Experimental Evaluation of the DECOS Fault-Tolerant Communication Layer

Jonny Vinter[1], Henrik Eriksson[1], Astrit Ademaj[2],
Bernhard Leiner[3], and Martin Schlager[3]

[1] SP Technical Research Institute of Sweden, {jonny.vinter, henrik.eriksson}@sp.se
[2] Vienna University of Technology, ademaj@vmars.tuvien.ac.at
[3] TTTech Computertechnik AG, {bernhard.leiner, martin.schlager}@tttech.com

Abstract. This paper presents an experimental evaluation of the fault-tolerant communication (FTCOM) layer of the DECOS integrated architecture. The FTCOM layer implements different agreement functions that detect and mask errors sent either by one node using replicated communication channels or by redundant nodes. DECOS facilitates a move from a federated to an integrated architecture which means that non-safety and safety-related applications run on the same hardware infrastructure and use the same network. Due to the increased amount of data caused by the integration, the FTCOM is partly implemented in hardware to speed up packing and unpacking of messages. A cluster of DECOS nodes is interconnected via a time-triggered bus where transient faults with varying duration are injected on the bus. The goal of the experiments is to evaluate the fault-handling mechanisms and different agreement functions of the FTCOM layer.

1 Introduction

In a modern upper class car the number of electronic control units (ECUs) has passed 50 and the total cable length is more than two km. As a consequence, the weight of the electronics systems constitutes a non-negligible part of the total weight, and the cost of trouble-not-identified problems (TNIs) associated with connector faults is raising [1]. The trend of using a single ECU for each new function is costly and for mass produced systems indefensible. Additionally, several emerging subsystems will require data from and control over other subsystems. An example could be a collision avoidance subsystem which has to simultaneously control both the steer-by-wire and brake-by-wire subsystems [2]. This becomes very complex to implement on an architecture based on the federated design approach where a single ECU is required for each function. All these issues imply that a move towards an integrated architecture is necessary and this is the goal of DECOS (Dependable Embedded Components and Systems) [3], and also for similar initiatives such as AUTOSAR [4]. In an integrated architecture several applications can and will share the same ECU which requires temporal and spatial separation of applications. Within DECOS such encapsulation is ensured by high-level services like encapsulated execution environments [5] as well as virtual networks [6]. Other high-level services provided by the DECOS architec-

ture are integrated services for diagnosis [7] and a new FTCOM layer based on the one used in TTP/C [8]. The goal of the DECOS FTCOM layer, whose validation is the focus of this paper, is to implement different agreement functions that detect and mask errors sent either by one node using replicated communication or by redundant nodes. Since the communication to and from a node will be increased in an integrated architecture, parts of the DECOS FTCOM layer are implemented in hardware to accelerate message packing and unpacking. The remainder of the paper is organized as follows. Section 2 briefly describes the DECOS fault-tolerant communication layer. The experimental setup used for the dependability evaluation is described in Section 3, and the results of the evaluation are presented and discussed in Section 4. Finally, the conclusions are given in Section 5.

2 DECOS FTCOM Layer

The FTCOM layer consists of one part which is implemented in hardware (Xilinx Virtex-4) together with the communication controller to accelerate frame sending and receiving as well as message packing and unpacking. During frame receiving, the two frame replicas received on the two replicated channels are checked by a CRC for validity and merged into a single virtual valid frame which is handed over to the upper FTCOM layers. The other parts of the FTCOM layer are implemented in software and are automatically generated by the node design tool [9]. The software-implemented parts have services for sender and receiver status as well as message age. However, the most important service is the provision of replica deterministic agreement (RDA) functions to handle replication by node redundancy. A number of useful agreement functions are provided. For *fail-silent subsystems*, where the subsystems produce correct messages or no messages at all, the agreement function '*one valid*' is provided. It uses the first incoming message as its agreed value. For *fail-consistent* subsystems, where incorrect messages can be received, the agreement function '*vote*' is provided. It uses majority voting to establish the agreed message and thus an odd number of replicas are needed. For *range-consistent* subsystems, e.g. redundant sensor values, the agreement function '*average*' is provided. It takes the arithmetic mean of the messages as its agreed value. For a case where the message is e.g. a boolean representing that the corresponding subsystem is alive, the agreement function '*add*' is provided, which sums up the living number of replicas and can therefore be used to get subsystem membership information. The function '*valid strict*' compares all valid raw values and only uses the (agreed) value if they are identical.

3 Experimental Setup

The DECOS cluster consists of a set of nodes that communicate with each other using replicated communication channels. A node knows the message sending instants of all the other nodes. Each node consists of a communication controller and a host application. The communication controller executes a time-triggered communication protocol, whereas user applications and the software part of the FTCOM layer are executed

in the host computer. Each DECOS node has a FPGA where the communication controller and the hardware part of the FTCOM layer are implemented. Application software, DECOS middleware and high-level services are implemented in the Infineon Tricore 1796 which is the host computer of the node. During the experiments, relevant data is logged in an external SRAM and after the experiments are finished data is downloaded via the debugger to a PC for analysis [10] (see Fig. 1). The disturbances are injected by the TTX - Disturbance Node [11] which is running synchronously with the cluster. A wide variety of protocol-independent faults can be injected such as bus breaks, stuck-at faults, babbling idiot faults and e.g. mismatched bus terminations. Faults can have a random, but bounded or specified duration, repeated at a random or specified frequency. The shortest pulse which can be injected has duration of 1 µs which corresponds to roughly 10-15 bits on the bus at a bandwidth of 10 Mbps. A counter application is used in this study to evaluate the FTCOM layer. The application toggles one boolean state variable and increments one float, and one integer variable, before the values of these variables are sent on the bus. The counter workload is executed on three redundant nodes. A fourth node is executing its RDA functions on the three replicated messages read on the bus to recover or mask an erroneous message. Each fault injection campaign is configured in an XML file where parameters such as fault type, target channel(s), fault duration and time between faults are defined.

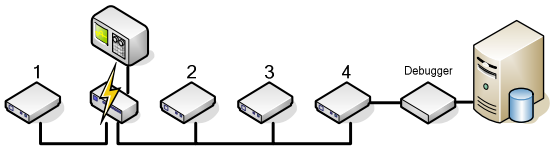

Fig. 1. Fault injection platform.

4 Results

Several fault injection campaigns are carried out to validate that the FTCOM layer handles a message failure at replicated channels or at redundant nodes. The results are presented in the following sub sections.

4.1 Message Failure at Replicated Channels

The goal of this campaign is to validate the error detection and masking mechanisms of frames sent in replicated channels. Transient stuck-at-0 faults (denoted SA0), stuck-at-1 faults (denoted SA1), and white noise (denoted WN) with different burst lengths (from 1µs up to 500µs) are injected on communication channel A or channel B. The types of erroneous frames observed are *Invalid* frames and *Incorrect* frames. An Invalid frame is a case when a frame was expected but not received, or the frame does not have the specified header, or a wrong length. An Incorrect frame is the case when there is a mismatch between the data and the calculated CRC value (which means that data in the frame or the CRC are corrupted by a fault).

The expected number of erroneous frames *nef* due to injected disturbances can be calculated as:

$$nef = nf \cdot \frac{(bl - 1 + fl)}{sl} \cdot \frac{3}{4} \qquad (1)$$

Where *nf* denotes the number of fault injected and *bl*, *fl*, and *sl* denotes the burst, length, frame length and slot length respectively (in µs). The ratio ¾ is needed because only three out of four nodes are analyzed while the fourth logs the results. In the counter workload, the frame and slot lengths are 35 and 625 µs, respectively and the number of injected faults is 310. For a burst length of 1 µs, the equation gives *nef* = 13 which agrees well with the experimental data. Equation (1) is also experimentally validated by a comparison between theoretical and practical results. The percentage of activated faults (Invalid and Incorrect frames) is shown in Fig. 2.

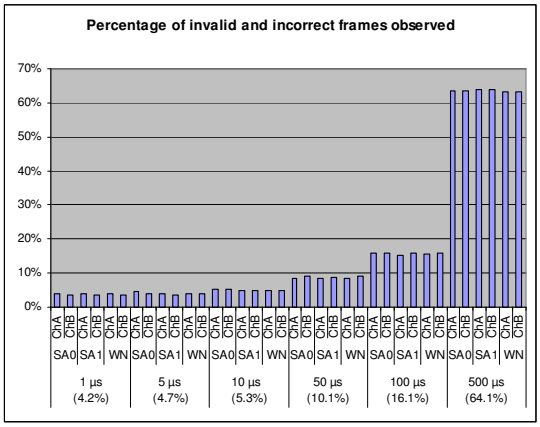

Fig. 2. Percentage of erroneous frames caused by different faults and burst lengths (presented both in micro seconds and as the probability of erroneous frames according to Equation 1).

4.2 Message Failure at Redundant Nodes

The goals of the campaigns presented in this section are to test the error detection and masking capabilities of the FTCOM layer in the case of failure of one of the redundant nodes. Here, the fault is injected in one of the redundant nodes, for example by disturbing the frames of the same node in both channels (campaign RN1). An additionally test was performed (RN2) to check the correctness at application level. In campaign RN3, faults are injected in the application layer of one node, forcing the node to produce value failures.

4.2.1 Campaign RN1

The type of faults used in RN1 could be caused by an erroneous packing of frames in the sending node. Fig. 3 shows the percentage of detected erroneous frames caused by simultaneous faults on both channels.

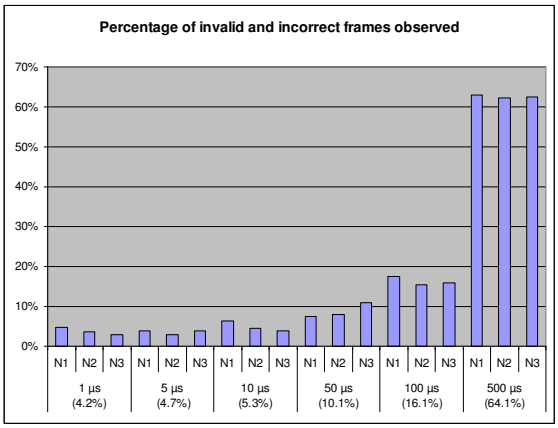

Fig. 3. Percentage of erroneous frames (observed at different nodes) caused by a uniform distribution of SA0, SA1 and WN faults with different burst lengths.

4.2.2 Campaign RN2

The objective of this campaign is to not rely directly on the detection and masking capabilities of the FTCOM layer as in campaign RN1, but to check the correctness at the application level. During approximately 64 hours more than 4.5 million white noise faults of duration 500 µs were injected on both channels. According to Equation 1, nearly 2.9 million effective faults have thus been injected. The correctness was validated at the application level by checking the continuous increment of the counters and not a single application failure was observed.

4.2.3 Campaign RN3

Erroneous frames that are syntactically correct but semantically incorrect (i.e. correctly built frames including e.g. value failures) will not result in e.g. Invalid or Incorrect frames and should be recovered or masked by the RDA algorithms. The objective of this campaign is to evaluate how well the RDA algorithms can detect and handle a message failure caused by one of the redundant nodes with or without replicated channels. Five different RDA functions operating on three data types are evaluated. One node in the cluster is deliberately forced to send non-expected data such as min and max values, NaN (Not a number) and Infinity floats as well as arbitrary float, boolean and integer values. All RDA functions behaved as expected but some cases needs to be discussed. A valid float value [12] can be changed into NaN or Infinity floats by e.g. a single bit-flip fault (e.g. if a 32-bit float value is 1.5 and bit 30 is altered). Arithmetic with NaN floats will produce more NaN floats and the error may propagate quickly in the software [13]. Thus, the result of the RDA algorithms, '*average*' and '*add*' operating on NaN floats are therefore also NaN and this should be considered during the choice of appropriate RDA functions. The RDA function '*valid strict*' demands that all three replicated values must be exact copies and performs therefore a roll-back recovery each time one or two faulty nodes are detected. Thus, this RDA function seems powerful to recover from transient or intermittent node errors, even such errors causing e.g. NaN floats.

5 Conclusions

An experimental evaluation of the fault-tolerant communication layer of the DECOS architecture is presented. The evaluation shows that built-in error detection and recovery mechanisms including different RDA functions are able to detect, mask or recover from errors both internal in a redundant node or on a replicated communication network. However, some erroneous data values such as NaN floats must be handled by the RDA algorithm before they are passed to the application level. This is automatically handled e.g. by the RDA function '*valid strict*' which detects and recovers from a transient or an intermittent node value failure by using the previously agreed value instead (roll-back recovery). A final observation is that a '*median*' RDA function could be useful as an alternative to the *average* function.

Acknowledgments. This work has been supported by the European IST project DECOS under project No. IST-511764. The authors also give their credit to Guillaume Oléron who has been involved in the experiments as a part of his studies.

References

1. Tils, V. von: Trends and challenges in automotive engineering, in Proceedings of the 18th International Symp. on Power Semiconductor Devices & IC's, (2006)
2. Broy, M: Automotive software and systems engineering, in Proc. of the third ACM and IEEE International Conference on Formal Methods and Models for Co-Design, (2005)
3. Kopetz, H. et al.: From a federated to an integrated architecture for dependable embedded real-time systems, Technical Report 22, Institut für Technische Informatik, Technische Universität Wien, (2004)
4. AUTOSAR - Automotive Open System Architecture, http://www.autosar.org, 2006
5. Schlager, M. et al.: "Encapsulating application subsystems using the DECOS Core OS", in Proceedings of the 25th International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Gdansk, Poland, September 27-29, (2006), 386–397
6. Obermaisser, R. and Peti, P: Realization of virtual networks in the DECOS integrated architecture, in Proceedings of the 20th Intern. Parallel and Distributed Processing Symposium, (2006)
7. Peti, P and Obermaisser, R: A diagnostic framework for integrated time-triggered architectures, in Proceedings of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, (2006)
8. Bauer, G. and Kopetz H: Transparent redundancy in the Time-Triggered Architecture, in Proceedings of the International Conference on Dependable Systems and Networks, (2000)
9. TTP-Build - The DECOS Cluster Design Tool for Layered TTP (Prototype), Edition 5.3.69a, TTTech Computertechnik AG, (2006)
10. TTX-Disturbance Node User Manual, Edition 1.0.4, TTTech Computertechnik AG, 2006.
11. Eriksson, H., et al.: "Towards a DECOS Fault Injection Platform for Time-Triggered Systems" in Proceedings of the 5th IEEE International Conference on Industrial Informatics, (2007)
12. ANSI/IEEE Std 754 – IEEE Standard for binary floating-point arithmetic, IEEE, (1985).
13. Vinter, J. et al.: Experimental dependability evaluation of a fail-bounded jet engine control system for unmanned aerial vehicles, in Proceedings of the International Conference on Dependable Systems and Networks, Japan, (2005), 666–671