Simulation-Based Fault Injection as a Verification Oracle for the Engineering of Time-Triggered Ethernet networks

Loïc Fejoz, RealTime-at-Work, France
Bruno Regnier, CNES, France
Philippe Miramont, CNES, France
Nicolas Navet, University of Luxembourg

Abstract: TTEthernet (TTE) is considered for use as high-speed backbone in the avionics of next-generation orbital space launchers. Given the key role of communication in launchers, the OEM must acquire a precise understanding of TTE's functioning and its performances in nominal and error conditions. This holds especially true for the clock synchronization algorithm, the cornerstone of time-triggered communication in TTE, which involves complex distributed algorithms. In this study, we use both an experimental platform and fault-injection on a simulation model to gain quantified insights in these questions. We first describe a fine-grained simulation model of TTE model and discuss how it has been validated against communication traces recorded on the TTE platform. We then present experiments that evaluate the accuracy of the clock synchronization in TTE in the fault-free case as well as considering permanent link failure and transient transmission errors. Finally, we discuss what we have learned during the project in terms of development process and programming language support for complex simulation models used in the design of critical systems.

Keywords: Orbital launch system, Time-Triggered Ethernet, Simulation-Based Fault-Injection, clock synchronization, CPAL.

1 Introduction

1.1 Context of the study

The French Space Agency (CNES) is one of the stakeholders in the development of future generation launchers. TTEthernet (TTE) from TTTech company, based on AS6802 SAE standard [AS6802], is among the technologies of interest for future launchers [MiMoRe15]. TTE lifts the limitations of the legacy MIL-STD-1553B network in terms of bandwidth and addressing scheme, thus better fulfilling the requirements of next-generation launchers' avionics. In addition, the deterministic nature of time-triggered communication and the native support of fault-tolerance features in TTE facilitates the design of applications that must meet stringent dependability constraints.
This work on model-based fault-injection takes place in the broader context of a set of studies done by CNES, RealTime-at-Work and ONERA that includes the identification of possible network faults and their consequences on a reference architecture. The objective for CNES is to serve as a technical referent to ESA, for instance by performing independent assessments and cross-verifications in the field of dependability.

1.2 Complementing formal verification with simulation and tests

Formal methods provide mathematical proofs of certain properties of a system, which is of great value in the design of safety-critical systems. Over more than 2 decades, a lot of R&D efforts have been invested in proving key correctness properties of the Time-Triggered Architecture (TTA, see for instance [Pfe08]) with a particular focus on the two time-triggered communication networks TTP and TTE and their synchronization protocols (see [StDu13] for a good starting point). A valuable body of results about TT systems has actually been established on which the designer can build.
However, validating the correctness of a system cannot rely on formally established results alone for several reasons. First, there are not formal results known for all properties of interest, some of which are specific to the system/application under study. In addition, proofs are made under certain hypotheses (e.g., at most one faulty component at a time) and the designer needs to go beyond and understand the system's behavior outside the fault-hypothesis domain. For instance, the designer may want to gain quantified insights into the actual efficiency of possible "never-give-up" mechanisms. It also happens that commercial products simply do not fully comply with the standards. In addition, proofs are done based on specifications and standards which leave out important implementation choices and, usually, do not inform about, or even impose limits, on implementation overhead. For instance, by analyzing TTE network traces and comparing with the expected results of a simulation, we were able to figure out several implementation choices done in

TTE, like the number of observation windows, an important parameter of the synchronization algorithm, that is set to three in the products while the AS6802 standard does not impose it.
The mere process of implementing a fine-grained simulation model of TTE imposes to have a precise and almost complete understanding of its internal mechanisms. Importantly, confidence in the resulting model can only be truly gained by cross-verification with traces monitored on a real network. These reasons motivate the use of both an experimental platform and simulation models in addition to the existing formal results.

1.3 Simulation-based fault injection in the engineering of TTE networks

Injecting faults in a simulation model is a powerful technique to understand the behavior of a system when faults happen and to assess its overall robustness (see [NaCoMa16] for a recent survey on software fault-injection, including in models). In the case of a network like TTE, the transient and permanent faults that can typically happen are transmission errors on data frames and Protocol Control Frames (PCF), abnormal clock drifts, stations sending outside their specification (e.g., “babbling idiot” due to a clock failure or a software bug), link/ports failures, nodes that are not operational, defective switches, etc. Studying the effects of these faults on a simulation model provides evidence about the capability of TTE to tolerate these errors, and about the time it takes for the network to be back in full operational mode. More generally, simulation informs about the robustness of the network especially outside the scope of the design fault-hypothesis. In addition, experimenting with a model provides the designer with quantitative information to make architectural and configuration choices, for instance about redundancy strategies.

1.4 Contributions of the study

This work contributes to a better understanding of TTE, which is a powerful but complex protocol that, at the time of writing, has not yet been broadly deployed. Especially, experimental results shown in this paper shed some light on the behavior of the clock-synchronization mechanisms in the presence of faults. In that regard, this work is complementary to [StDu11, DuEaHaSt12, StDu13]. More generally, this work enriches the sparse literature on how to validate and optimize the configuration of TTE networks (see [TSPoSt15,BoDaNaMi16]).
The description of the TTE model, how it has been validated and used in this study can benefit engineers and researchers in the field. This work illustrates the feasibility of developing timing-accurate simulation model of complex networking technologies that can serve as building blocks for virtual platforms for instance used in Model-Driven Engineering (MDE) flow. Indeed, MDE relies on two complementary activities: modeling of the system on the one hand, and analysis of the non-functional properties, such as timing and dependability properties, on the other hand. This latter activity requires models, be they analytical or simulation models, to evaluate the fulfilment of non-functional properties. The reader can consult [BrNaHu17] for a case-study illustrating how to integrate and automate verification within MDE.

2 Modeling TTEthernet

2.1 The CPAL language

The modeling language and simulation environment used in this study is CPAL [NaFeHaAl16], which is freely available at https://www.designcps.com. CPAL is a domain-specific language developed by RealTime-at-Work and the University of Luxembourg that supports the programming, graphical visualization and simulation of embedded systems. CPAL can be used as a stand-alone tool or in a co-simulation environment, for instance in Matlab/Simulink® to develop control systems [SuAlHaNa16], or interfaced with RTaW-Pegase™ embedded network simulator to model high-level protocol layers.
A requirement of the project is that the models of TTE are available to end-users in source code so that they can be modified, especially to implement new fault scenarios. This motivated the choice of CPAL, which is a small and simple language (see [NaFe17]), based on Finite-State Machines, and other abstractions and modeling patterns that are natural in the domain of embedded systems. For instance, CPAL is well suited to implement software patterns for fault-tolerance and fault-injection (see [HuCBNa17]). Another interest of CPAL is that the simulation code can be re-used with no change for rapid-prototyping.

2.2 TTE model

TTE implements time-triggered transmission through a global clock, that is a system-wide time base that is established and maintained using a fault-tolerant clock synchronization protocol. This protocol involves the exchange of “Protocol Control Frames” (PCF) amongst nodes having distinct roles: “synchronization masters” (SM), “compression masters” (CM) and clients. Importantly, it is the role of the CM to execute the

"compression function", based on the values of the local clocks of the SMs previously received through PCFs, that decides the value of the global clock and thus the correction to apply to each of the local clocks. The clock adjustment is provided to the SMs in a PCF sent by each of the CMs, SMs will then execute a "convergence function" to derive the actual clock correction to apply. The reader is referred to [StDu13] for a good introduction to TTE's clock synchronization protocol, which is defined in the standard [AS6802].

Figure 1 shows the functional architecture of the TTE model instantiated on a small network comprising two end-systems, both acting as SMs and a communication switch serving as the CM.

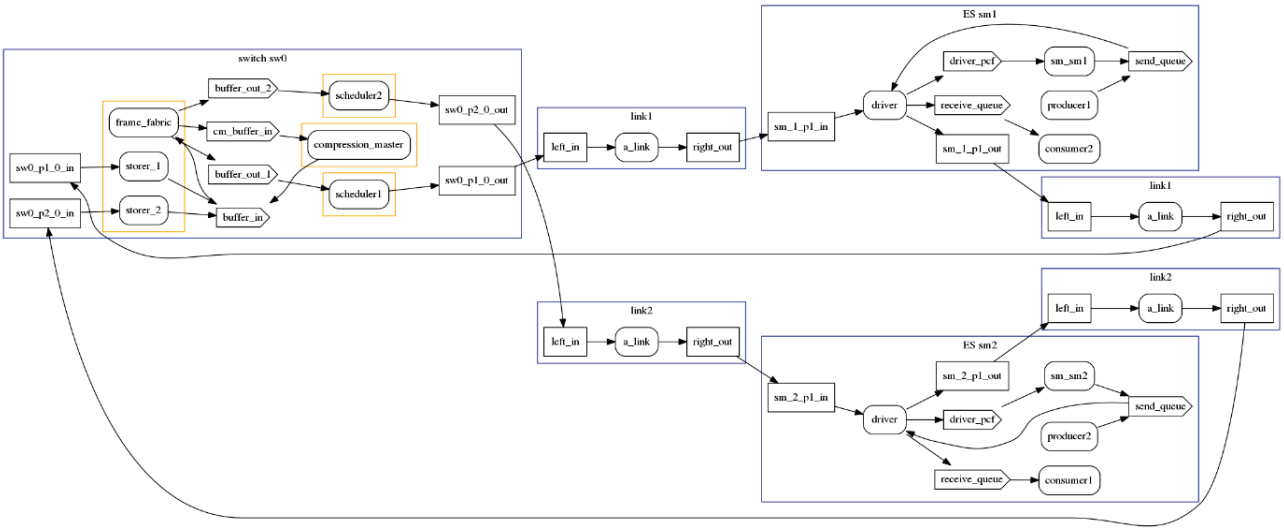


Figure 1: Functional architecture of the TTE model in CPAL with a switch (sw0), two end-systems (sm1 and sm2) and the 4 links between the end-systems and the switch (2 half-duplex links). The CPAL processes, which are activities of the model that can execute in parallel, are the rounded rectangles. The data exchanged among processes are the rectangles with the square angles and the arrow-shaped rectangles. Yellow rectangles in the switch means that activities such as sending and receiving frames can happen in parallel without interferences.

In CPAL, the functional architecture of the model, that is the processes and the flows of data among them, is extracted from the code during edition and visible to the developer within the development environment. At a lower level, the activities of the model are specified within *processes*, which are similar to functions having specific activation patterns that can be either event or time-triggered. Each process embeds at least an automaton that describes its internal logic. Figure 2 below shows for instance the states and the transitions between the states comprising the clock synchronization algorithm of a SM node.

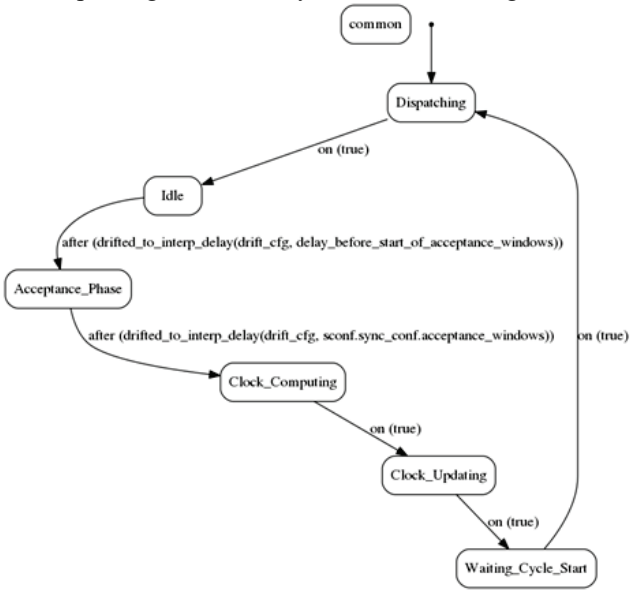


Figure 2: Automaton describing the logic of the clock synchronization algorithm executed in a SM node. A transition is taken if the associated condition evaluates to true. Blocks of code define the instructions to execute in the states and, optionally, in the transitions.

The TTE model developed in the project implements the transmission of PCF and TT frames. It also implements all the algorithms executed in the CMs and SMs for the synchronization of the clocks when the TTE network is in the synchronized state (e.g., not in the cold-start state, which is the topic of our ongoing work). The number of lines of code of the model, when instantiated in the larger configuration studied in the experiments of Section 3, comprises 3200 lines of CPAL code, considering that CPAL is a domain-specific language that leads to more compact code than C, C++ or Java. In terms of execution speed, the slowdown of simulation with respect to real time is about a factor 4 on a standard desktop machine (Intel I5), which is sufficient for the purpose of the study.

2.3 Model validation

Validation is the process by which confidence is gained in the relevance and accuracy of a model with respect to the real system. Validation is a mandatory step in critical systems where wrong design choices can lead to hazardous consequences. We used several techniques contributing to this objective: cross-validation with another simulation model, with measurements taken on the experimental platform, with formal results from the literature and with the results expected by mathematical reasoning.
Two network configurations were used in this study: a small configuration with 2 SMs (the end-systems) and one CM (the switch), whose communication cycle is shown in Figure 3, and a larger configuration described in Section 3.

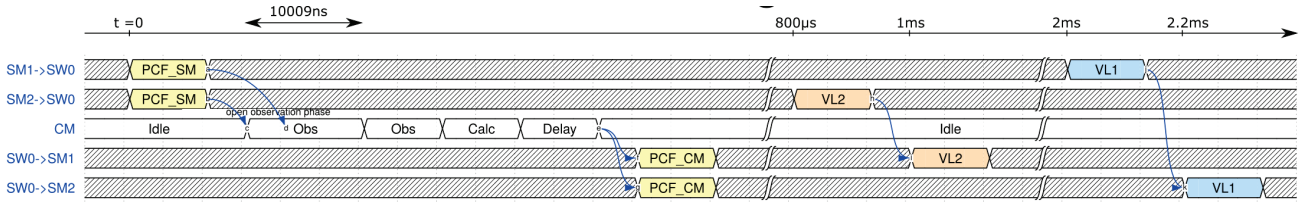


Figure 3: Excerpt of the 5ms communication cycle of the small TTE network as derived from the network configuration. Each SM sends a PCF to the CM, the first PCF received by the CM opens an observation window at the end of which the CM calculates the clock adjustment and send it to the SMs. The functional frames (streams VL1 and VL2) are transmitted from time 800us onwards.

Validation of the model was performed in successive steps:

1. The transmission schedule created by the model was first checked against the transmission schedule of the small configuration we manually derived based on our understanding of the expected behavior of the protocol ("pen and paper" solution).
2. The execution traces of the CPAL model were compared to the results of the TTE model implemented in RTaW-Pegase. A different team at RTaW developed the latter TTE model using a different simulation environment not based on CPAL. This diversity in the cross-validation techniques, tools and people involved is of course beneficial in terms of validation coverage.
3. The model outputs were compared to traces monitored with network taps on the small TTE network. In addition to verifying the timing correctness of the transmission schedule, traces help to validate the values contained in the frames and thus gain confidence in the implementation of the synchronization algorithms in the model. Importantly, traces also enabled to set parameters that depend on the implementation such as computation overhead, communication stacks and switch commutation latencies.
4. Finally, the quantified results on the desynchronization of the clocks derived from the model on the largest network were compared 1) with bounds from the literature obtained by model-checking [StDu11,DuEaHaSt12,StDu13] 2) estimations derived by mathematical reasoning about the maximum desynchronization given the intensity of the clock drifts.

3 Accuracy of clock synchronization

Once confidence has been gained in the correctness of the TTE model, simulations on a realistic configuration are performed with the objective to assess the quality of the clock synchronization considering clock drifts, implementation latencies in end-systems and switches, link failures and transmission errors.

3.1 Experimental setup

The configuration under study, shown in Figure 4, is made up of 4 SMs (the end-stations), 2 CMs (the switches). This configuration with 2 CMs supports the failure of a single CM ("single-failure hypothesis"). In the following the term "node" refer to any station on the network, be it a SM or a CM.

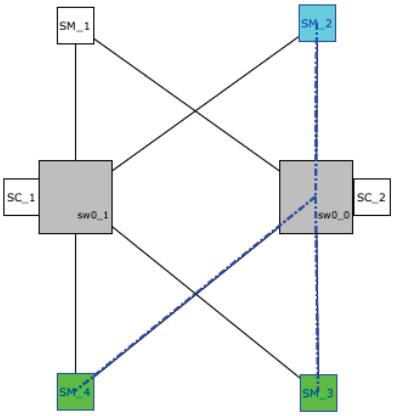


Figure 4: topology of the larger TTE network with two CMs. The blue end-system is the source of a multicast stream received by the green nodes (RTaW-Pegase screenshot).

Experimental conditions are a follows :

- The clocks of all nodes, CMs and SMs, drift apart at a rate of ±50ppm (parts per million), which means a desynchronization between any two clocks of at most 500ns per 5ms communication cycle. The drift for each node is drawn at random at the beginning of each communication cycle and remains constant over the cycle. This drift model, called *unstable* (cf [StDu13] p298), is not aimed at reproducing the actuality of drifts evolution, which varies mainly due to temperature changes thus with a slow dynamics, but to cover a search space that is as wide as possible.
- Two cases are considered, an *ideal case*, where implementation latencies in end-systems and switches are ignored, and the *actual case*, where delays are accounted for, with information provided by TTTech or through statistics on events seen in communications traces. The results in both the ideal and actual case are very close, which suggests that the network is robust to realistic implementation latencies. Hereafter, only the results in the actual case are presented.
- The clocks are not perfectly synchronized at the beginning of a simulation, but in a ±100ns interval.
- Although this is not imposed by TTE, the resynchronization of the clock occurs in our configuration at the very beginning of each 5ms communication cycle. The first frames sent in a communication cycle are the PCFs from the SMs containing their local clock value, followed by the frames from the CMs indicating to each SM the clock adjustment that is to be performed (similar as in Figure 3).
- The statistics are calculated over samples of at least 100 000 values (*i.e.*, at least 100 simulations of 5 seconds).

Two performance metrics measuring the accuracy of clock synchronization are considered:

- The **maximum residual desynchronization** during a communication cycle, which is the maximum difference between any two clocks in the system immediately after the last SM has re-synchronized its clock according to the PCFs received from the CMs.
- The **maximum desynchronization** during a communication cycle, which is the maximum difference between any two clocks in the system at any given point of a communication cycle. This quantity is also referred to in the literature as the *precision* of the synchronization [StDu11].

In the following, the values of these metrics are collected at each communication cycle during the simulation. Given that drifts are modeled as linear functions with constant slope over a cycle, desynchronization maxima occur immediately before resynchronization points.

3.2 Residual desynchronization in the fault-free scenario

Figure 5 below shows the empirical distribution of the *maximum residual desynchronization* between any two clocks in the system over a communication cycle.

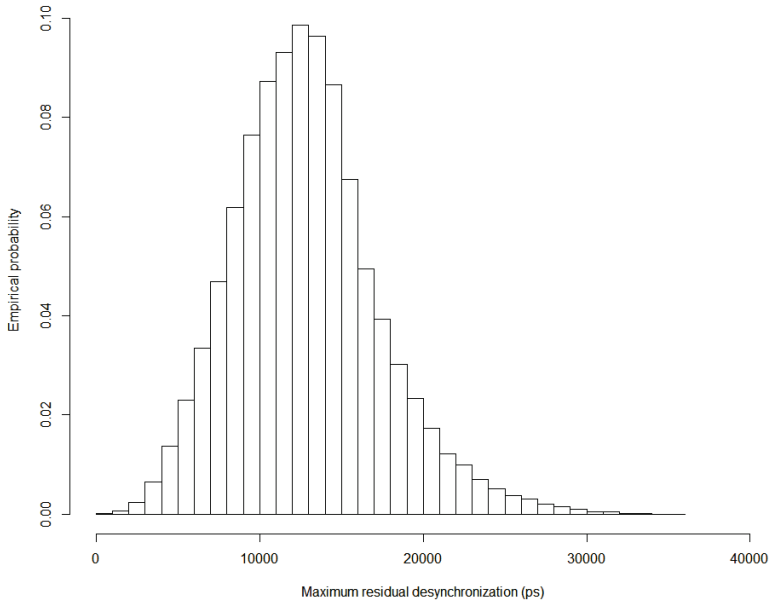


Figure 5: Empirical distribution of the *maximum residual desynchronization* among the clocks of all nodes measured after the last SM resynchronization (unit: picosecond).

The sample of the residual clock desynchronizations observed during simulation possesses the following characteristics expressed in picoseconds:

Min.	1st Qu.	Median	Mean	3rd Qu	Max.
431	9 828	12 545	12 840	15 331	35 569

Although not negligible, a residual desynchronization of maximum 36ns is small compared to the maximum 500ns clock drifts (*i.e.*, ±50ppm during 5ms), which shows that the TTE synchronization protocol is efficient in our setup to keep clocks synchronized over time. It is noteworthy that on the contrary to the maximum desynchronization (see §3.3) the residual desynchronization does not depend on the length of the communication cycle. It will however depends on the latencies incurred by the PCFs and, thus, the topology and configuration of the network.

Another observation is that the residual desynchronization between CMs is about twice smaller (around 18ns maximum in that configuration) as the residual desynchronization between any two nodes. Typically, the maximum divergence between "any two nodes" is observed between the clock of a SM and the clock of a CM. This can be explained because the residual desynchronization between CMs clocks essentially depends on the drifts occurring during the worst-case transmission time (WCTT) of one PCF (sent by SMs to CMs) while, between CMs and SMs clocks, the residual desynchronization depends on the WCTT of two PCFs (SMs to CMs, then CMs to SMs). The outputs of the model in our configuration with ±50ppm drifts, the implementation delays considered, and WCTT of the frames equal to 143µs600ns are in line with this reasoning.

3.3 Maximum desynchronization in the fault-free scenario

Figure 6 below shows the empirical distribution of the *maximum desynchronization* between any two clocks in the system over a communication cycle (called *precision* of the clock for short). At ±50ppm, the clocks of the nodes drift apart by at most 500ns over the 5ms communication cycle. This quantity is an uncompressible term that linearly depends on the intensity of the drifts and the length of the communication cycle. The rest of the desynchronization comes from the residual desynchronization that remains after the resynchronization done at the beginning of the communication cycle.

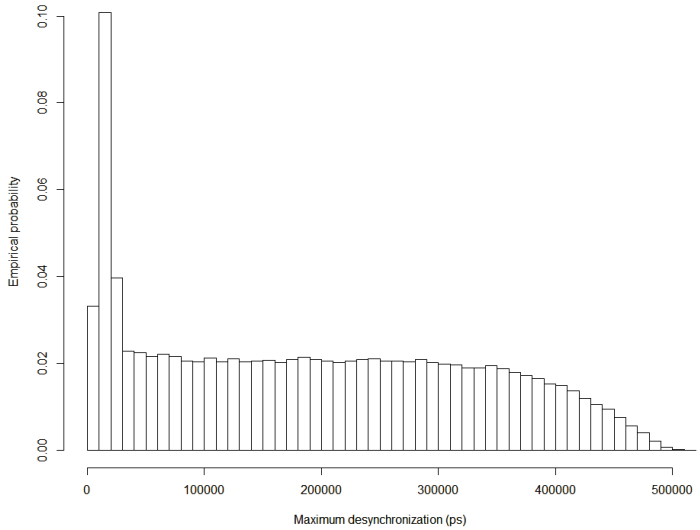


Figure 6: Empirical distribution of the *maximum desynchronization* among the clocks of all nodes during a communication cycle (unit: picosecond).

The sample of the maximum clock desynchronizations observed during the simulation possesses the following characteristics expressed in picoseconds:

Min.	1st Qu.	Median	Mean	3rd Qu	Max.
478	64 090	184 713	193 091	306 316	512 821

The precision is here thus well below the value of 10us9ps, assumed in our setup, which is a crucial parameter for the entire network configuration done by TTE tools. These simulation results are compatible with results from the literature derived by model-checking which state that "the precision of the system with an arbitrarily faulty SM is [..] the same as the precision in a fault-free system: 2 · *drift_offset*" (see [StDu11] p387), where *drift_offset* is the "sum of deviations of a clock from the perfect clock within one integration cycle" (see [StDu13] p294). The formal model is however more coarse-grained that the simulation model in this study and does not explicitly account for residual desynchronization, transmission time jitters and permanence function.

To detect possible trends in the evolution of the precision of the system over time, Figure 7 shows the empirical distribution of the maximum desynchronization between any two clocks *segmented into time intervals of 500ms*. Although there is a certain variability in the empirical distributions observed, no clear trend can be observed.

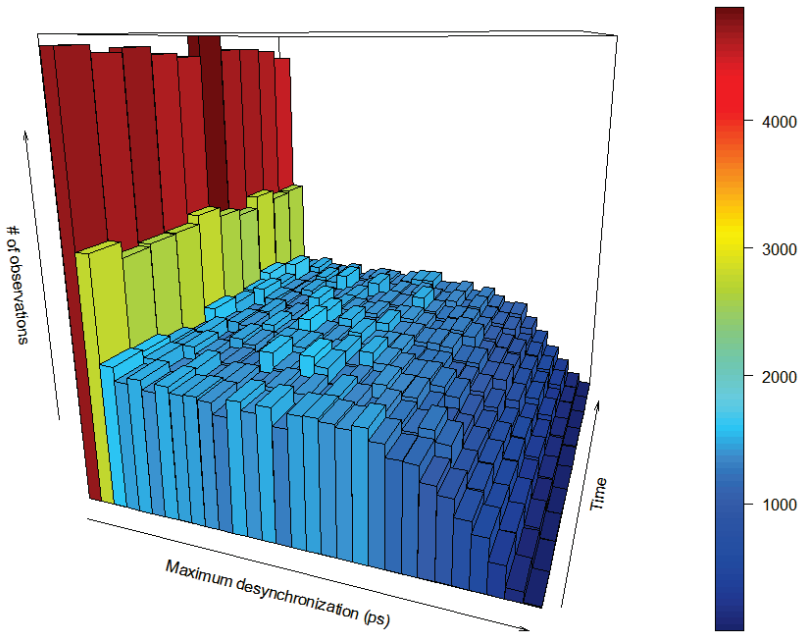
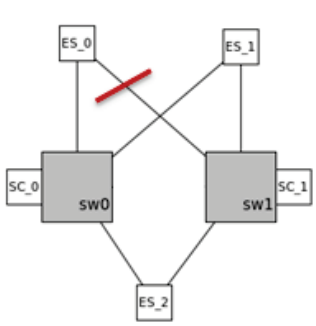
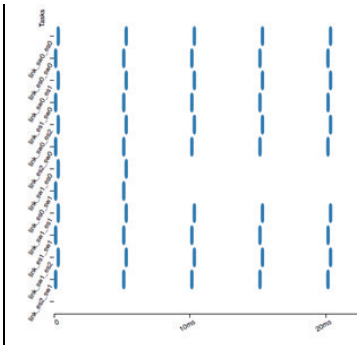


Figure 7: Variability of the empirical distributions of the maximum clock desynchronization (unit: picosecond). Simulations of length 5s are subdivided into 500ms intervals for which the empirical distributions are shown. The value on the right-hand side of the x-axis are the largest and less frequent values.

3.4 Maximum desynchronization will link failure and transmission errors

We now study TTE in the presence of permanent and transient faults. The first scenario, depicted in Figure 8, is the permanent loss of a link between a SM and a CM. SM ES0 will only receive the PCFs from a single CM, while the other SMs adjust their clocks according to the PCFs coming from the two CMs.





```
[0.005337070946] es0 SM Clock updating
with 2 PCFs
[0.005337070946] es1 SM Clock updating
with 2 PCFs
[0.005337070946] es2 SM Clock updating
with 2 PCFs
[0.010343805556] es0 SM Clock updating
with 1 PCFs
[0.010343805556] es1 SM Clock updating
with 2 PCFs
[0.010343805556] es2 SM Clock updating
with 2 PCFs
[0.015350540165] es0 SM Clock updating
with 1 PCFs
```

Figure 8: Scenario with a permanent link failure between the SM ES0 and the CM SW1 as shown in the left-hand side graphic. The middle part shows the transmission schedule (only PCFs frames) before and after the link failure that happens at time 1s. Two frames are then missing, a PCF from ES0 to SW1, and the PCF from SW1 to EC0 that contains the clock adjustment. As a result, ES0 updates its clock considering only one PCF (execution trace on the right-hand side).

The empirical distribution of the maximum desynchronizations with a permanent link failure is shown in Figure 9 (left-hand part). The maximum desynchronization observed is 720ns, that is 208ns more than without link failure (see §3.2). The 75% quantile is only 6ns larger with link failure. This suggests that the failure of a link, which is within the scope of the fault-hypothesis of our TTE configuration, does not substantially affects the precision of the clock synchronization. Here again the simulation results are in line with formal results considering inconsistent omission behavior failures.

In the next scenario, in addition to the link failure, we assume that PCFs can be corrupted by transmission errors for instance caused by EMI or α-particles. In the experiment, the error model is memoryless (*i.e.*, successive errors are not correlated) and each PCF has a probability of 3% to be corrupted. The number of consecutive errors is unbounded although its probability decreases exponentially fast. With transmission errors, it is possible for each of the nodes to not receive any PCF during a communication cycle. Such isolated node will not be able to adjust its clock and its drift with respect to the other nodes will be at most of 500ns per cycle. The probability of not adjusting the local clock is larger for the SM that can only communicate with one CM.

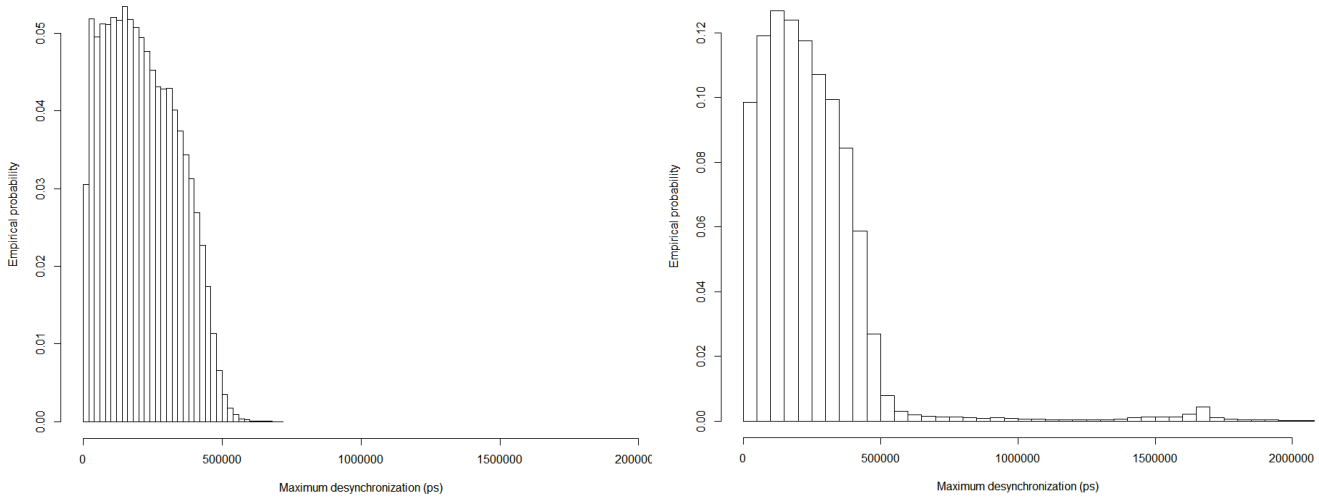


Figure 9: Empirical distribution of the maximum desynchronization among the clocks of all nodes collected during a communication cycle with a broken link. The left-hand graphic shows the case without transmission errors, while the right-hand graphics shows the case with 3% of transmission errors (unit: picosecond).

Figure 9 (right-hand part) shows the empirical distribution of the maximum desynchronizations with a permanent link failure and transmission errors. The maximum desynchronization observed is 2.41us, that is

more than 3 times more than without transmission errors. The 75% quantile however is only 18ns larger with transmission errors. In this situation, which is outside the fault-hypothesis of the design, the TTE network was able to keep nodes tightly synchronized and maintain communication with an important safety margin. Indeed the clock precision assumed for the configuration is 10us9ps. However, we think it must be verified that the precision assumed is in line with the errors that the system may face. For instance, more bursty transmission errors as in [NaSoSi00] may require reconsidering the clock precision that is assumed.

4 Discussion

The experiments reported in this paper suggest that, under realistic clock drifts, the TTE clock synchronization algorithm is efficient in maintaining a global time with a high precision. The quantitative results obtained with a fine-grained simulation model of TTE's clock synchronization protocol show results that are in line with the existing upper bounds derived by model-checking on a more abstract model. This study provides new insights and quantitative results about the residual clock desynchronisation, which can be seen as desynchronization carried over from the previous cycle. These residual desynchronizations do not add up over time but, if the worst-case transmission time of the PCFs is large, their value may not be negligible.

Even in the presence of one permanent link failure, the clock synchronization remains very precise. When, in addition to link failure, transient transmission errors can occur the precision of the clock synchronization in the worst-case drops significantly. However, in our simulation with a 3% frame error rate, the system remained always well within its tolerance margin. This suggests that if the occurrence of transmission errors is likely, the precision the global clock, which is assumed in the configuration of the network (e.g., for the size of the TT slots) should consider it and be decreased, at the expense of a lower protocol efficiency.

The analysis of TTE highlights that the accuracy of the clock synchronization entirely depends on the accuracy of the "transparent clock" mechanism that is implemented also in other protocols like IEEE1588. Transparent clocks transmitted in PCF frames serve to account for variable latencies due to message transmissions and delays in nodes (e.g. switching times). This implies that the correctness of its implementation in end-systems and switches is critical, and, for instance, that execution jitters must not degrade the precision of the transparent clock.

If building on TTE can simplify the design of fault-tolerant applications, internally it is a complex technology. The precise understanding of the distributed algorithms involved in the clock synchronization was facilitated by experiments on platforms but mostly by simulations. The calibration of the models with the actual implementation overhead required substantial efforts and was made possible thanks to TTTech technical support and statistics on communication traces.

CPAL proved to offer an adequate modeling and simulation environment with good modeling capabilities and sufficient simulation speed. The native support of physical time units and automata in the language, the possibility to visualize the protocol automata in addition to the functional architecture of the system, were key features in that regard. During the validation phase of the model, three issues in the code were identified by comparing the expected outputs of the model with its actual outputs. The analysis of the root causes of these issues led us to implement for the ongoing phase of the project systematic code review and traceability between the code of the model and the standard. We believe the latter objective should be facilitated by proper programming language support. For that purpose, we have implemented a library that extends the constructs of the CPAL language with respect to timers so that the state transitions defined in the TTE protocol automata can be directly translated into simulation code. In addition, to simplify the code of the model and reduce the distance between the model and TTE algorithms as defined in the standard, our ongoing work is to extend the CPAL simulation engine to support natively different time bases and clock-drifts. The corresponding code will then be removed from the model, and replaced by just a few configuration parameters.

The ongoing phase of the project is to model and study the startup of the TTE network, as well as situations where nodes do not share the same view of the network, forming "cliques". Of specific interest in the context of the launcher avionics is the time it takes for the network to reach the synchronized state, and the time needed for a node to re-integrate the network after it has been excluded from it by the clique detection algorithm.

5 References

[AS6802] “Time-Triggered Ethernet”, SAE standard AS6802, 2011.

[BoDaNaMi16] M. Boyer, H. Daigmorte, N. Navet, J. Migge, “Performance impact of the interactions between time –triggered and rate-constrained transmissions in TTEthernet", Embedded Real-Time Software and Systems (ERTS 2016), Toulouse, France, January 27-29, 2016.

[BrNaHu17] G. Brau, N. Navet and J. Hugues, "Heterogeneous models and analyses in the design of real-time embedded systems - an avionic case study", 25th International Conference on Real-Time Networks and Systems (RTNS’2017), Grenoble, France, October 4-6, 2017.

[DuEaHaSt12] B. Dutertre, A. Easwaran, B. Hall and W. Steiner, "Model-based analysis of Timed-Triggered Ethernet," 2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC), Williamsburg, VA, 2012, pp. 1-25. doi: 10.1109/DASC.2012.6383129.

[HuCBNa17] T. Hu, I. Cibrario Bertolotti, N. Navet, "Towards Seamless Integration of N-Version Programming in Model-Based Design", 22nd IEEE International Conference on Emerging Technologies And Factory Automation (ETFA’2017), Limassol, Cyprus, September 12-15, 2017.

[MiMoRe15] P. Miramont, D. Monchaux, B. Regnier, “New Technologies for Future Launchers Avionics”, DASIA 2015 – Data Systems in Aerospace, ESA Special Publication, Vol. 732, Sept. 2015.

[NaFe16] N. Navet, L. Fejoz, "CPAL: High-Level Abstractions for Safe Embedded Systems", 16th Workshop on Domain-specific Modeling Workshop (DSM), Amsterdam, October 30, 2016.

[NaFe17] N. Navet, L. Fejoz, "The CPAL Programming Language”, V1.08, July 2017.

[NaSoSi00] N. Navet, Y-Q. Song, F. Simonot, "Worst-Case Deadline Failure Probability in Real-Time Applications Distributed over CAN (Controller Area Network)", Journal of Systems Architecture, Elsevier Science, vol. 46, nº7, 2000.

[NaCoMa16] R. Natella, D. Cotroneo, and H. S. Madeira, “Assessing dependability with software fault injection: A survey,” ACM Comput. Surv., vol. 48, no. 3, pp. 44:1–44:55, Feb. 2016.

[Pfe08] Holger Pfeifer, “Formal Methods in the Automotive Domain: The Case of TTA”, Chapter 15 of the Automotive Embedded Systems Handbook, Taylor and Francis CRC Press, ISBN: 9780849380266, 2008.

[TSPoSt15] D. Tămaş-Selicean, P. Pop, and W. Steiner. “Design optimization of TTEthernet-based distributed real-time systems”. Real-Time Systems, vol. 51, n°1, pp1-35, January 2015.

[StDu11] Wilfried Steiner, Bruno Dutertre, “Automated Formal Verification of the TTEthernet Synchronization Quality”, NASA Formal Methods: Third International Symposium, NFM 2011, pp375-390, Pasadena, CA, USA, April 18-20, 2011.

[StDu13] W. Steiner, B. Dutertre, “The TTEthernet Synchronization protocols and their formal verification”, International Journal of Critical Computer-Based Systems, Vol. 4, n° 3, pp280-300, 2013.

[SuAlHaNa16] S. M. Sundharam, S. Altmeyer, L. Havet, N. Navet, "Model-Based Development Environment for Rapid-Prototyping of Latency-Sensitive Control Software", 6th International Symposium on Embedded Computing and System Design (ISED), Patna, India, December 15-17,2016.