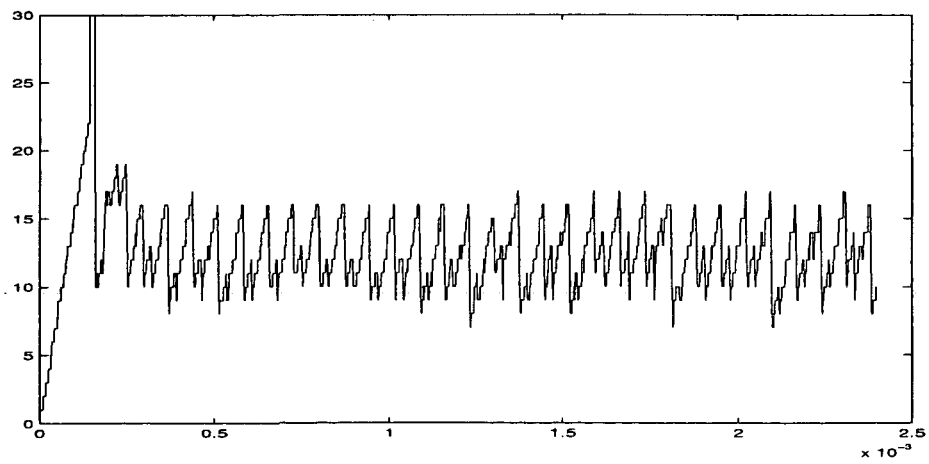

Figure 5.6: Test 2 system precision - non-calibrated clusters

Figure 5.7 shows the system precision in case that Cluster 1, Cluster 2 and

Cluster 3 are calibrated clusters. Compared to Figure 5.6, system precision improves from 17 microticks to 7 microticks.

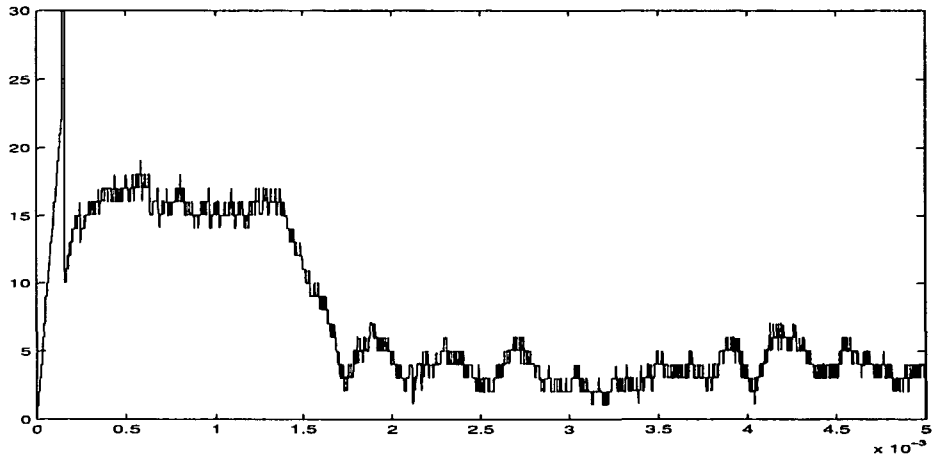

Figure 5.7: Test 2 system precision - calibrated clusters

5.2.5 Loopback configurations

We continue our investigations with loopback system configurations. Loopback means that the flow of timing information is not unidirectional anymore. A cluster that serves as an external time reference for another cluster may be implicitly or explicitly externally synchronized to the cluster time of that cluster. This is done by means of feedback loops.

If feedback loops are used, calibration of the clusters makes no sense because cluster calibration is done by means of a central rate correction algorithm based on the drift rate of a dedicated rate master node (see Section 4.4.3). The intention of using a central rate correction algorithm for a multi-cluster system is that the drift rate of the whole system shall follow the rate of a dedicated rate master node as described in Section 5.1.1 which is only meaningful in hierarchical configurations where the flow of timing information is unidirectional. We therefore did not evaluate the implications of feedback loops for multi-cluster systems with calibrated clusters.

Test 3

We connect Cluster 1 and Cluster 2 in a hierarchical manner such that Cluster 2 is externally synchronized to the cluster time of Cluster 1. Additionally, we feed back the timing information from Cluster 2 to Cluster 1 such that Cluster 1 is also externally synchronized to the cluster time of Cluster 2. The gateway consists of Node 3 in Cluster 1 and Node 3 in Cluster 2. Timing information flows from Cluster 1 to Cluster 2 and vice versa. Figure 5.8 shows the system configuration.

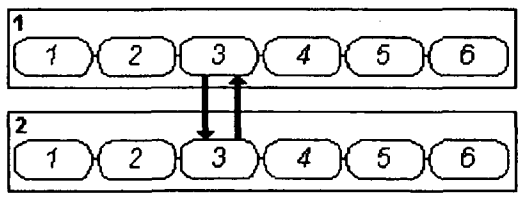

Figure 5.8: Test 3 - system configuration

Figure 5.9 shows that the system achieves a precision of 18 microticks.

Compared to the hierarchical configuration (see Figure 5.3), system precision deteriorates remarkably. Another remarkable characteristics of Figure 5.9 is that system precision oscillates between 18 and 5 microticks, a deviation which is remarkably larger than in Figure 5.3.

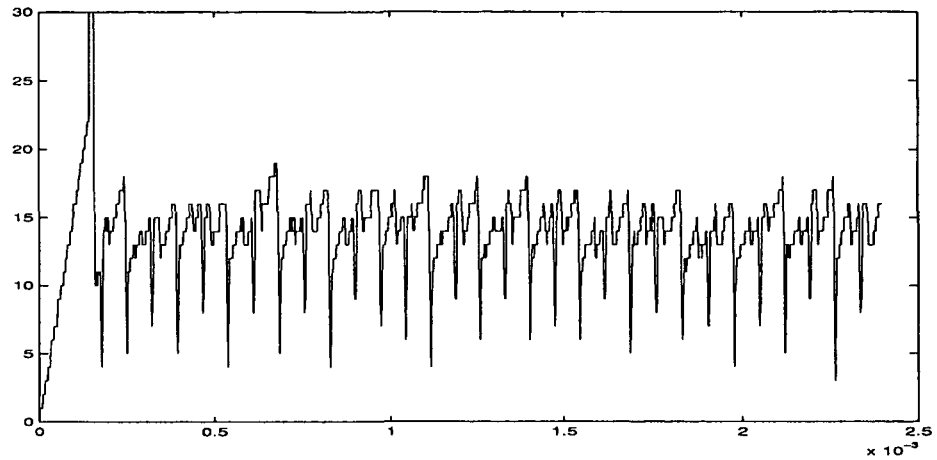

Figure 5.9: Test 3 system precision - external correction term not bounded

Figure 5.10 reveals the reason for this behavior. Due to the feedback loop, there is a mutual approximation of the cluster times of Cluster 1 and Cluster 2. Cluster 1 tries to follow the cluster time of Cluster 2 which in turn tries to approximate the cluster time of Cluster 1. Due to the mutual approximation of the cluster times the external clock state correction terms determined by the two clusters are too big, leading to overoscillation phenomena.

Figure 5.10 shows the course of the external clock state correction term of Cluster 1 over simulation time. It can be seen that the external clock state correction terms oscillate between positive and negative values due to the overcompensation of the external deviation to the cluster time of Cluster 2. The external clock state correction terms applied by Cluster 2 show the same behavior.

One obvious possibility to overcome the problem of overoscillation of the cluster times of Cluster 1 and Cluster 2 due to the mutual approximation of the cluster times is to bound the external clock state correction terms applied by the two clusters. All nodes in Cluster 1 and Cluster 2 divide their external clock state correction terms by 2 before application. This remarkably reduces the oscillation of the cluster times and improves precision as shown in Figure 5.11. The system reaches a precision of 14 microticks, which is almost the precision achieved with the hierarchical configuration (see Figure 5.3).

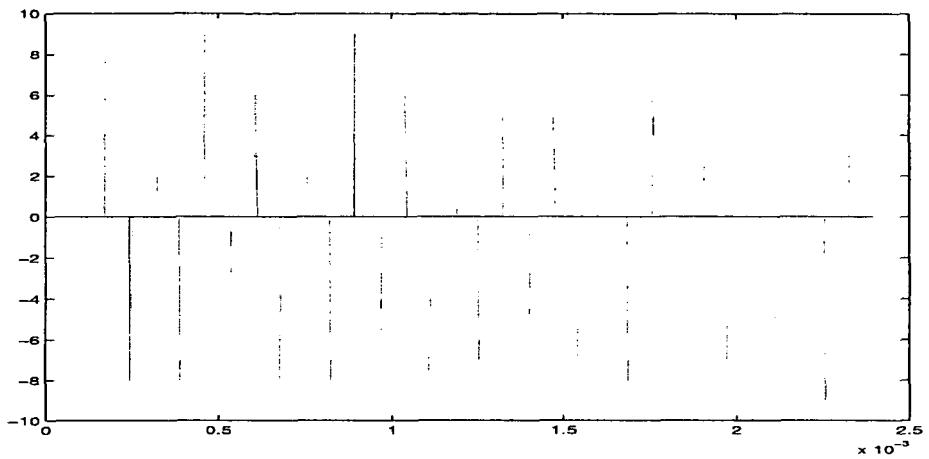

Figure 5.10: Test 3 external clock state correction terms of Cluster 1 - external correction terms not bounded

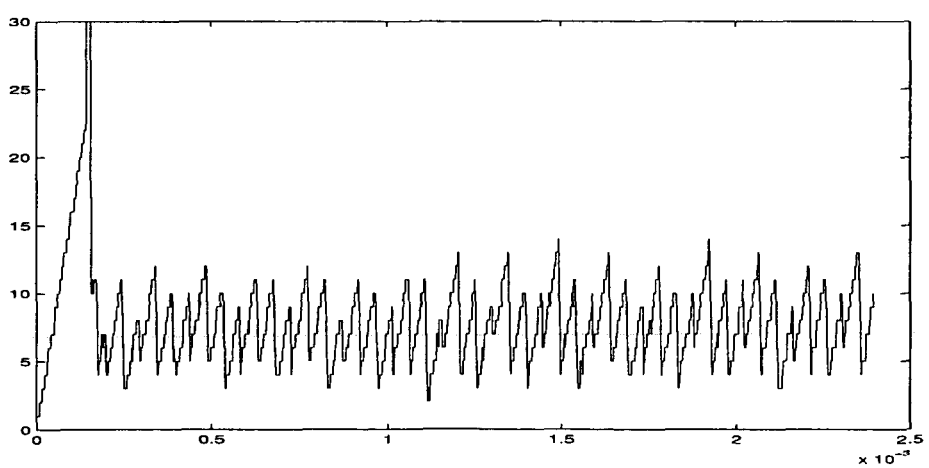

Figure 5.11: Test 3 system precision - external correction term bounded

Test 4

The system configuration is equal to that of Test 2 (see Section 5.2.4) with the exception that we added a feedback loop from Cluster 3 to Cluster 1. Timing information flows from Cluster 1 to Cluster 2, from Cluster 2 to Cluster 3 and from Cluster 3 back to Cluster 1. Figure 5.12 shows the system configuration.

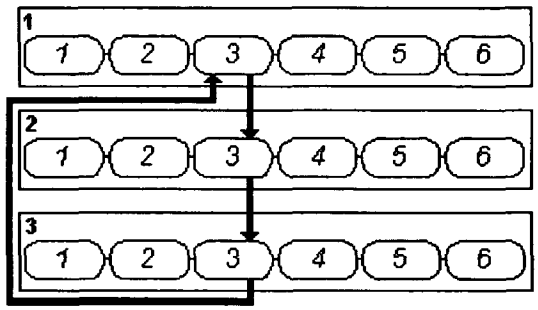

Figure 5.12: Test 4 - system configuration

Figure 5.13 shows that the system achieves a precision of 24 microticks.

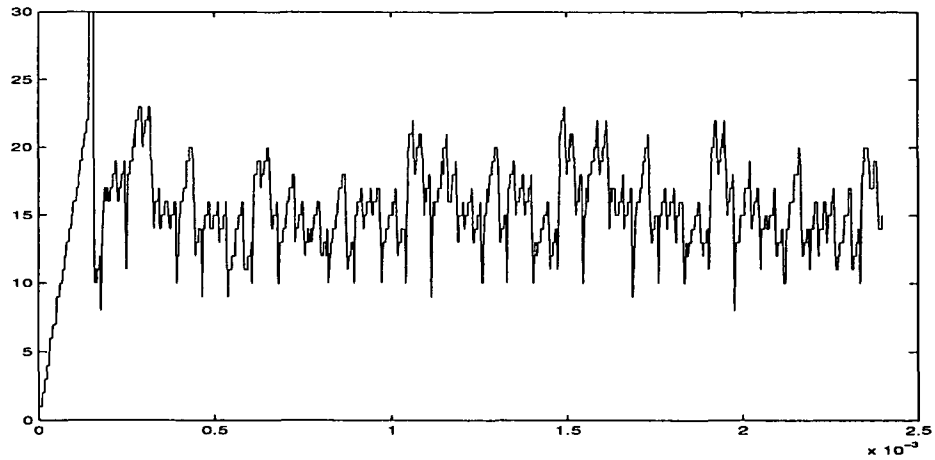

Figure 5.13: Test 4 system precision - external correction terms not bounded

If compared to the hierarchical configuration (see Figure 5.6), system precision deteriorates remarkably. Another remarkable characteristics in the course of system precision shown in Figure 5.13 is that system precision oscillates between 24 and 9 microticks, a deviation which is remarkably larger than in Figure 5.6. Again, the reason for this behavior is the feedback loop

between Cluster 3 and Cluster 1, leading to overcompensation of the cluster times in both clusters (see Section 5.2.5).

We levelled the oscillation of the cluster times of Cluster 1 and Cluster 3 by dividing the external clock state correction terms in both clusters by 2 before application. The result is shown in Figure 5.14. System precision improves from 24 microticks to 17 microticks, which is almost the precision achieved with the hierarchical configuration (see Figure 5.6).

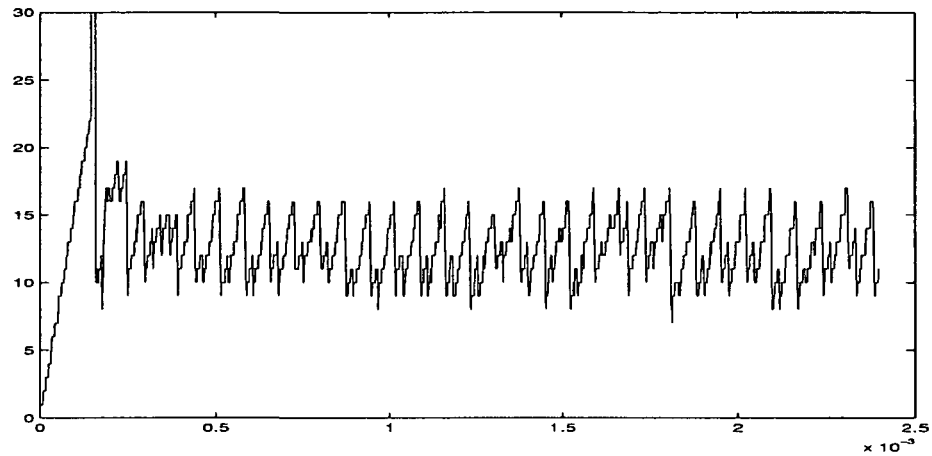

Figure 5.14: Test 4 system precision - external correction terms bounded

5.2.6 Blackout survivability

The topic of this section is the investigation of the blackout survivability (see Section 5.1.4) of a calibrated and a non-calibrated cluster.

For this purpose, we connect Cluster 3 to a simulated GPS receiver. The GPS receiver is simulated by a free running node with a very low drift rate (10^{-12} sec/sec). Node 3 in Cluster 3 is the time master node. Figure 5.15 shows the system configuration.

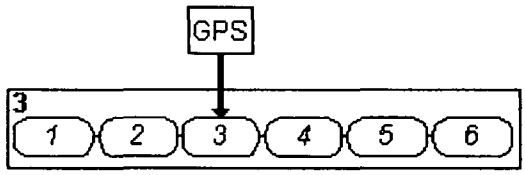

Figure 5.15: Blackout survivability - system configuration

Figure 5.16 shows the course of the system drift rate for non-calibrated Cluster 3 externally synchronized to the simulated GPS receiver. It can be seen that Cluster 3 follows the drift rate of the reference time signal until our GPS receiver dies due to a crash failure (see Section 3.3.1). The blackout phase is delimited by the two dotted vertical bars in Figure 5.16. The blackout phase starts in slot 200 (leftmost dotted vertical bar) for a duration of 120 slots (20 TDMA rounds). With the start of the blackout phase, Cluster 3 loses external synchronization and is only internally synchronized. Figure 5.16 shows that Cluster 3 drifts apart from reference time with a rate of about 8×10^{-5} sec/sec, which equals the cluster drift rate of Cluster 3 observed in the model verification tests (see Table 5.2). The deviation of the internally synchronized cluster time of Cluster 3 at the end of the blackout phase is about 50 microticks as shown in Figure 5.16. With the end of the blackout phase, our simulated GPS receiver resumes operation from which point in time Cluster 3 becomes externally synchronized again. The drift rate of Cluster 3 gets into agreement with the external time signal within a duration of 70 slots.

Figure 5.17 shows the course of the system drift rate for calibrated Cluster 3 externally synchronized to the simulated GPS receiver. At system start, Cluster 3 undergoes a calibration phase of 15 TDMA rounds during which the time master node determines its drift rate against the reference time signal (rising edge in Figure 5.17). At the end of the calibration phase, the time master node starts to approach the reference time signal with a rate of $\frac{1}{20}$

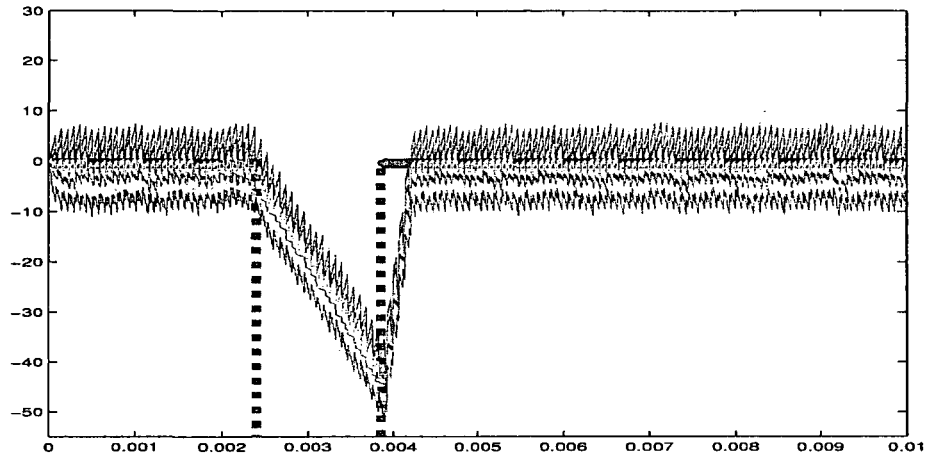

Figure 5.16: Blackout survivability - non-calibrated cluster

microtick per macrotick (falling edge in Figure 5.17) until it is in agreement with reference time. The time master node applies the drift rate estimated during the calibration phase to its local clock. From now on, the time master node follows the rate of the reference time signal by adjusting the rate of its local clock with a rate of $\frac{1}{20}$ microtick per macrotick. The time keeping nodes follow the rate of the time master node on the base of implicit timing information from the rate master node as described in Section 4.4.2. The blackout scenario is the same as in the last test with non-calibrated Cluster 3. The time master node detects the loss of the reference time signal with the start of the blackout phase and applies the rate determined during calibration phase to its local clock. Figure 5.17 shows that during the blackout the cluster drift rate of Cluster 3 does not remarkably deviate from reference time. This is due to the fact that Cluster 3 has been calibrated against the reference time signal before the blackout.

As argued in Section 5.1.4, calibration leads to a much smaller deviation from reference time at the end of the blackout phase. As a consequence, the resynchronization to the reference time signal after the blackout is no big deal compared to the scenario for non-calibrated Cluster 3 in Figure 5.16.

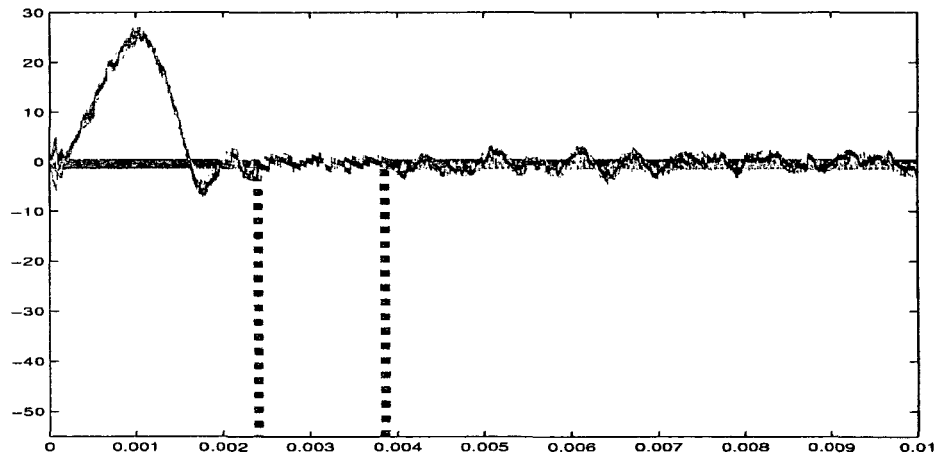

Figure 5.17: Blackout survivability - calibrated cluster

Chapter 6

Conclusion

This section summarizes the contents of this thesis and provides an outlook of the future work.

Contributions

This thesis made two major contributions.

First, SIDERA, a simulation model for time-triggered distributed systems was brought to life. SIDERA is based on the Time-Triggered Architecture and the Time-Triggered Protocol and allows the simulation of systems consisting of one or more clusters. The model was verified by means of verification tests against a VHDL implementation of a TTP controller in course of which it was shown that SIDERA follows the behavior of the VHDL reference model in all test cases.

Second, we used SIDERA for the investigation of mechanisms and means that improve the quality of synchronization in time-triggered distributed systems.

Achievements

Dynamic internal cluster calibration

We provided detailed analysis of dynamic cluster calibration mechanisms. In the course of our investigations, we identified the combination of a rate master node that determines the drift rate of its cluster and time keeping nodes that dynamically calibrate their local clocks on the base of implicit

timing information from the rate master node as the method of choice. Using this method it is possible to get a cluster precision in the order of 3 microticks, even if the local clocks are driven by low quality quartzes with drift rates in the order of 10^{-4} sec/sec.

Dynamic external cluster calibration

We have extended the approach of dynamic internal cluster calibration by integration of the tasks of a rate master and a time master into one node. This node dynamically adapts its rate according to an external reference time signal and dictates the clock rate of all time keeping nodes within its cluster.

Integration of internal and external clock synchronization

For timing information from the rate master node is obtained by the time keeping nodes by implicit means, no explicit mechanisms for external clock synchronization are needed. The time keeping nodes follow the rate of their rate master node, which in turn might follow the rate of an external reference time signal. There is no need for the distribution of an external clock state correction term which saves bandwidth and simplifies the synchronization mechanism.

Improved quality of synchronization

In the course of our investigations we have shown that a clock synchronization algorithm combining a central rate correction approach with a distributed state correction approach is ideally suited to improve the quality of synchronization in both single- and multi-cluster systems. In the multi-cluster configurations under analysis in course of simulation experiments, system precision improved by about 300 percent using a combined clock state and clock rate correction approach compared to an approach based on clock state correction only.

Improved blackout survivability

We have shown that calibrated clusters have a remarkably better blackout survivability than non-calibrated clusters. Clusters once calibrated against an external reference time signal drift apart from reference time with a much smaller rate during phases where the reference time signal is not available

than non-calibrated clusters. This facilitates resynchronization of such clusters at the end of the external synchronization gap.

Outlook

SIDERA

Due to performance and portability considerations, we plan to realize SIDERA as an object model according to the ANSI C/C++ standards. At the moment, SIDERA is a MATLAB/Simulink model using S-Functions written in C and the Real-Time Workshop RTW. The availability of an object model improves modularity and extendability of SIDERA. Furthermore, it facilitates the portation of SIDERA to different operation system platforms.

Quality of synchronization

Although we have observed remarkable performance improvements with regard to system precision, we think that further improvements are feasible. One important aspect concerns the rate calibration mechanism in the time master node. At the moment, this mechanism is based on simple rate control using a constant positive or negative rate correction term (according to the sign of the deviation of the time master node from the reference time signal). We think that we can attain a much better agreement between the time master node and the reference time signal by deploying advanced control engineering strategies.

Hardware experiments

Another important point is the verification of the results gained from the simulation experiments using hardware configurations. We have already verified dynamic internal cluster calibration by means of hardware experiments with a single cluster. However, the verification of the dynamic external cluster calibration mechanism is an important part of our future work.

Bibliography

[Ade03] A. Ademaj. Assessment of Error Detection Mechanisms of the Time-Triggered Architecture Using Fault Injection. PhD Thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2003.

[ALR01] A. Avizienis, J. Laprie, and B. Randell. Fundamental Concepts of Dependability. Research Report N01145, LAAS-CNRS, April 2001.

[AP98] E. Anceaume and I. Puaut. Performance Evaluation of Clock Synchronization Algorithms. Technical Report 3526, Institut de Recherche en Informatique et Systèmes Aléatoires, www.irisa.fr, October 1998.

[Ari91] Aristotle. Physica-Laterza, 1991.

[Aug] St. Augustine. Confessions, book XI, chapter XXX.

[Bau99] G. Bauer. Implementation and Evaluation of a Fault-Tolerant Clock Synchronization Algorithm for TTP/C. Master Thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1999.

[BHHN00] Boaz Barak, Shai Halevi, Amir Herzberg, and Dalit Naor. Clock synchronization with faults and recoveries (extended abstract). In *Symposium on Principles of Distributed Computing*, pages 133–142, 2000.

[BKP95] Günther Bauer, Hermann Kopetz, and Peter Puschner. Assumption Coverage under Different Failure Modes in the Time-Triggered Architecture. Research Report 14/1995, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1995.

[BKS03] Günther Bauer, Hermann Kopetz, and Wilfried Steiner. The Central Guardian Approach to Enforce Fault Isolation in the Time-Triggered Architecture. In *Proceedings of the Sixth International Symposium on Autonomous Decentralized Systems (ISADS 03)*, pages 37–44, Apr 2003.

[BP00a] G. Bauer and M. Paulitsch. An Investigation of Membership and Clique Avoidance in TTP/C. *19th IEEE Symposium on Reliable Distributed Systems, 16th - 18th October 2000, Nürnberg, Germany*, Oct. 2000.

[BP00b] G. Bauer and M. Paulitsch. External Clock Synchronization in the TTA. Research Report 3/2000, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2000.

[CAS94] Flaviu Cristian, Houtan Aghili, and Ray Strong. Clock Synchronization in the Presence of Omission and Performance Failures, and Processor Joins. In Zhonghua Yang and T. Anthony Marsland, editors, *Global States and Time in Distributed Systems, IEEE Computer Society Press.* 1994.

[CASD85] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. In *Proceedings of the Fifteenth International Symposium on Fault-Tolerant Computing*, pages 200–206, Ann Arbor, MI, June 1985.

[CF85] F. Cristian and C. Fetzer. Fault-tolerant External Clock Synchronization. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 70–77. IEEE, 1985.

[CF94] Flaviu Cristian and Christof Fetzer. Probabilistic Internal Clock Synchronization. In *Symposium on Reliable Distributed Systems*, pages 22–31, 1994.

[Cri91] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, 1991.

[dAB94] M. M. de Azevedo and D. M. Blough. Software-Based Fault-Tolerant Clock Synchronization for Distributed UNIX Environments. Technical Report ECE 94-03-01, Department of Electrical and Computer Engineering, University of California, Irvine, March 1994.

[Dan97] P.H. Dana. Global Posinitioning System (GPS) time dissemination for real-time applications. *Real-Time Systems*, 12:9–40, January 1997.

[DHS84] Danny Dolev, Joe Halpern, and H. Raymond Strong. On the Possibility and Impossibility of Achieving Clock Synchronization. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 133–143, 1984.

[Dij74] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[DW95] S. Dolev and J. L. Welch. Self-stabilizing clock synchronization in the presence of byzantine faults. In *Proceedings of the Second Workshop on Self-Stabilizing Systems*, 1995.

[Ein55] A. Einstein. *The Meaning of Relativity.* Princeton Univ. Press, Princeton, 1955.

[FC95] Christof Fetzer and Flaviu Cristian. An Optimal Internal Clock Synchronization Algorithm. In *Compass '95: 10th Annual Conference on Computer Assurance*, pages 187–196, Gaithersburg, Maryland, 1995. National Institute of Standards and Technology.

[FC97] Christof Fetzer and Flaviu Cristian. Integrating External and Internal Clock Synchronization. *Real-Time Systems*, 12(2):123–171, 1997.

[Gal99] Thomas M. Galla. Cluster Simulation in Time-Triggered Real-Time Systems. PhD Thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1999.

[GP99] T. Galla and R. Pallierer. Cluster simulation-support for distributed development of hard real-time systems using TDMA-based communication. In *Proceedings of the 11 th Euromicro Conference on Real-Time Systems*, pages 150 – 157, 1999.

[HD93] Kenneth Hoyme and Kevin Driscoll. SAFEbus. *IEEE Aerospace and Electronics Systems Magazine*, 8(3):34–39, March 1993.

[HS95] Matti A. Hiltunen and Richard D. Schlichting. Properties of Membership Services. In *Second International Symposium on Autonomous Decentralized Systems (ISADS'95)*, Phoenix, Arizona, USA, April 1995.

[HSSD84] J.Y. Halpern, B. Simons, R. Strong, and D. Dolev. Fault-tolerant Clock Synchronization. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, pages 89–102, 1984.

[Jon98] Erland Jonsson. An Integrated Framework for Security and Dependability. In *Proceedings of the 1998 workshop on New security paradigms*, pages 22–29, Charlottesville, Virginia, United States, 1998.

[KAH04] Hermann Kopetz, Astrit Ademaj, and Alexander Hanzlik. Integration of Internal and External Clock Synchronization by Combination of the Clock-State and Clock-Rate Correction in Fault-Tolerant Distributed Systems. Research Report, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2004.

[KB03] Hermann Kopetz and Günther Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1):112 - 126, January 2003.

[KBP01] Hermann Kopetz, Gnther Bauer, and Stefan Poledna. Tolerating Arbitrary Node Failures in the Time-Triggered Architecture. *SAE 2001 World Congress, March 2001, Detroit, MI, USA*, Mar. 2001.

[KKMS95] H. Kopetz, A. Krüger, D. Millinger, and A. Schedl. A Synchronization Strategy for a Time-Triggered Multicluster Real-Time System. In *14th Symposium on Reliable Distributed Systems*, Bad Neuenahr, Germany, September 1995.

[KO87] H. Kopetz and W. Ochsenreiter. Clock Synchronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, C-36(8), August 1987.

[KO02] H. Kopetz and R. Obermaisser. Temporal composability. *IEE Computing and Control Engineering Journal*, 13, August 2002.

[Kop92] Hermann Kopetz. Sparse Time versus Dense Time in Distributed Real-Time Systems. In *Proceedings of the 19th IEEE Systems Symposium (RTSS98), December 1998*, pages 460–467, Saitama, Japan, October 1992.

[Kop97] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, 1997.

[Kop98] Hermann Kopetz. The Time-Triggered Model of Computation. *Proceedings of the 19th IEEE Systems Symposium (RTSS98), December 1998*, Dec. 1998.

[Lap92] J. C. Laprie. *Dependability: Basic Concepts and Terminology.* Springer-Verlag, Vienna, 1992.

[Lap95] J.-C. Laprie. Dependability – its attributes, impairments, and means. In *B. Randell, J.-C. Laprie, H. Kopetz, and B. Littlewood, editors, Predictably Dependable Computing Systems*, pages 3–24, Heidelberg, Germany, 1995. Springer-Verlag.

[Leb98] M. Lebedev. GLONASS as instrument for precise UTC transfer. In *Proceedings of the 12 th European Frequency and Time Forum*, Warsaw, Poland, March 1998.

[Lic97] R. Lichtenecker. Terrestrial time signal dissemination. *Real-Time Systems*, 12:41–61, January 1997.

[LL84] 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. ACM, 1984.

[LM94] M. C. Little and D. L. McCue. Construction and Use of a Simulation Package in C++. *C User's Journal*, 12(3), 1994.

[LMS85] L. Lamport and P. M. Melliar-Smith. Synchronizing Clocks in the Presence of Faults. *Journal of the ACM*, 32(1):52–78, 1985.

[Lön99] Henrik Lönn. Initial Synchronization of TDMA Communication in Distributed Real-Time Systems. In *19th IEEE International Conference on Distributed Computing Systems*, Austin, Texas, USA, May 1999.

[LSP82] Lamport, Shostak, and Pease. The Byzantine Generals Problem. In *Advances in Ultra-Dependable Distributed Systems, N. Suri, C. J. Walter, and M. M. Hugue (Eds.), IEEE Computer Society Press.* 1982.

[Mil94] David L. Mills. Internet Time Synchronization: The Network Time Protocol. In *Zhonghua Yang and T. Anthony Marsland (Eds.), Global States and Time in Distributed Systems, IEEE Computer Society Press.* 1994.

[Min04] Paul S. Miner. SPIDER Formal Models – Where are we now? In *Internal Formal Methods Workshop, NASA Langley Research Center*, Hampton, VA, Feb 2004.

[MS85] Stephen R. Mahaney and Fred B. Schneider. Inexact Agreement: Accuracy, Precision and Graceful Degradation. In *4th ACM Symposium on Principles of Distributed Computing*, pages 237–249, Minaki, Canada, Aug 1985.

[MS92] S. Mishra and R. Schlichting. Abstractions for Constructing Dependable Distributed Systems. Technical Report TR 92 -12, 1992.

[New87] I. Newton. Mathematical Principles of Natural Philosophy. In *volume 34 of Great Books*, Chicago, Illinois, 1687. Encyclopaedia Britannica, Inc., 1687.

[Pal00] R. Pallierer. Validation of Distributed Algorithms in Time-Triggered Systems by Simulation. PhD Thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2000.

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

[Pet02] Philipp Peti. The Concepts behind Time, State, Component, and Interface: A Literature Survey. Research Report, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2002.

[Pfe00] Holger Pfeifer. Formal Verification of the TTP Group Membership Algorithm. In *FORTE/PSTV 2000*, Pisa, Italy, October 2000.

[PG98] Roman Pallierer and Thomas Galla. TTPSIM: A Versatile Simulation Environment for TTP/C. Research Report 26, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1998.

[Pol94] S. Poledna. Replica Determinism in Fault-Tolerant Real-Time Systems. PhD Thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1994.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, 1980.

[RSB90] P. Ramanathan, K.G. Shin, and R.W. Butler. Fault-tolerant Clock Synchronization in Distributed Systems. *IEEE Computer*, 23(10):33–42, October 1990.

[Rus36] B. Russell. *Proc. Camb. Philos. Soc.*, 32:216–228, 1936.

[SC90] Frank Schmuck and Flaviu Cristian. Continuous Clock Amortization need not affect the Precision of a Clock Synchronization Algorithm. In *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pages 504–511, Quebec City, Quebec, Canada, 1990.

[Sch86] Fred B. Schneider. A Paradigm for Reliable Clock Synchronization. Technical report, Department of Computer Science, Cornell University, Ithaca, New York 14853, April 1986.

[Sch88] W. Schwabl. Der Einfluss zufälliger und systematischer Fehler auf die Uhrensynchronisation in verteilten Echtzeitsystemen. PhD Thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1988.

[Sch94a] A. Schedl. Introduction to the ClockSync Project. Research Report 19, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1994.

[Sch94b] A. Schedl. Program Documentation of the ClockSync Project. Research Report 20, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1994.

[Sch94c] F. A. Schreiber. Is Time a Real Time? An Overview of Time Ontology in Informatics. In W. A. Halang and A. D. Stoyenko, editors, *Real Time Computing*, pages 283–307. Springer, Berlin, Heidelberg, 1994.

[Sch95] A. V. Schedl. The Simulation of Multicluster Clock Synchronization Strategies. Research Report 22, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1995.

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

[SP01] Wilfried Steiner and Michael Paulitsch. The Transition from Asynchronous to Synchronous System Operation: An Approach for Distributed Fault-Tolerant Systems (Including Simulation). Research Report 26/2001, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2001.

[SPHH04] Wilfried Steiner, Michael Paulitsch, Brendan Hall, and Alexander Hanzlik. Structuring of Time-Triggered Architecture (TTA) Systems and Initial Synchronization. Research Report, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2004.

[SRSP04] Wilfried Steiner, John Rushby, Maria Sorea, and Holger Pfeifer. Model checking a fault-tolerant startup algorithm: From design exploration to exhaustive fault simulation. *The International Conference on Dependable Systems and Networks (DSN 2004)*, Jun. 2004.

[SS83] R. Schlichting and F. Schneider. Fail-stop processors: An approach to designing fault tolerant computing systems. *ACM Transactions on Computer Systems*, 1(3):222–238, 1983.

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

[SW99] Klaus Schossmaier and Bettina Weiss. An Algorithm for Fault-Tolerant Clock State and Rate Synchronization. In *Symposium on Reliable Distributed Systems*, pages 36–47, 1999.

[SWGS99] Ulrich Schmid, Bettina Weiss, Günther Gridling, and Klaus Schossmaier. A Unified Approach for Simulation and Experimental Evaluation of Fault-Tolerant Distributed Systems. In *Proceedings of the IASTED International Conference on Applied Modelling and Simulation*, 1999.

[Tem99] C. Temple. Enforcing Error Containment in Distributed Time-Triggered Systems: The Bus Guardian Approach. PhD Thesis,

Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1999.

[TTT99] TTTech. Specification of the TTP/C Protocol. Specification, TTTech Computertechnik AG, Schönbrunner Strasse 7, 1040 Vienna, Austria, 1999.

[WGSS99] Bettina Weiss, Günther Gridling, Ulrich Schmid, and Klaus Schossmaier. The SimUTC Fault-Tolerant Distributed Systems Simulation Toolkit. Technical report, Department of Automation, TU Vienna, April 1999.

[Whi90] G. J. Whitrow. *The Natural Philosophy of Time.* Oxford Univeristy Press, second edition, 1990.

[Wie14] N. Wiener. A contribution to the theory of relative position. *Camb. Philos. Soc.*, 17:441–449, 1914.

Appendix A

Configuration File Structure

This section shows the structure of the configuration file *model_param.dat*. For correct interpretation of the configuration file it is necessary that all entries are in the order shown. The column *exists* in tables A.2 and A.3 describes under which conditions an entry *must be defined.* Mandatory means that the entry has to exist on all conditions. Optional means that the entry is not mandatory. Any other entry in this column denotes the condition on which the entry becomes mandatory. These rules only ensure *syntactical correctness* of the configuration file. The default value for all entries is 0.

File section	quantity
System specific parameters	1
Cluster specific parameters	*num_clusters*

Table A.1: File structure

entry	exists	quantity
sim_time	M	1
num_clusters	M	1
ext_corr_rate_max	*num_clusters* > 1	1
sample_rate_mt_log	O	1
sample_rate_ict_log	O	1
sample_rate_ect_log	O	1
sample_rate_MT_log	O	1
sample_rate_user_log	O	1
num_cluster_map_entries	O	1
master *slave*	*num_cluster_map_entries* > 0	*num_cluster_map_entries*

Table A.2: System specific parameters

entry	exists	quantity
num_nodes	M	1
slot_len_ref	M	1
slot_len_sim	M	1
delta_0	O	1
symm_capturing	O	1
mt_actiontime	O	1
recover_from_freeze	O	1
self_calibration	O	1
use_master_clock	O	1
master_clock_node	O	1
recalibration_threshold	O	1
central_guardian	O	1
guardian_node	O	1
div_ext_corr_term	O	1
time_flooding	O	1
no_cluster_startup	O	1
no_sync_nodes	O	1
signal_speed	O	1
prop_delay_corr	O	1
spread_corr_int	O	1
spread_corr_ext	O	1
num_slots	M	1
LogicalSenderName	M	*num_slots*
CS	O	
SYF	O	
LCP	M	*num_nodes*
OSC	M	
gateway	O	
time_master_node	O	
coldstart_flag	O	
max_coldstart_frames	O	
sys_drift	*delta_0* $= 0$	
freeze_at_slot		
freeze_duration	*freeze_at_slot* $\neq 0$	
freeze_num	*freeze_at_slot* $\neq 0$	
freeze_repeat_rate	*freeze_at_slot* $\neq 0$	
faulty_msg_in_round		
faulty_msg_num	*faulty_msg_in_round* $\neq 0$	
faulty_msg_repeat_rate	*faulty_msg_in_round* $\neq 0$	
change_drift_at_slot		
change_drift_factor	*change_drift_at_slot* $\neq 0$	
change_drift_num	*change_drift_at_slot* $\neq 0$	
change_drift_repeat_rate	*change_drift_at_slot* $\neq 0$	
distance_to_medium	O	

Table A.3: Cluster specific parameters

Appendix B

Simulation Log File structure

File section	offset (bytes)
Header	0
Data	sizeof (Header)

Table B.1: File structure

field	offset (bytes)	content
byte_order	0	0 = LITTLE ENDIAN 1000 = BIG ENDIAN
sample_size	4	number of rows per sample = $(num\_clusters \times 8) + 1$
num_samples	8	number of samples in file
reserved	12	0
data_name_len	16	number of characters in *data_name*
data_name	20	data name

Table B.2: Header structure

field	offset (bytes)
Sample 0	0
Sample 1	$sample\_size \times 8$
.....	.....
Sample $num\_samples - 1$	$(sample\_size \times 8) \times (num\_samples - 1)$

Table B.3: Data structure

$num\_clusters$	field	offset (bytes)	content
	sim time	0	simulation time at sample generation
1	data [0 0]	8	data cluster 0 node 0
	data [0 1]	16	data cluster 0 node 1
	.....	.....	.....
	data [0 7]	64	data cluster 0 node 7
2	data [0 0]	72	data cluster 0 node 0
	data [0 1]	80	data cluster 0 node 1
	.....	.....	.....
	data [0 7]	128	data cluster 0 node 7
.....	.....	.....	.....
n	data [n-1 0]	$64 \times (n-1) + 8$	data cluster n-1 node 0
	data [n-1 1]	$64 \times (n-1) + 16$	data cluster n-1 node 1
	.....	.....	.....
	data [n-1 7]	$64 \times (n-1) + 64$	data cluster n-1 node 7

Table B.4: Sample structure

Appendix C

Reference tests

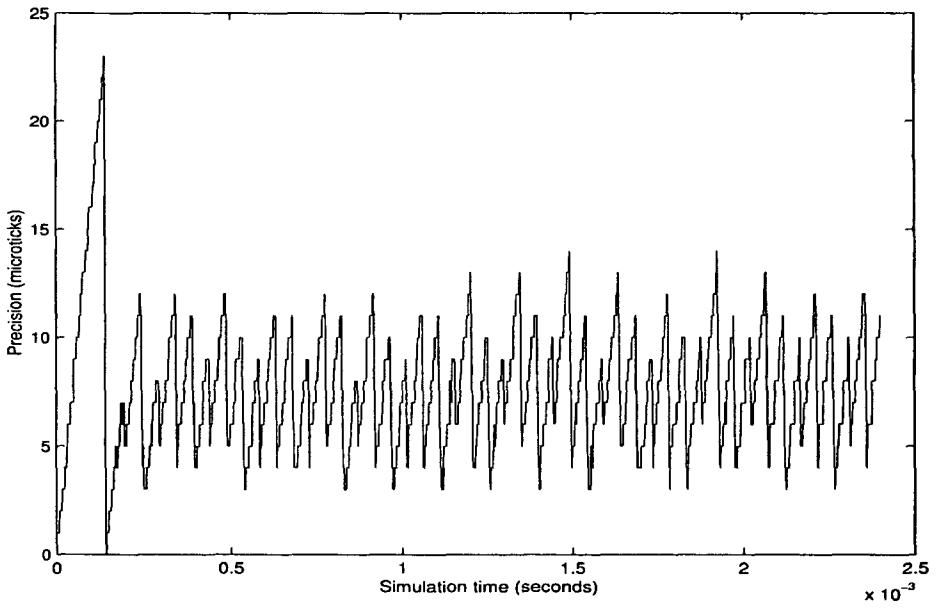

Figure C.1: Reference Test 2 - Cluster precision

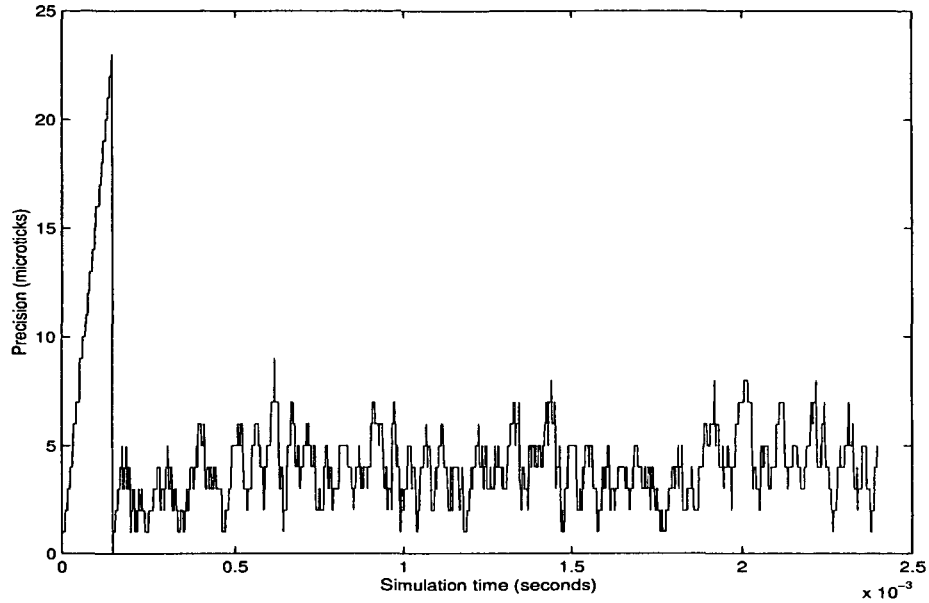

Figure C.2: Reference Test 3 - Cluster precision

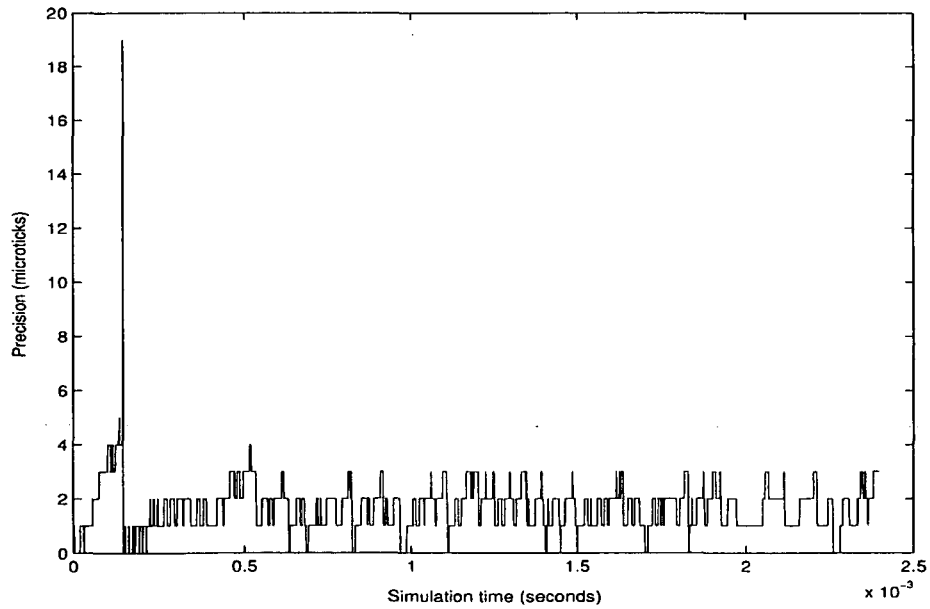

Figure C.3: Reference Test 5 - Cluster precision

Curriculum Vitae

Alexander Hanzlik

March 26, 1970	Born in Vienna, Austria
September 1976 – July 1980	Elementary School in Vienna, Austria
September 1980 – May 1988	Secondary School with bias on Latin and Greek Bundesgymnasium 3, Kundmanngasse 22, Vienna, Austria Graduation with distinction
November 1988 – June 1989	Military Service, Landwehrstammregiment 22, Austria Officer cadet, quit due to injury
October 1989 – June 1995	Studies of Computer Science at the Technische Universität Wien, Vienna, Austria Graduation in Computer Science
October 1995 – November 1998	Frequentis Nachrichtentechnik Ges.m.b.H. Software engineering for the ARTEMIS project (voice communication system for air traffic control) for the Service de Navigation Aérienne (STNA)
December 1998 – date	SIEMENS AG Austria Software engineering in the field of embedded systems for telecommunication applications
March 2001 – date	Doctoral Studies in Technical Sciences Technische Universität Wien, Vienna, Austria
November 2003 – date	Research Assistant Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria