SIDERA - a Simulation Model for Time-Triggered Distributed Real-Time Systems

Alexander Hanzlik[1]

Abstract – *SIDERA (SImulation model for DEpendable Real-time Architectures) is a simulation model for time-triggered distributed real-time systems. It is based on the Time-Triggered Architecture TTA and allows the simulation of large-scale real-time systems. SIDERA provides simulation of various real-time protocol services like system startup, communication, clock synchronization, membership service and protocol error detection and handling. A failure simulation module allows testing the stability of the systems under investigation in the presence of node failures. This paper is about basic concepts of time-triggered distributed systems and the implementation of these concepts in a simulation environment. Further, it provides a case study of clock synchronization in distributed systems and presents an approach that significantly improves the synchronism in a distributed system while reducing the need for high-quality oscillators.*

Keywords: *Simulation, Clock Synchronization, Time-Triggered Systems, Distributed Real-Time Computing.*

I. Introduction

Simulation is a reasonable, powerful and frequently used mean for gaining insight into the functionality and structure of distributed systems. SIDERA has been developed for the simulation of large-scale time-triggered systems, with the focus on simulation of fault-tolerant clock synchronization algorithms. The paper is structured as follows: the rest of this section consists of some related work, the system structure that we are dealing with and a short overview of the Time-Triggered Architecture. Section 2 introduces the five basic concepts of the Time-Triggered Protocol TTP that are essential for the simulation. Section 3 describes how these basic concepts are implemented in SIDERA. Section 4 gives an overview of the features that SIDERA provides. Section 5 presents a simulation case study and a clock synchronization algorithm that remarkably improves the synchronism among an ensemble of clocks in a distributed system. Section 6 concludes the paper.

I.1. Related work

[1] surveys 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 [2]. SimUTC has been developed in the course of the SynUTC project[1] 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 ([3], [4]) 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.

[5] 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.

[6] 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.

[7] analyzes 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.

[8] 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

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

in CPU and network load.

[9] compares the performance of different clock synchronization algorithms and clock correction strategies with regard to achievable synchrony by means of simulation. [10] deals with the simulation of multi-cluster clock synchronization strategies in the course of the ClockSync project [11], which is concerned with the development of a large simulation model for the synchronization of clocks in a distributed real-time system.

[12] presents the CESIUM tool that addresses the problem of testing communication protocols considering various obstacles to testing such as observability, controllability, deterministic experiment control and absence of the probe effect. The tool has been used for various testing experiments on implementations of protocols, covering group membership protocols [13] and clock synchronization algorithms.

[14] presents a discrete event distributed simulation framework that addresses the increased complexity of simulation models and the related modeling which demands the execution of simulations on multiple processors. The framework supports easy and fast development of distributed simulations and efficient adaptive synchronization of simulation processes.

[15] introduces DSSimulator, an object oriented simulation model which provides a scalable and flexible platform for the simulation of distributed systems. It deploys different distributed applications and is designed for the simulation of a large number of nodes on a single workstation.

I.2. System structure

We assume that a system can be built by repetitive use of the following components:

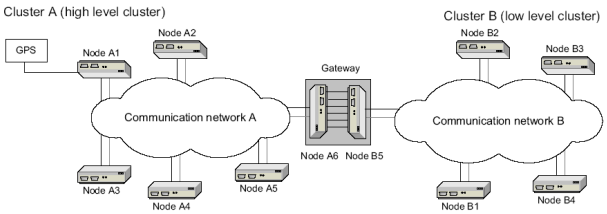


Fig. 1 System Structure.

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

Communication network. A shared communication resource connecting nodes.

Cluster. A set of spatially separated nodes that exchange messages via a communication network and that execute a distributed application in a cooperative manner.

Gateway. A connection between two clusters. It consists of a pair of nodes, one in each cluster, that communicate over a dedicated communication link.

Figure 1 shows an example for a system consisting of two clusters connected via a gateway.

I.3. *The Time-Triggered Architecture*

The conceptual basics of SIDERA are the Time-Triggered Architecture TTA [16] and the Time-Triggered Protocol TTP [17], both developed at the Vienna University of Technology for over more than twenty years.

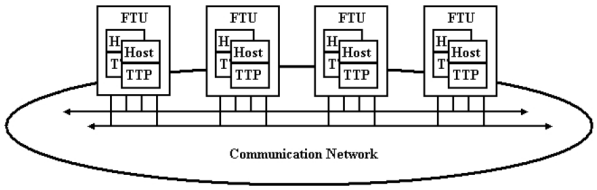


Fig. 2 TTA cluster.

Figure 2 depicts a *cluster* of the TTA. It consists of a set of Fault-Tolerant Units (FTUs) that communicate using the Time-Triggered Protocol (TTP). The host application covers the application specific functionalities of the FTU, whereas the TTP handles protocol specific tasks (startup, clock synchronization, communication etc.). Access to the communication network is controlled by a cyclic time-division multiple access (TDMA) scheme derived from a global notion of time. An FTU can consist of one, two or more nodes and provides the specified service without delay even after the failure of a node.

II. Basic concepts

This section gives an overview of the basic concepts of the simulation model.

II.1. Communication medium access control

In distributed systems with a shared communication resource the access to the communication medium has to be regulated such that all nodes are able to deliver messages within an upper bound in time. In time-triggered protocols like TTP [17] or FlexRay [18], the access to the communication medium is controlled by a collision-free, Time Division Multiple Access strategy (TDMA). Real-time is divided into *slots*. Each node is assigned one slot during which it is allowed to send. The sequence of slots in which each node sends at most one message forms a *TDMA round*.

Figure 3 shows the principle of operation.

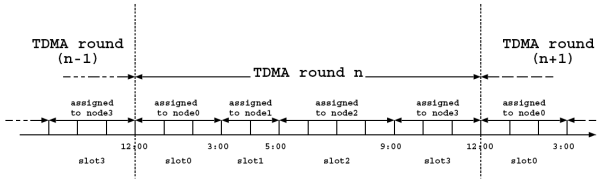


Fig. 3 TDMA scheme.

II.2. Global time

A key issue in time-triggered systems is the establishment of a fault-tolerant, global time base using a *clock synchronization algorithm* ([7], [19], [20]). A common notion of time among the nodes in a cluster is a prerequisite for a TDMA access strategy to the communication medium.

Internal clock synchronization. The task of internal clock synchronization is to maintain a global time base that all nodes agree on. The TTP protocol utilizes the concept of microticks and macroticks [21]. Microticks correspond to the local oscillator ticks at each node, while macroticks represent the global notion of time used to trigger and order global events. 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, according to the clock state correction term that is delivered periodically by the clock synchronization algorithm. 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 the node's view of the global time (cluster time). Each node uses the macrotick counter for the timestamping and ordering of events that occur in the distributed system.

External clock synchronization. The task of external clock synchronization is to bring the internally synchronized cluster time into agreement with external reference time. In a cluster there has to be at least one node that provides reference time to the other nodes within its cluster, referred to as *reference time server* [22] or *time master node* [23], respectively.

Multi-cluster clock synchronization. Multi-cluster clock synchronization aims at bringing the global times of an ensemble of clusters into agreement. An unidirectional flow of timing information is provided via the gateways. The cluster times are synchronized by means of external clock synchronization. In Figure 1, the global time of cluster A (provided by node A6) serves as reference time for cluster B (node B5 is the reference time server for cluster B).

II.3. Membership

Membership. A membership service provides each node with a consistent view of the operational state of the other nodes [24].

II.4. Protocol error detection and node deactivation

Each node in the system periodically checks whether it fulfills the conditions for correct operation dictated by the communication protocol and deactivates itself upon detection of a protocol error.

II.5. System startup and node reintegration

Whereas clock synchronization aims at maintaining a global time base within a distributed system, many clock synchronization algorithms require that the clocks are initially synchronized. This is the task of a startup algorithm [25]. Further, a mechanism is required for the reintegration of nodes that have stopped operation due to the detection of a protocol error.

III. Implementation concepts

This section gives implementation-specific aspects of the basic concepts introduced in the last section.

III.1. Communication medium access control

Each simulated node maintains a quartz oscillator. A microtick counter is incremented after a given number of oscillator ticks. The duration of the oscillator tick is dependant on a node-individual *sys_drift* value (Table I) that determines whether the clock of this node is running fast or slow against real-time.

Real-time denotes the progression of physical time that can be measured using clocks. In a real system, to be able to classify clocks as fast or slow, it is necessary to define a *reference clock* that serves as a standard for time measurement in this system and to relate the clock rate of the reference clock to the rates of the clocks under consideration. In a simulation environment, the observed rates of the simulated clocks are related to the progression of simulation time which serves as a reference clock.

A macrotick counter is incremented after a given number of microticks. The macrotick counter at each node represents the node's view of global time.

The access to the communication medium is based on a Time Division Multiple Access (TDMA) strategy. Real-time is divided into *slots*. Slots are assigned to nodes. A list of slot entries forms a *communication schedule*.

TABLE I
NODE SPECIFIC OFFLINE PARAMETERS

Simulation model variable	Meaning
nodeID	node identifier
OSC	oscillator ticks per microtick
LCP	microticks per macrotick
sys_drift	systematic drift of the node's clock
gateway_node	gateway node flag
time_master_node	time master node flag
free_running_MT_int	free running macroticks used for internal clock synchronization
free_running_MT_ext	free running macroticks used for external clock synchronization
CF	cold start allowed flag
cold_start_max	maximum number of frames to be sent in state COLD START

Each node traverses the communication schedule in a cyclic manner. A node enters a slot if its macrotick counter reaches *slot_start_time* (Table II). If *nodeID* (Table I) 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.

TABLE II
SLOT ENTRY IN COMMUNICATION SCHEDULE

Simulation model variable	Meaning
slot_start_time	start time of slot in macroticks
LogicalSenderName	nodeID of the sender in this slot
msg_send_time	message send time in macroticks
CS	Clock synchronization flag
SYF	Time difference capturing flag
RA	Reintegration allowed flag

III.2. *Global time*

In a time-triggered distributed system all nodes have to agree on a global notion of time. All nodes periodically adjust their local clocks according to a clock state correction term that is delivered by a clock synchronization algorithm.

Time difference capturing. A node has to know the deviation of its local clock to the other clocks to be able 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 II), all nodes use the arrival time of an incoming message for the estimation of the deviation of their clocks from the sender's clock. They calculate the expected arrival time (*msg_send_time*, Table II) in terms of local microticks and compare it to the observed arrival time (the microtick counter at message reception time). The result is the deviation from the sender's clock in terms of microticks and is stored in *deltas* (Table III), a push-down stack of depth four. Each time an estimate is added to *deltas*, the oldest estimate is discarded.

TABLE III
NODE SPECIFIC RUNTIME PARAMETERS

Simulation model variable	Meaning
deltas[4]	time difference capturing stack
slot_number	current slot in communication schedule
corr_term_int	current internal clock correction term
corr_term_ext	current external clock correction term
time_message	current deviation from external reference time
frame_ack_counter	frame acknowledgement counter
frame_inv_counter	frame invalid counter
frame_fail_counter	frame fail counter
iframe_counter	frames sent in state COLD START
c_state	current C-State of controller

Internal clock synchronization. At the end of a slot with the CS flag set (Table II), all nodes calculate an internal clock state correction term *corr_term_int* on the base of the estimates contained in *deltas* using the fault-tolerant average (FTA) algorithm [26]: the minimum and the maximum estimates are discarded, the internal correction term is the average of the two remaining estimates. Positive values of *corr_term_int* indicate that the local clock is running fast, negative values that the local clock is running slow against global 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 I) macroticks one microtick is corrected by manipulation of the local microtick counter 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. External clock synchronization is done by means of inter-cluster communication via *gateways* (Figure 1). A gateway consists of two interconnected nodes, a *gateway node* in one cluster and a *time master node* in another cluster. A node with the *gateway_node* flag set (Table I) provides its local time in terms of seconds to its associated time master node. A node with the *time_master_node* flag set calculates the deviation from gateway node time to local time in terms of seconds and distributes it to the other nodes (*time_message*) at its *msg_send_time* (Table II). Upon reception of a *time_message* all nodes calculate an external clock state correction term *corr_term_ext* (Table III) 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 I) macroticks one microtick is corrected by manipulation of the local microtick counter until *corr_term_ext* is exhausted. The local clocks run free afterwards until the reception of the next *time_message*.

III.3. *Membership*

The membership service keeps the nodes informed about the operational state of the other nodes in their cluster.

Message format. Each active node sends a message (*frame*) once per TDMA round containing its *controller state* (C-state).

TABLE IV
CONTROLLER STATE (C-STATE)

Simulation model variable	Meaning
Time	Global time in macroticks
MEDL_Entry	current slot in communication schedule
Membership	Membership vector

Membership. The membership vector is a bit-vector containing one bit entry for each node. Active nodes are set to 1 and inactive nodes are set to 0. The membership vector represents a node's view of the state of all nodes in the cluster. A receiver node adds a sender node to the membership 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 removes a sender from the membership if it receives an *invalid* frame during the slot of the sender or no frame at all (*null frame*).

Clique avoidance. The clique avoidance logic shall prevent a cluster from being partitioned into different subsets (*cliques*) that are not able to communicate with each other. Each node maintains a *frame_ack_counter*, a *frame_inv_counter* and a *frame_fail_counter* (Table III). Each node increments the *frame_ack_counter* when it receives a valid frame, the *frame_inv_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 stops operation.

III.4. *Protocol error detection and node deactivation*

Each node stops operation upon detection of one of the following protocol errors:

Clock synchronization error. When the internal 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 (Section III).

Communication system blackout. The node detects that no other node has sent a frame during the last TDMA round (i.e. the sum of *frame_ack_counter* and *frame_invalid_counter* is 0, Table III).

III.5. System startup and node reintegration

System startup (after initial power-on or after a communication system blackout) and node reintegration (after the detection of a protocol error) are based on a four-state model (Figure 4).

Timeouts. The following node-specific timeouts are of importance for correct operation of the node during startup and reintegration phase.

- 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*.

$$\begin{aligned} \tau_i^{startup} &= 0 : i = 0 \\ \tau_i^{startup} &= \sum_{j=1}^{i} \tau_{j-1}^{slot} : i > 0 \end{aligned} \quad (1)$$

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

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

$$\tau_i^{coldstart} = \tau_i^{startup} + \tau^{round} \quad (2)$$

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

$$\tau_i^{listen} = \tau_i^{startup} + 2 \times \tau^{round} \quad (3)$$

This choice for the listen timeout ensures that the longest cold start timeout is shorter than the shortest listen timeout [17].

Node states and state transitions. Figure 4 shows the states and state transitions of a node.

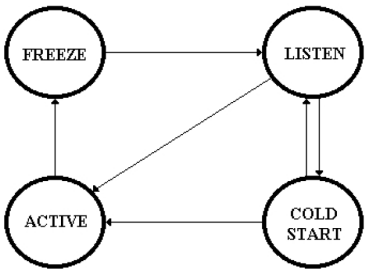


Fig. 4 Node states and state transitions.

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

  The node initializes its internal data structures, starts the listen timeout and transits to the LISTEN state.

- A node transits to the LISTEN state
  - 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 node is not allowed to enter COLD START state.

- 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, Table I) and
- *iframe_counter* (Table III) is less than *cold_start_max* (Table I).

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

Upon expiration of the cold start timeout the node increments *iframe_counter*, sends 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 send another frame, its starts the listen timeout and transits to state LISTEN.

- A node transits to the ACTIVE state
  - from state LISTEN
    upon reception of a valid frame (Section III). The C-state (Table IV) is copied from the received frame.
  - from state COLD START
    upon reception of a valid frame. The *iframe_counter* (Table III) is cleared.

III.6. *Failure simulation*

SIDERA provides a failure simulation module that allows testing the stability of the systems under consideration in the presence of node failures. A node failure occurs when a node stops operation due to the detection of a protocol error (Section III).

The following node failures can be modeled:

- Crash failures
  A node stops operation.
- Transmission failures
  A node sends an invalid frame at its message send time defined in the communication schedule (Section III).
- Clock state failures
  The microtick counter at a node changes to a specified value.
- Clock rate failures
  The systematic drift rate of a node changes to a specified value.

A *node failure script* consists of an arbitrary sequence of *node failure entries* and allows a node to fail in different ways during one simulation experiment.

A node failure entry defines

- the kind of node failure
- the point in simulation time at which the node failure occurs
- the duration of the node failure
- if the node recovers from the failure or not (for simulation of transient and permanent failures)

IV. SIDERA features

SIDERA provides

- simulation of single-cluster and multi-cluster time-triggered systems
- simulation of real-time protocol services (system startup, communication, clock synchronization, membership service, protocol error detection and handling)
- simulation of the FlexRay [18] clock synchronization algorithm
- node failure simulation

on a single computer system with no need for special hardware. SIDERA is written in C++ and consists of a simulation runtime module, a graphical user interface for generation of the simulation input parameters and a graphical analysis tool for interpretation of the simulation results (Figure 5).

The GUI and the analysis tool have been developed using Qt[2], a tool for convenient design of graphical user interfaces and cross-platform C++ development.

The current version of SIDERA runs on LINUX platforms (Kernel version 2.6)[3].

SIDERA has been validated by means of reference tests using a VHDL model of a TTPC/C1 controller in course of which it was shown that the simulation model follows the behavior of the VHDL reference model. The tests and the results can be found in [27].

Fig. 5 SIDERA simulation environment.

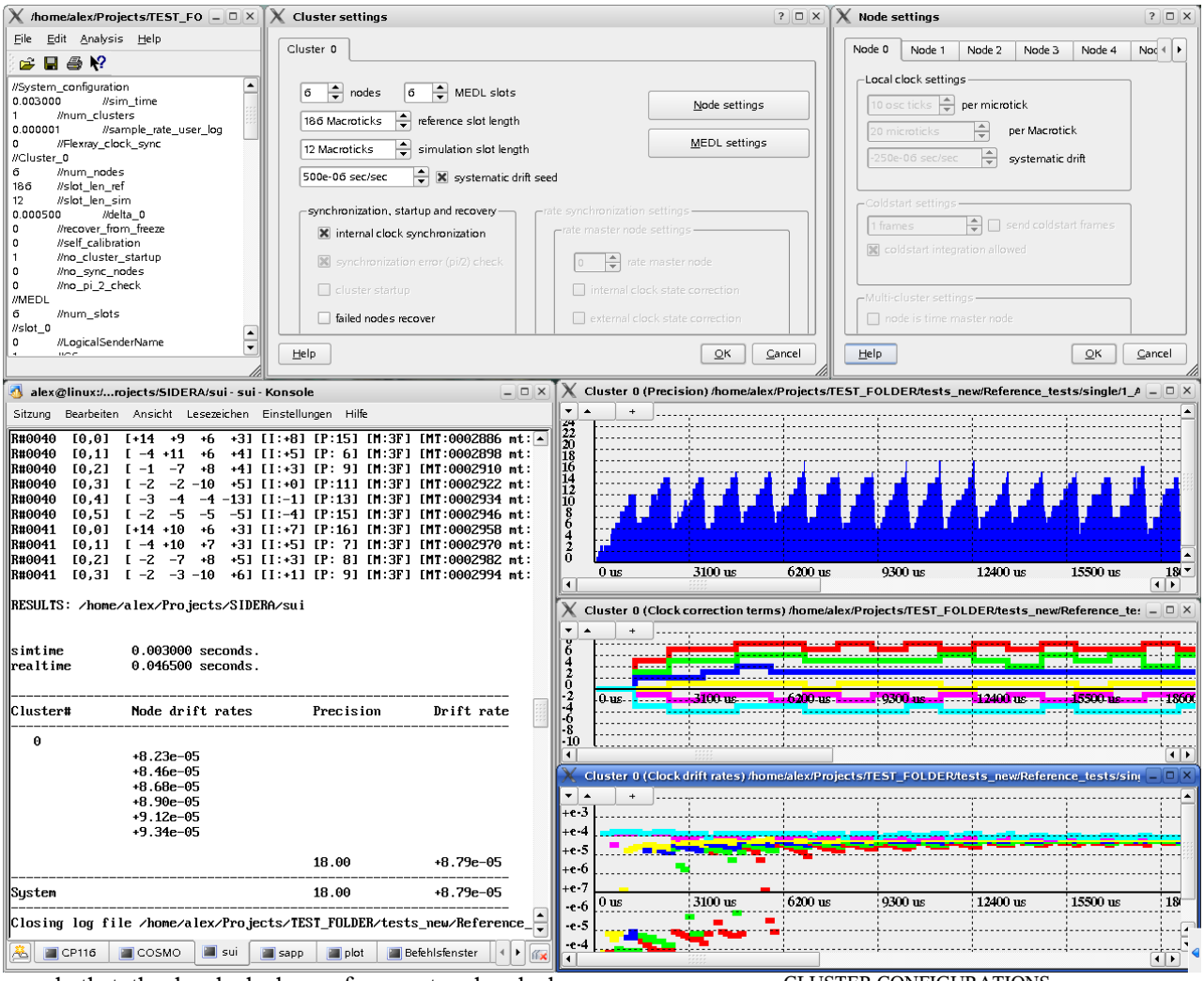
V. Simulation of clock synchronization algorithms - a case study

We have used SIDERA for the investigation of fault-tolerant clock synchronization algorithms. In the course of our investigations, we have developed a clock synchronization algorithm that combines fault-tolerant clock state correction with central clock rate correction. This algorithm significantly improves the achievable precision in time-triggered distributed systems while reducing the need for high-quality oscillators. The algorithm has been validated by means of hardware experiments, using the TTA for a case study [28].

In this section we consider a 2-cluster system like the one depicted in Figure 1 First, we will study the properties of two different single-cluster systems that perform internal clock synchronization using the fault-tolerant clock synchronization algorithm of the TTA (Section III). We will then connect the two internally synchronized clusters via a gateway (Section III) and analyze the achievable precision of this 2-cluster system using an external clock synchronization algorithm for the TTA [23]. Finally, we will apply our new algorithm

[2] Qt is available at www.trolltech.com.

[3] A demo version of SIDERA is available for download at http://www.vmars.tuwien.ac.at/people/alexhanzlik.html.



such that the local clocks perform not only clock state correction, but also clock rate correction during the clock synchronization process. We will show that the algorithm not only improves the achievable precision in single- and multi-cluster systems, but that it also integrates internal and external clock synchronization in multi-cluster systems.

V.1. *Experimental setup*

Table V summarizes the system configuration that we have used for the simulation experiments.

The numbering of clusters, nodes and slots starts from 0. The drift rate values describe the systematic drift of the nodes against simulation time. Negative values indicate fast running clocks and positive values indicate slow running clocks.

The figures used in the descriptions of the simulation experiments consist of a set of different windows. The numbering of the windows starts with 0, starting at the top window in each figure. The x-axis denotes the progression of simulation time with the same granularity for all windows (i.e. events on the same x-coordinate in different windows happen at the same point in simulation time).

TABLE V

CLUSTER CONFIGURATIONS

Property	Cluster 0	Cluster 1
Number of nodes	6	8
Number of TDMA slots	6	8
SYF slots	all	0,2,4,6
CS slot	5	6
Slot duration	2ms	1,5ms
TDMA round duration	12ms	12ms
Macrotick duration	1µs	1µs
Simulation time	2s	2s
Clock drift rate window	40ppm	55ppm
Drift rate Node 0 (s/s)	-2×10^{-5}	$-2{,}75 \times 10^{-5}$
Drift rate Node 1 (s/s)	$-1{,}2 \times 10^{-5}$	-2×10^{-5}
Drift rate Node 2 (s/s)	-4×10^{-6}	$-1{,}2 \times 10^{-5}$
Drift rate Node 3 (s/s)	4×10^{-6}	-4×10^{-5}
Drift rate Node 4 (s/s)	$1{,}2 \times 10^{-5}$	4×10^{-5}
Drift rate Node 5 (s/s)	2×10^{-5}	$1{,}2 \times 10^{-5}$
Drift rate Node 6 (s/s)	-	2×10^{-5}
Drift rate Node 7 (s/s)	-	$2{,}75 \times 10^{-5}$

V.2. *Experiment 1: Fault-tolerant clock state correction - single-cluster system*

In Experiment 1 we determine the achievable precision of two clusters that perform fault-tolerant clock state correction according to the internal clock synchronization algorithm used in the TTA (Section III).

According to Figure 6, Cluster 0 achieves a precision of 14 microticks (window 0) and Cluster 1 achieves a precision of 20 microticks (window 2). The *cluster drift rate* is the drift rate of the internally synchronized cluster time against simulation time. Cluster 0 shows a cluster drift rate of $7x10^{-6}s/s$ (i.e. it is slow against simulation time, window 1); Cluster 1 has a cluster drift rate of $-4x10^{-6}s/s$ (i.e. its cluster time proceeds fast against simulation time, window 3).

V.3. *Experiment 2: Fault-tolerant clock state correction - multi-cluster system*

In Experiment 2 we connect Cluster 0 and Cluster 1 via a gateway (Section III) to a 2-cluster system like the system shown in Figure 1 The gateway consists of the gateway node (Node 0) in Cluster 0 and of the time master node (Node 4) in Cluster 1.

The gateway provides a unidirectional flow of timing information from the gateway node to the time master node. In our experiment, Cluster 1 is externally synchronized to the global time of Cluster 0. The time master node in Cluster 1 periodically retrieves the local time from the gateway node in Cluster 0, determines the deviation from its local clock to the gateway node clock and distributes the deviation to the other nodes in its cluster by means of a *time message [23]*. At a pre-defined instant once per TDMA round, all nodes in Cluster 1 calculate an external clock state correction term and apply it to their local clocks to keep synchronized to reference time provided by the gateway clock.

Figure 7 shows the results for Experiment 2. The 2-cluster system achieves a precision of 22 microticks (window 0). Window 1 shows the external clock state correction terms determined by the time master node in Cluster 1 which are in the range of 5 microticks. Note that the external clock state correction terms are positive which indicates that the cluster time of Cluster 1 proceeds faster than that of Cluster 0 (we know that from Experiment 1). Therefore, all nodes in Cluster 1 have periodically to slow down their local clocks to keep in pace with the cluster drift rate of Cluster 0. Finally, window 2 shows the cluster drift rate of Cluster 1 which is $7x10^{-6}s/s$. Not surprisingly, this is equal to the cluster drift rate of Cluster 0 (Experiment 1) whose cluster time serves as reference time for Cluster 1.

V.4. *Experiment 3: Fault-tolerant clock state correction and central clock rate correction - single-cluster system*

In Experiment 3, we add a central clock rate correction algorithm to the fault-tolerant clock state correction algorithm to achieve a tighter synchronism among the clocks in a cluster.

For this purpose, we introduce the notion of a *rate master node*. A rate master node is a node whose local clock serves as a reference for the clock state and the clock rate of the other clocks in its cluster (that are referred to as *time keeping nodes*).

The combined fault-tolerant clock state and central clock rate correction algorithm works as follows:

- The rate master node and the time keeping nodes execute the fault-tolerant clock state correction algorithm as described in Section III.
- The time keeping nodes use the time difference capturing values from the rate master node to adjust their clock states and clock rates to that of the rate master node. This state and rate adjustment has to be bounded such that it does not interfere with the internal clock synchronization algorithm[4] [29].

[4] The state and rate correction at the time-keeping nodes is bounded to one microtick per TDMA round.

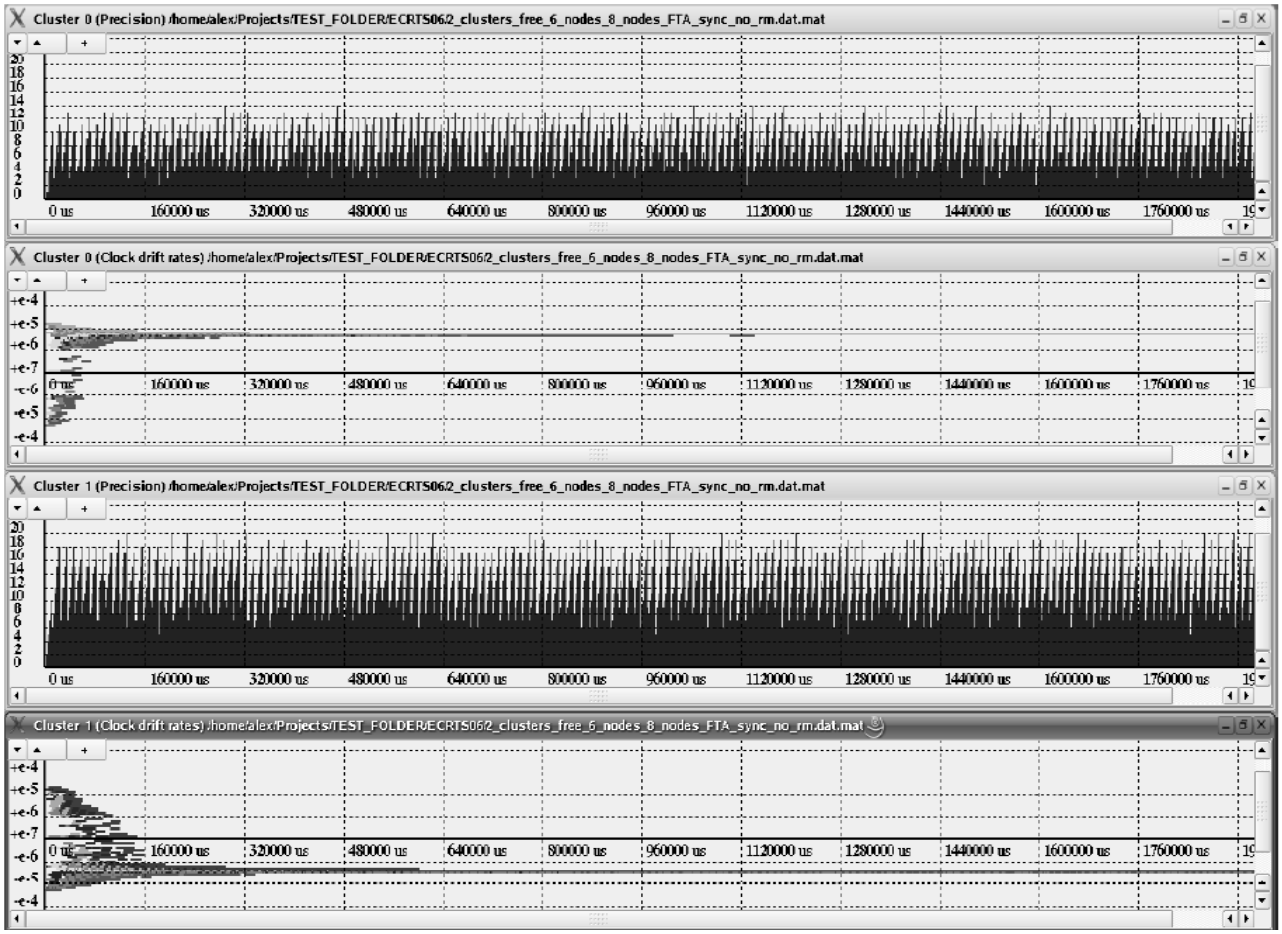


Fig. 6 Fault-tolerant clock state correction - single-cluster system.

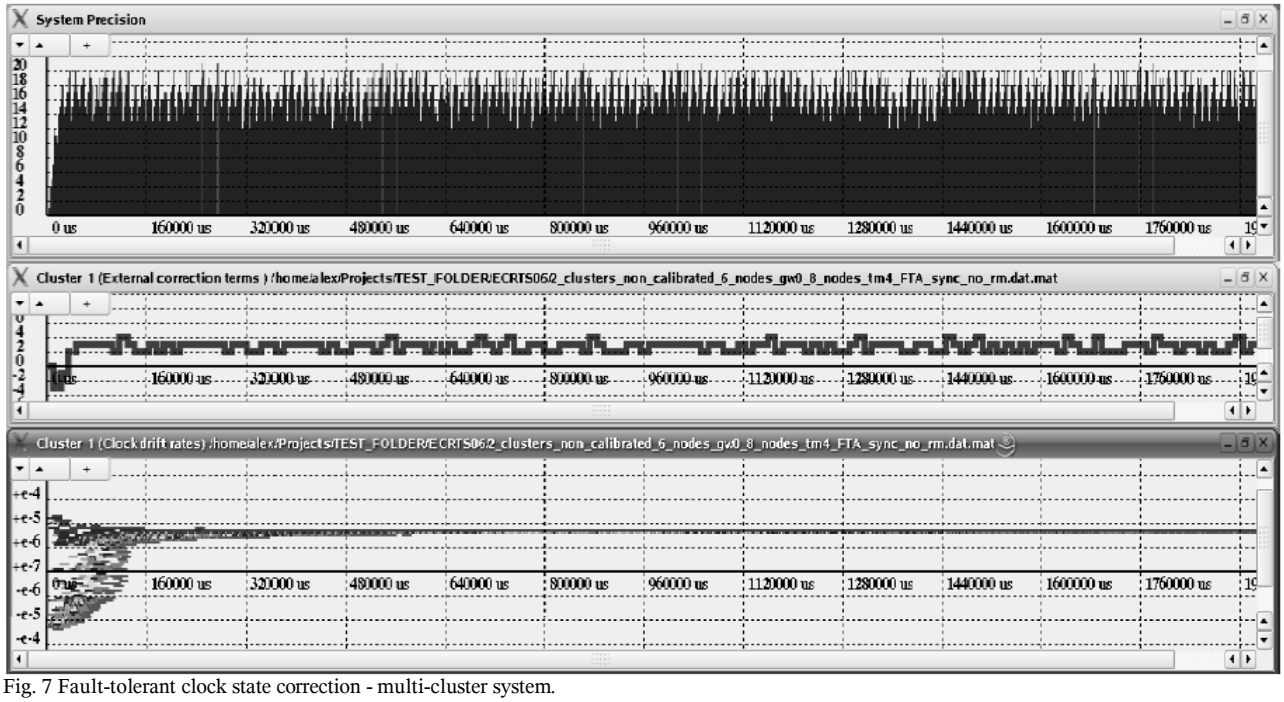


Fig. 7 Fault-tolerant clock state correction - multi-cluster system.

A detailed description of the combined algorithm as well as validation experiments with a TTA hardware cluster are presented in [28].

We will now analyze the performance of the combined algorithm by comparing the achievable precision of Cluster 0 and Cluster 1 to the results of Experiment 1. We use Node 2 as the rate master node in Cluster 0 and Node 4 as the rate master node in Cluster 1.

Figure 8 shows the results for Experiment 3. The combined algorithm improves the precision of Cluster 0 by 300% from 14 microticks to 4 microticks (window 0) and that of Cluster 1 by 500% from 20 microticks to 4 microticks (window 2). The cluster drift rate of Cluster 0 is $-4x10^{-6}s/s$ (window 1) and the cluster drift rate of Cluster 1 is $4x10^{-6}s/s$ (window 3). It can be seen from Table V that the cluster drift rates of both Cluster 0 and Cluster 1 follow the clock drift rates of their rate master nodes.

V.5. *Experiment 4: Fault-tolerant clock state correction and central clock rate correction - multi-cluster system*

The setup for Experiment 4 is very similar to the setup used for Experiment 2. Cluster 0 and Cluster 1 are connected via a gateway (Section III) to a 2-cluster system. The gateway consists of the gateway node (Node 0) in Cluster 0 and of the time master node (Node 4) in Cluster 1. The time master node in Cluster 1 is also the rate master node for Cluster 1.

The difference to Experiment 2 is that clock synchronization is performed using the combined clock state and clock rate correction algorithm as described in Section V.

Like in Experiment 2, the time master node in Cluster 1 is externally synchronized to the gateway node in Cluster 0. The time master node in Cluster 1 periodically retrieves the local time from the gateway node in Cluster 0 and adjusts its clock state and clock rate to that of the gateway clock[5].

From the point of view of Cluster 0, the time master node in Cluster 0 behaves like a time keeping node. The time master node (that is also the rate master node for Cluster 1) does not need to distribute a *time message* to the time keeping nodes like in Experiment 2. The time keeping nodes learn from the state and rate change at the time master node from the next time difference capturing value obtained from the time master node. Like in a single-cluster system, this state and rate change of the rate master node is handled by the combined clock state and clock rate correction algorithm. No explicit means for external clock synchronization are necessary at the time keeping nodes.

This approach integrates internal and external clock synchronization into one algorithm: within a cluster, all nodes establish an internally synchronized global time base that is externally synchronized to the clock state and the clock rate of the rate master node. The rate master node may be

- externally synchronized to a time standard like a GPS [30] receiver (like Node A1 in Cluster A in Figure 1)
- externally synchronized to the global time of another cluster (like Node B5 in Cluster B in Figure 1)
- not externally synchronized at all.

Further, this approach reduces the need for high-quality oscillators in distributed real-time systems. Due to the periodic rate adjustment at the time keeping nodes, the achievable precision does not depend on the systematic drift rates of the node clocks. Experimental results show that the short-term stability of crystal oscillators of average quality[6] is in the range of several hours [31]. This is remarkably longer than the clock rate adjustment period. With the exception of the rate master node that should be equipped with a high-quality oscillator, the time keeping nodes may deploy cheaper oscillators with a wider drift rate margin and a poorer long-term stability than high-quality (and more expensive) oscillators. This is meaningful in a market of mass production like the emerging automotive market for drive-by-wire systems, where the cost of every single component is scrutinized in order to find alternatives that are less costly [28].

Figure 9 shows the results for Experiment 4. The combined algorithm improves the precision of the 2-cluster system by 400% from 22 microticks to 5 microticks (window 0). The bounded clock state corrections (one microtick per TDMA round) at the time master node in Cluster 1 (window 1) according to the deviations from the gateway node in Cluster 0 are sufficient to maintain the system precision of 5 microticks . The system drift rate of the 2-cluster system equals $-4x10^{-6}s/s$. According to Table V, this is exactly the clock drift rate of the rate master node (Node 2) in Cluster 0 whose clock serves as a state and rate reference for the 2-cluster system.

VI. Conclusion

This paper presents a simulation model for time-triggered distributed real-time systems. It provides detailed information about the structure and the features of the simulation model. It also provides a simulation case study in the course of which a clock synchronization algorithm was developed that integrates internal and external clock synchronization in distributed systems and that remarkably improves

[5] For not to interfere with the internal clock synchronization algorithm, the state and rate correction at the time master node is bounded to one microtick per TDMA round.

[6] The oscillators used for the case study had a nominal frequency of 10Mhz and a frequency stability of 100ppm.

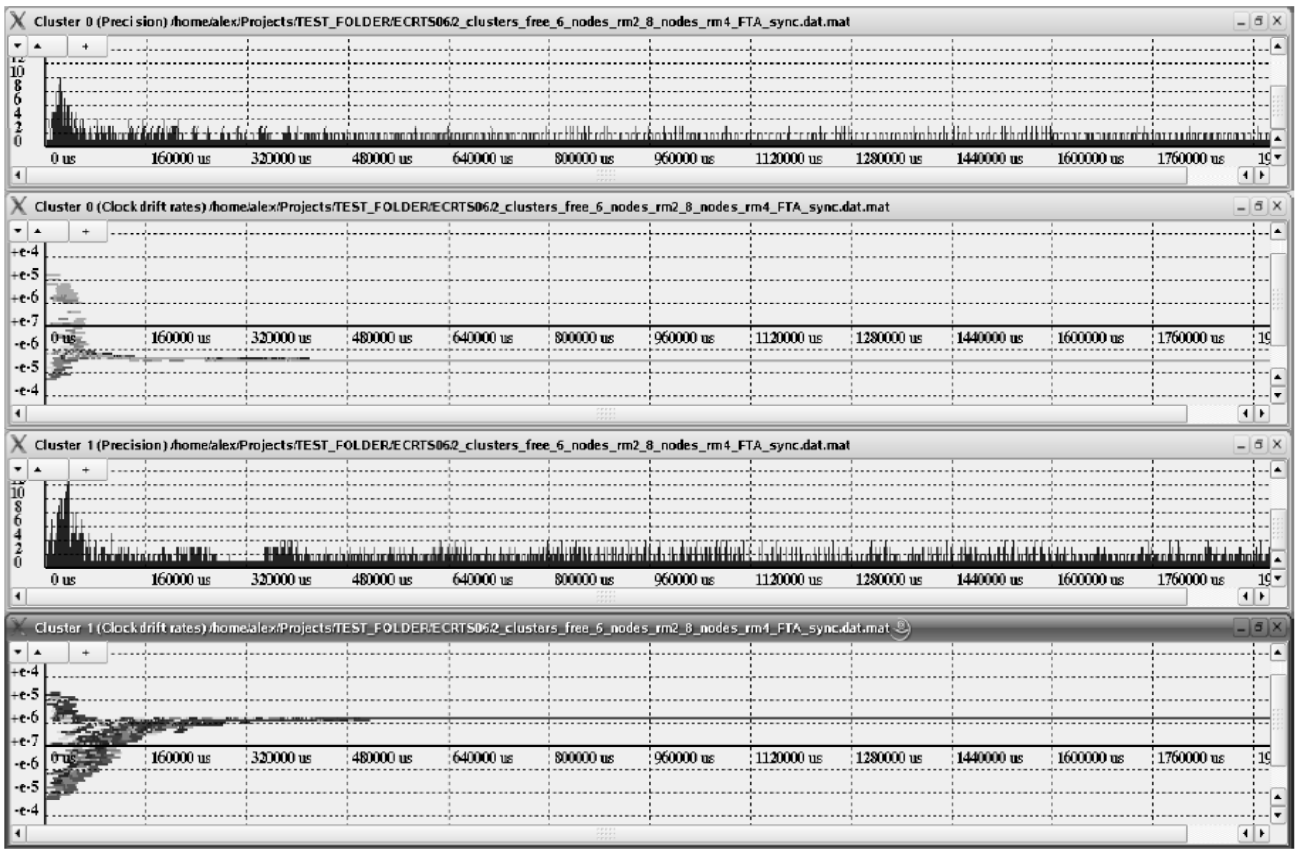


Fig. 8 Fault-tolerant clock state correction and central clock rate correction - single-cluster system.

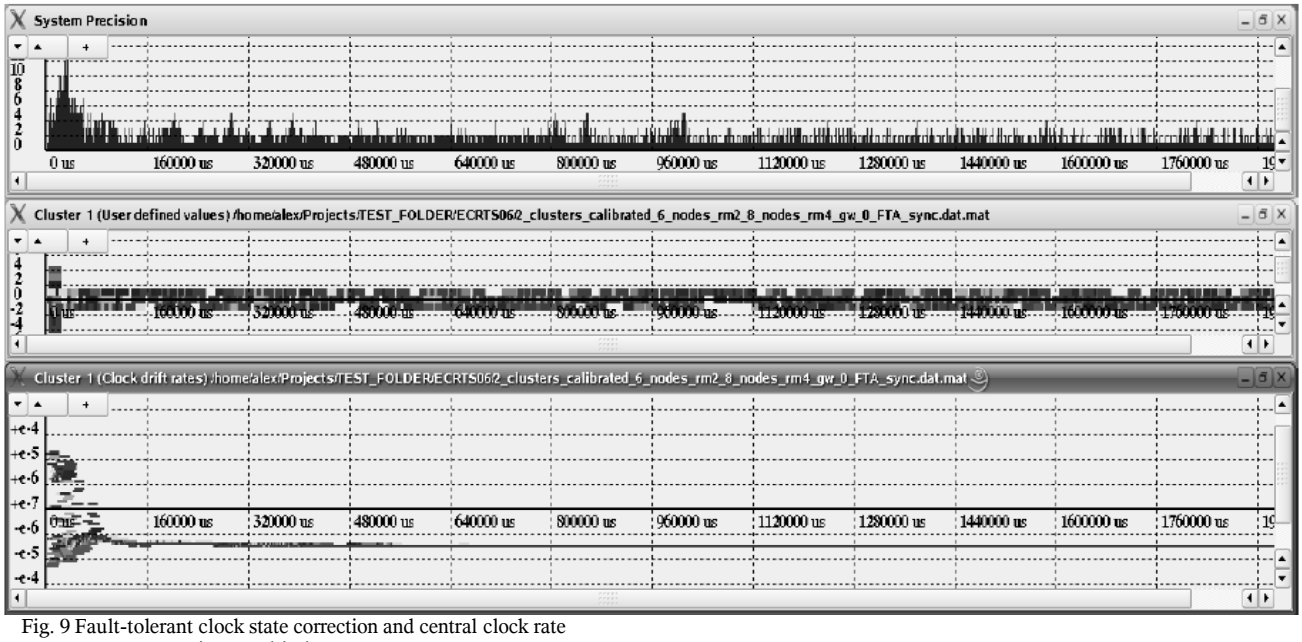


Fig. 9 Fault-tolerant clock state correction and central clock rate correction - multi-cluster system.

system precision while reducing the need for high-quality oscillators.

Beside the simulation of clock synchronization algorithms in distributed systems, SIDERA has been used for the investigation of startup algorithms and the

transition from asynchronous to synchronous operation in multi-cluster real-time systems [32] and for the investigation of the impact of transient communication system outages (blackouts) on the stability of clock synchronization in the TTA [27].

Currently SIDERA supports the analysis of a clock rate calibration mechanism used for the Time-Triggered Ethernet TTE [33] as well as for the investigation of the stability of the FlexRay clock synchronization algorithm in system configurations used for typical automotive applications [34].

Acknowledgements

This work was supported by the Austrian Science Fund, FWF, under project number P16638.

References

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

[2] M. C. Little and D. L. McCue, Construction and Use of a Simulation Package in C++, *C User's Journal*, Vol. 12, No. 3, 1994.

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

[4] Thomas M. Galla, *Cluster Simulation in Time-Triggered Real-Time Systems*, Ph.D. Thesis, Dept. Computer Engineering, University of Technology, Vienna, Austria, 1999.

[5] R. Pallierer, *Validation of Distributed Algorithms in Time-Triggered Systems by Simulation*, Ph.D. Thesis, Dept. Computer Engineering, University of Technology, Vienna, Austria, 2000.

[6] G. Bauer, *Implementation and Evaluation of a Fault-Tolerant Clock Synchronization Algorithm for TTP/C*, Master Thesis, Dept. Computer Engineering, University of Technology, Vienna, Austria, 1999.

[7] E. Anceaume and I. Puaut, *Performance Evaluation of Clock Synchronization Algorithms*, Technical Report No. 3526, Institut de Recherche en Informatique et Systemes Aleatoires, www.irisa.fr, October 1998.

[8] M. M. de Azevedo and D. M. Blough, *Software-Based Fault-Tolerant Clock Synchronization for Distributed UNIX Environments*, Technical Report No. ECE 94-03-01, Dept. Electrical and Computer Engineering, University of California, Irvine, March 1994.

[9] A. V. Schedl, *Design and Simulation of Clock Synchronization in Distributed Systems,* Ph.D. Thesis, Dept. Computer Engineering, University of Technology, Vienna, Austria, 1996.

[10] A. V. Schedl, *The Simulation of Multicluster Clock Synchronization Strategies*, Technical Report No. 22, Dept. Computer Engineering, University of Technology, Vienna, Austria, 1995.

[11] A. V. Schedl, *Introduction to the ClockSync Project*, Technical Report No. 19, Dept. Computer Engineering, University of Technology, Vienna, Austria, 1994.

[12] G. A. Alvarez and F. Cristian, Simulation-Based Testing of Communication Protocols for Dependable Embedded Systems, *Journal of Supercomputing*, Kluwer Academic Publishers, 1999.

[13] G. A. Alvarez and F. Cristian, Simulation-Based Test of Fault-Tolerant Group Membership Service, *Proceedings of the 12th Annual IEEE Conference on Computer Assurance, Gaithersburg, Maryland, June 1997.*

[14] B. Altuntas and Richard A. Wysk, A framework for adaptive synchronization of distributed simulations, *Proceedings of the 36th conference on Winter simulation, pp. 317-377, Washington, D.C., 2004*.

[15] J. Shamsi and M. Brockmeyer, DSSimulator: Achieving million node Simulation of Distributed Systems, *Applied Telecommunication Symposium. Spring Simulation Conference, San Diego, CA, April 2005*.

[16] Hermann Kopetz and Günther Bauer, The Time-Triggered Architecture, *Proceedings of the IEEE*, *Vol. 91, No. 1, pp. 112-126*, *2003*.

[17] TTTech, *Time-Triggered Protocol TTP/C*, Specification, TTTech Computertechnik AG, 2003, available at www.tttech.com.

[18] FlexRay Consortium, *FlexRay Communications System Protocol Specification Version 2.1*, 2005, available at www.flexray.com.

[19] Christof Fetzer and Flaviu Cristian, An Optimal Internal Clock Synchronization Algorithm, *Compass '95: 10th Annual Conference on Computer Assurance (National Institute of Standards and Technology), pp. 187-196, 1995*.

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

[21] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, ISBN 0792398947, 1997.

[22] Christof Fetzer and Flaviu Cristian, Integrating External and Internal Clock Synchronization, *Real-Time Systems*, Vol. 12, No. 2, pp.123-171, 1997.

[23] G. Bauer and M. Paulitsch, *External Clock Synchronization in the TTA*, Technical Report No. 3, Dept. Computer Engineering, University of Technology, Vienna, Austria, 2000.

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

[25] **Henrik Lonn, Initial synchronization of TDMA communication in distributed real-time systems, *The 19th International Conference on Distributed Computing Systems (ICDCS '99), pp. 370-379, Austin, TX, May 1999.***

[26] J. Lundelius and N. Lynch, A new Fault-tolerant Algorithm for Clock Synchronization, *Proceedings of the 3rd annual ACM symposium on Principles of Distributed Computing*, *pp. 75-88, 1984*.

[27] Alexander Hanzlik, *Investigation of Fault-Tolerant Multi-Cluster Clock Synchronization Strategies by Means of Simulation*, Ph.D. Thesis, Dept. Computer Engineering, University of Technology, Vienna, Austria, 2004.

[28] Hermann Kopetz, Astrit Ademaj and Alexander Hanzlik, Combination of clock-state and clock-rate correction in fault-tolerant distributed systems, *Real-Time Systems*, Vol. 33, pp. 139-173, Springer Netherlands, July 2006.

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

[30] P.H. Dana, Global Positioning System (GPS) time dissemination for real-time applications, *Real-Time Systems*, Vol. 12, pp. 9-40, January 1997.

[31] A. V. Schedl, *The Short-Term Stability of Crystal Oscillators: Experimental Results*, Technical Report No. 1, Dept. Computer Engineering, University of Technology, Vienna, Austria, 1995.

[32] Wilfried Steiner, Michael Paulitsch and Alexander Hanzlik, *Structuring of TTA Systems and Initial Synchronization*, Technical Report No. 17, Dept. Computer Engineering, University of Technology, Vienna, Austria, 2003.

[33] Hermann Kopetz, Astrit Ademaj, Petr Grillinger and Klaus Steinhammer, The Time-Triggered Ethernet (TTE) Design, *8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC), Seattle, Washington, 2005*.

[34] Alexander Hanzlik, Stability and Performance Analysis of Clock Synchronization in FlexRay, *International Review on Computers and Software*, Vol. 1, No. 2, pp. 146-155, September 2006.

Authors' information

[1,] Vienna University of Technology, Real-Time Systems Group. Treitlstr. 1-3/182-1, 1040 Vienna, Austria, hanzlik@vmars.tuwien.ac.at.

A. Hanzlik was born 1970 in Vienna, Austria. He received the master degree in Computer Science in 1995 and the Ph.D. in Technical Sciences in 2004, both from the Vienna University of Technology, Vienna, Austria.
Alex has done some research in the field of medical expert systems in the course of his master thesis. His main current research interests are fault-tolerant clock synchronization in distributed real-time systems and simulation of dependable real-time architectures.
Dr. Hanzlik is currently under contract at Siemens Austria in his professional life, where he works as a project manager mainly concerned with design and implementation of software and firmware for embedded systems for telecommunication and process control applications.