Comparing Scheduling Policies for a Message Transient Error Recovery Server in a Time-Triggered Setting

Luis Marques*, Verónica Vasconcelos
*Instituto de Telecomunicações
Instituto Politécnico de Coimbra,
ISEC, DEE, Rua Pedro Nunes,
Coimbra, Portugal
lmarques@isec.pt, veronica@isec.pt

Paulo Pedreiras
Instituto de Telecomunicações,
DETI – Universidade de Aveiro
Aveiro, Portugal
pbrp@ua.pt

Luis Almeida
Instituto de Telecomunicações,
Fac. Engenharia – Univ. Porto
Porto, Portugal
lda@fe.up.pt

Abstract - **Distributed systems rely in communication networks, typically a bus, in order to exchange messages and fulfill their goals. However, message transmission is subject to interferences that ultimately can lead to message corruption. In systems where a high-reliability is sought, error recovery mechanisms can be deployed in order to give the required reliability level, and this can be done in the spatial or temporal domain. In the scope of the FTT paradigm, and applied to the FTT-CAN protocol, the authors have previously presented a time domain recovery method using message retransmissions controlled by a server. In this article we assess the impact of different scheduling policies for the server, presenting a qualitative evaluation of the alternatives, complemented by a simulation study, in order to verify their advantages and weak points.**

Keywords — transient errors; Time-Triggered; scheduling policies; server

I. INTRODUCTION

Nowadays, large pieces of equipment or machinery use several embedded nodes that can be physically distant and must cooperate to fulfill a given objective, composing what is known as a Distributed Embedded System (DES). These are present in cars, planes, trains, medical equipment or in factory plants, to name a few. A DES depends on exchanging information between its nodes, this being accomplished by an underlying communication network. Depending on the specific characteristics of the application, e.g. required bandwidth, criticality level or cost, the designer can choose one of the many technologies available today.

From the multitude of available network technologies, CAN [1] is still one of the most used networks, e.g. in the automotive industry, with more than 700 million nodes sold annually [2].

A common problem to all communication networks is the occurrence of errors in the transmission of messages that can compromise the correct system operation. The faults that can lead to message errors are unavoidable and can be due to electromagnetic interference, radiation, temperature variations, loose connectors, etc. Faults can be classified as permanent or transient, being the transient ones prevalent and the ones we are interested in this paper.

To obtain tolerance to communication errors, spatial redundancy, temporal redundancy or a combination of both can be used. In spatial redundancy extra hardware (nodes and buses) is needed in order to allow sending each message through different paths, increasing the system complexity and cost. As an advantage, this technique tolerates both permanent and transient faults. Temporal redundancy consists in sending a message and its replicas over the same path but in different time instants. This technique does not require additional hardware, but only tolerates transient faults.

This work focuses on Time-Triggered (TT) communication, i.e., in which the network transmits its messages in specified time instants, but using the Flexible Time-Triggered paradigm (FTT) that combines a TT approach with online traffic scheduling [3]. This feature allows using temporal redundancy with message retransmissions triggered by the occurrence of errors, being this in contrast to typical static TT systems, where the retransmissions are pre-planned and always occur, independently of the existence of errors, leading to poor bandwidth efficiency. Our retransmission mechanism is centrally controlled and the scheduling is done dynamically and jointly with the system messages [4]. Upon error detection, a server manages the retransmission of messages, interacting with the message scheduler, attempting to deliver messages affected by errors before its deadline, but with a bounded and predictable interference on the remaining messages.

Clearly, the server scheduling policy impacts both in the retransmission recovery time and on the way the server interferes with the other messages, implying that each server scheduling policy presents a tradeoff between these two parameters. In this paper we propose different policies for the server scheduling, present their main characteristics and analyze their implications.

2014 IEEE Emerging Technology and Factory Automation (ETFA)
978-1-4799-4845-1/14/$31.00

The paper is organized as follows. The related work is presented in section II, followed by the system description in section III, which includes a brief description of the FTT-CAN protocol, the assumed fault model and the server configuration. In section IV the alternative policies for the server are presented, and in section V the policies are assessed. Finally section VI concludes the paper.

II. Related Work

Fault tolerance in classical Time-Triggered systems, e.g. TTP/C [5] or FlexRay [6], when using temporal redundancy, implies the use of extra slots that are statically allocated and that are left unused when errors do not occur, leading to low bandwidth efficiency. Even if optimization techniques are used to minimize the number of used slots, like in [7], the extra bandwidth for a specified reliability level can reach more than 100% of the original bandwidth.

Another approach, a specific protocol proposed for CAN [8], allows the use of some slots defined for TT traffic by event-triggered traffic, when there are no TT messages to transmit. This is achieved by assigning lower priority CAN ID's to event-triggered messages. Nevertheless, this protocol suffers basically from the same problem as classic TT systems, because these slots are statically allocated, although with an improvement in average bandwidth efficiency.

III. System Description

In this section we briefly describe the FTT paradigm and FTT-CAN protocol, define the system, the assumed fault model, the server and parameter choices.

A. FTT-CAN Protocol

The FTT-CAN protocol [3] is an instantiation of the Flexible-Time Triggered paradigm on the CAN protocol [1]. In this paradigm, the global time is divided in Elementary Cycles (EC) with a predefined duration LEC. The EC is further divided in two disjoint windows, each reserved for a particular traffic class. The nodes transmit autonomously Event-Triggered traffic in the Asynchronous Window. The TT traffic is transmitted in the Synchronous Window (SW), when polled by a special message, the Trigger Message (TM), transmitted by the Master node. The TM also marks the beginning of the EC. The duration of the SW depends on the traffic scheduled for each EC but it is upper bounded by a configuration parameter – LSW - that must be set appropriately to provide the necessary guarantees of schedulability for the TT traffic.

The Master node is then responsible for scheduling the TT traffic online and can, in this way, implement any scheduling policy, e.g. Rate-Monotonic or Earliest Deadline First, thus being very flexible and independent of the underlying communication technology.

As said previously, the focus of this work is on the use of temporal redundancy to recover transient errors. Due to the high flexibility brought by the online scheduling characteristic available in the FTT paradigm, it is possible to implement distinct ways to recover transmission errors, namely by scheduling the server with different techniques, which is the topic of this paper.

B. System Definition and Fault Model

The system is composed by network nodes connected through a simplex CAN bus. One of the nodes, the Master, is responsible for scheduling all the time-triggered traffic and also for implementing the error recovery mechanism. The message set has *n* messages, each one characterized by a period T_i, transmission time (including maximum stuff bits) C_i, relative deadline D_i and offset O_i.

A simple but effective way to model the arrival of faults in the CAN bus is by using a Poisson model [9]. In this model the mean time between faults is λ and the faults arrive at random instants with an inter-arrival time that follows an exponential distribution. The probability of having *k* errors in the time interval *t* is given by equation 1.

$$P(X = k, t) = \frac{e^{-t\lambda}(t\lambda)^k}{k!} \qquad (1)$$

We also assume that λ is known and can be calculated from experimental data, obtained for instance from the Bit Error Rate (BER) values in [10], and that it is dependent on the environment where the embedded system is deployed, classified as aggressive, normal or benign environments. These values are essential to correctly define the server parameters, as described in the next section.

C. Server and its configuration parameters

A server is an entity that behaves like a periodic task and is characterized by a period T_S and capacity C_S. The way that the server is allowed to use its capacity and the time instants and amount of capacity replenishment is dependent on the server type [11]. The fault incidence in the server period, for a given probability, is obtained through equation 1, using the cumulative distribution function, allowing in this way to choose the necessary capacity in order to recover all errors, with a residual probability of non-recovery. This process was described in a previous paper [4]. The chosen server was a Deferrable Server (DS), mainly because of its simplicity and reactivity, since its capacity may be used at any time.

IV. Alternative Server Scheduling Policies

In our previous work [4], the server was granted with the highest priority (***Max_Pr***) in a fixed priorities setting. The rationale behind this option was scheduling the server immediately, to allow the retransmissions to occur as soon as possible, thus maximizing their likelihood of occurring before the messages deadline (Algorithm 1).

Algorithm 1: Server Scheduling with Maximum Priority

Inputs: M, Detected Errors, Server Retransmission Queue (SRQ), Ready Queue (RQ)
Output: Schedule for Next EC

1. For each message affected by Error or Omission
   Put it in the SRQ
   End
2. For each message in SRQ
   If server remaining capacity >= message size
   Move message from SRQ to the head of RQ and adjust server capacity
   End
   End
3. Build the EC Schedule
4. Put all retransmissions not dispatched back on SRQ and adjust server capacity
5. Return the EC Schedule

Firstly all detected errors in the current EC are added to the Server Retransmission Queue (SRQ). After that, messages in the SRQ are moved to the Ready Queue (RQ), provided that the server capacity is not exhausted (line 2). Under the **Max_Pr** scheme, retransmissions are added at the head of the RQ, giving them higher priority than all other messages. This way these messages are the first ones to be scheduled, minimizing their recovery time, as intended. Eventual message retransmissions that do not fit in the SW are pushed back to the SRQ (line 4). In both line 2 and 4 the server capacity is updated accordingly. In line 5 the EC schedule is returned.

A. Motivational Examples

Despite being intuitive, assigning the highest priority to the server is not the only option and may, in fact, not be the most sensible one. For example, consider that a message instance with a far deadline is affected by an error. Assigning the highest priority to the server would cause the retransmission to occur in the following EC, possibly delaying unnecessarily the transmission of messages with shorter deadlines.

In order to gain some insight into this problem, we show some exemplary scenarios, with and without errors, to observe the behavior of the recovery mechanism and the different types of interference produced by different server scheduling policies. The message set used in the following scenarios is detailed in TABLE I, where message deadlines are equal to periods and priorities are assigned using the Rate Monotonic policy. The transmission time of all messages is 0.2 time units, resulting in a utilization of 55%.

Figure 1 shows the timelines corresponding to the maximum priority policy, with and without errors (bottom and top, respectively). The numbers below the line indicate the message activation instants. This example illustrates the scenario described above, in which an error in a message with a longer period (m6) causes a delay of one EC on a message with a shorter period (m5). This kind of interference is unnecessary because message m6 could be retransmitted in the following EC, without violating its deadline.

TABLE I : MESSAGE SET DEFINITION

Msg n	*T*	*Msg n*	*T*
1	2	4	2
2	2	5	2
3	2	6	4

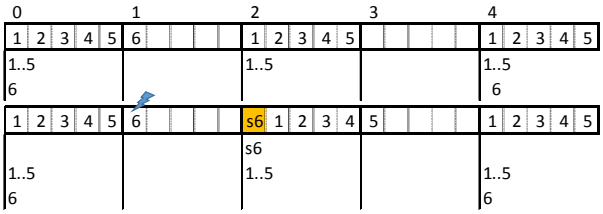

Fig. 1. Max_Pr scheduling policy illustration; top: no errors; bottom: with errors

An alternative policy, which prevents the priority inversion scenario depicted above, consists in preserving the message's priority during retransmissions (***Same_Pr).*** In terms of implementation, such policy is easily accomplished, since it consists in handling retransmissions in the same way as regular messages. This requires changing line 2 of Algorithm 1, only, which becomes "Insert message retransmission in the proper position of the RQ, according to its priority". Since, in this case, the messages in the SRQ are scheduled together with the remaining messages, following a system-wide fixed-priority scheduling policy, message retransmissions do not interfere with higher priority messages, as desired.

On the other hand, the error recovery latency depends on the priority of the message affected by the error and on the instantaneous system load. Consequently, this scheduling policy has a behavior that is dual of the ***Max_Pr*** scheme. For this reason it may lead to unnecessary deadline misses, as illustrated in Figure 2, where the message set is based on TABLE I, except in what concerns the period of m6, which is reduced from 4 to 3 time units.

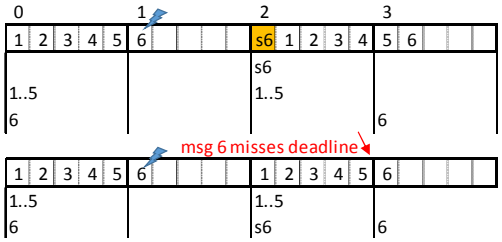

Fig. 2. Max_Pr (top) vs Same_Pr (bottom) scheduling policies illustration

In this case, the message retransmission was unnecessarily delayed, leading to a deadline miss (Figure 2, bottom). Conversely, if the ***Max_Pr*** scheme was used, no deadline would be missed (Figure 2, top).

Reasoning about the scenarios shown above, it can be concluded that both scheduling schemes have a common problem: the scheduling decisions regarding message retransmissions do not take into account the dynamic system state and thus are suboptimal. This observation suggests that a possible way of improving the system performance may be achieved by using the message slack as a scheduling decision parameter. One simple way of obtaining this behavior consists in using the ***Same_Pr*** scheme by default, in order to prevent priority inversions as much as possible, and assigning the maximum priority to the server whenever the message at the head of its retransmission queue has a slack of one EC, in order to reduce the number of deadline misses affecting retransmissions of low-priority messages. This scheme, designated ***Same_Pr_DMP***, is described in Algorithm 2. Note that Algorithm 2 shows only the differences with respect to Algorithm 1.

Algorithm 2: Server Scheduling with Same Priority with Deadline Miss Protection

```
2. For each message in SRQ
   If server capacity >= message size
      If absolute deadline > Next EC
         Insert message in the RQ, with normal
         priority
      else
         Insert message at the head of the RQ
      End
   End
End
```

Though simple and easy to implement, as it requires only one additional test with respect to the ***Same_Pr*** scheme, the ***Same_Pr_DMP*** policy is still suboptimal as it neglects the message's slack until a border condition is met, namely the last EC before the deadline.

Conversely, the Earliest-Deadline First [12] policy is a scheduling policy that takes into account the dynamic state of the system, giving priority to the tasks with more urgent deadlines. This is the feature that makes EDF optimal with respect to meeting deadlines, while allowing higher utilization factors than fixed-priority schemes. This feature is particularly well suited for the error server scheduling on FTT-CAN. As mentioned before, errors may affect both messages with short or long deadlines and, due to their random nature, it is not possible to forecast which kind of requests will be submitted to the server, thus optimum decisions cannot be fixed nor taken offline.

These observations, together with related work that proposes hierarchical scheduling using different scheduling policies at different levels in the scheduling process [13], led to the proposal of the ***EDF*** scheduling policy (for the server only), in which the deadlines of the message retransmissions are compared with the ones of normal messages, being scheduled as soon as their deadline is shorter than the one of any other of the normal ready messages.

In this way, retransmissions have a minimum interference on the other messages when their deadlines are farther away, but gain importance gradually, i.e., increase their priority, as their deadlines approach, as desired. The operation of the ***EDF*** policy is described in Algorithm 3.

As it will be shown in Section V, the ***EDF*** scheduling policy delivers the best results in terms of indirect interference, outperforming the other policies in many cases, as expected. However, it incurs in a relatively high overhead, since it requires that both the retransmission queue and the ready queue are sorted by deadline and insertions on the ready queue must also be made according to the deadline. Considering that CAN is often used in embedded systems based on extremely resource-constrained hardware, such overhead can be prohibitive. Therefore the cost-benefit relation must be carefully evaluated.

Figure 3 presents an example of the different behavior in terms of recovery latency and indirect interference of the four proposed server scheduling policies, showing a scenario in which only the EDF scheme can recover a message without causing any deadline miss.

LEC = 5 ms; LSW = 4 ms		
n	T	Ci
1	10	0.8
2	10	0.8
3	10	0.8
4	10	0.8
5	10	0.8
6	15	0.8
7	15	0.8
8	25	0.8
9	25	0.8
10	25	0.8
11	25	0.8
12	25	0.8
13	25	0.8

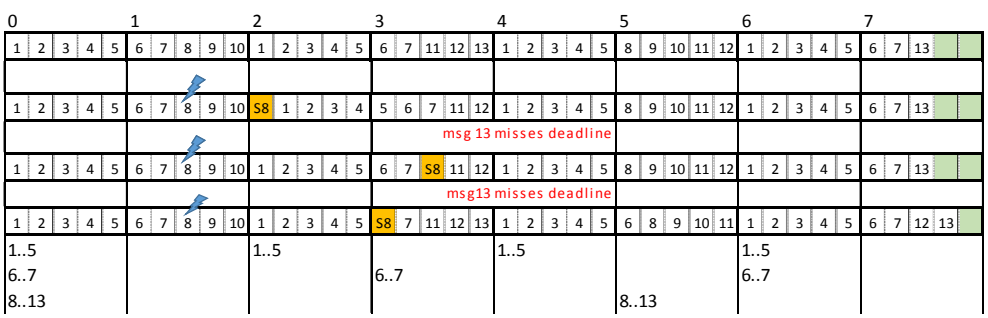

Fig. 3. Applying different server scheduling policies. Top – no errors; Second – Max_Pr; Third – Same_Pr or Same_P_DMP; Fourth – EDF

Algorithm 3: Server Scheduling with EDF Priority

```
2.1. Build normal schedule of next EC – RQ - using normal
     scheduler
2.2. Sort SRQ by deadline
2.3. Sort RQ by deadline
2.4. For each message in SRQ
        If server capacity >= message size
            Insert message in the RQ,
            sorted by deadline
        End
   End
```

V. RESULTS

In this section we present a set of simulation results that aim at evaluating the performance and correctness of the proposed algorithms. To this end, in addition to the more obvious numeric results regarding the number of recovered and non-recovered errors, we also include numerical results that allow assessing the response-time and internal effects of the diverse server scheduling policies.

The simulations were carried out on the FTT-CAN simulator presented in [4], which was extended with the server scheduling policies presented in this paper.

To evaluate the behavior of each server scheduling policy, we have used message sets obtained from the literature [14], [15] and also from an electric vehicle prototype [16]. The results obtained with the various message sets are analogous so, due to space limitations, only the results of the first message set are presented.

The message set is defined in Table II, where ID is the CAN message identifier (lower number means higher priority), T is the period in number of ECs and DLC the message size, in bytes.

TABLE II. MESSAGE SET UPDATED_SAE

ID	T	DLC	ID	T	DLC	ID	T	DLC
1	20	1	13	3	1	25	5	2
2	2	2	14	3	4	26	5	2
3	2	1	15	3	4	27	5	4
4	2	2	16	3	4	28	5	5
5	2	1	17	4	1	29	5	3
6	2	2	18	4	2	30	20	1
7	2	1	19	4	6	31	40	4
8	2	1	20	4	2	32	40	1
9	3	1	21	4	3	33	40	1
10	3	1	22	4	2	34	400	3
11	3	1	23	5	2	35	400	1
12	3	1	24	5	2	36	400	1

For each configuration, we start with the minimum LSW that allows the transmission of messages within their deadlines, without considering bus faults. Afterwards we generate the error pattern, using a BER 100 times greater (λ = 10 faults/s) than the scenario experimentally encountered in aggressive environments [10], in order to stress the error recovery mechanism. Then the system is simulated for each one of the alternative policies, retransmissions included, and the corresponding numerical values stored. Finally, the LSW size is increased and the whole process is repeated.

A. Recovered Errors and Interference

Table III presents the deadline misses suffered by messages affected by errors, for the diverse policies. The presented results correspond to the average value of ten runs of the simulator, each one done for 2 million ECs, where the BER used in the fault generator is rather high in order to get a significant number of errors per simulation.

The first line shows the total number of deadline misses, while the second one shows the number of errors that affected messages that had no slack, and thus that weren't recoverable by any of the mechanisms.

We can see that for ***Max_Pr*** and ***EDF*** policies all misses correspond to unrecoverable messages, i.e. messages that suffer an error in the EC in which they have their deadline. For the ***Same_Pr*** policy, as the server inherits the original message priority, retransmissions can be delayed by several cycles and eventually lose their deadlines, as shown in the scenario depicted in Figure 4. This situation happened in fact, as attested by the relatively large number of deadline misses. As expected, the ***Same_Pr_DMP*** policy significantly attenuates this problem, though not eliminating it.

TABLE III. DEADLINE MISSES DUE TO DIRECT INTERFERENCE (LSW=70%)

Unrecovered by the Server				
	Max_Pr	*Same_Pr*	*Same_Pr_DMP*	*EDF*
Total misses	29	510	101	29
Unrecoverable misses	29	31	31	29

Table IV shows the number of deadline misses suffered by messages not affected by errors, thus due to the server execution - indirect interference. As expected, the ***EDF*** policy has the best performance in this criterion, since it only schedules the retransmissions when they are more urgent (i.e. have shorter deadlines) than the original messages.

It should be remarked that the presented results depend strongly on the bus load. For instance, for LSW=72.5%, in this message set, there is no difference between the diverse policies, because the server execution does not cause a significant interference on the original messages.

TABLE IV. DEADLINE MISSES DUE TO INDIRECT INTERFERENCE (LSW=70%)

Deadline misses in original messages			
Max_Pr	*Same_Pr*	*Same_Pr_DMP*	*EDF*
1133	1133	1133	1093

B. Server Recovery Time

Table V shows the average server response times for each one of the server scheduling policies. We can see that the ***Max_Pr*** policy shows the lowest value, always equal to one EC. There are a few exceptions, too small to cause visible effects in Table 5, which correspond to scenarios in which errors affect message retransmissions. On the other hand, the **Same_Pr/Same_Pr_DMP** policies show the worst performance, being ***EDF*** a good compromise between these two extremes. We can also observe that the 3 initial schemes have a similar behavior regarding the high priority messages, which fit in the LSW even when the server executes, as expected. On the other hand, messages that have lower priority than these ones see their recovery time increasingly degrading for ***EDF***, ***Same_Pr_DMP*** and ***Same_Pr***, by this order. This behavior is also expected because, for these policies, the recovery can be delayed, something that does not happen with ***Max_Pr***.

Table V. SERVER AVERAGE RESPONSE TIME

msg nº	1	2..18	19	20	21	22	23	24
Max_Pr	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Same_Pr	1.000	1.000	1.001	1.001	1.070	1.355	1.558	1.478
Same_Pr_DMP	1.000	1.000	1.001	1.001	1.070	1.355	1.558	1.478
EDF	1.655	1.000	1.000	1.000	1.000	1.000	1.179	1.166

VI. CONCLUSIONS

In this paper we have explored alternative scheduling policies applied to a temporal redundancy scheme, in the scope of the FTT-paradigm, using retransmission servers. After the problem definition, we presented a few motivational scenarios, which show that different server scheduling policies have a real impact in the recovery time and in the interference caused by the server in messages not affected by errors.

Then, the algorithms that correspond to each server alternative were presented and their impact in terms of complexity was qualitatively evaluated. Afterwards, we presented a numerical evaluation of the proposed algorithms, obtained via simulation, concerning the efficiency of recovering erroneous transmissions. The obtained results show that the relative performance of the diverse algorithms depends on the particular scenarios and criteria. For instance, if the predominant criterion is minimizing the recovery time, the best scheduling policy is ***Max_Pr*** at the expense of a higher global number of messages with missed deadlines than ***EDF***. The ***Same_Pr*** policy showed the worst behavior, which can be significantly improved by using deadline miss protection. Ultimately, it is the designer that must evaluate carefully the pros and cons of each scheduling server policy based on the application criteria.

As future work, we will carry out an analytical evaluation of each policy, as well as a more detailed characterization of their associated overhead, including optimized implementation techniques, e.g. the ones presented in [17].

ACKNOWLEDGMENTS

This work was partially supported by the Portuguese Government through FCT - Fundação para a Ciência e a Tecnologia in the scope of project Serv-CPS -PTDC/EEA-AUT/122362/2010 and grant CodeStream PTDC/EEI-TEL/3006/2012.

REFERENCES

[1] Controller Area Network (CAN) specification - version 2.0., Bosch GmbH, 1991

[2] CAN in Automation site, http://www.can-cia.org

[3] L. Almeida, P. Pedreiras and J.A. Fonseca, "The FTT-CAN protocol: Why and how", IEEE Transactions on Industrial Electronics, 49 (6), December 2002

[4] L. Marques, V. Vasconcelos, P. Pedreiras and L. Almeida, "Error Recovery in Time-Triggered Communication Systems Using Servers", 8th IEEE International Symposium on Industrial Embedded Systems, June 2013, Porto, Portugal.

[5] H. Kopetz and G. Bauer, "The Time Triggered Architecture", Proceedings of the IEEE, 91(1), 2003

[6] FlexRay Consortium, "FlexRay Communications System - Protocol Specication Version 2.1 Revision A," http://www.flexray.com/, December 2005

[7] B. Tanasa, U. Bordoloi, P. Eles and Z. Peng, "Scheduling for fault-tolerant communication on the static segment of FlexRay", Proceedings of 31st IEEE Real-Time Systems Symposium, 2010.

[8] J. Kaiser, B. Cristiano and C. Mitidieri. "COSMIC: A real-time event-based middleware for the CAN-bus", Journal of Systems and Software 77.1, 2005

[9] I. Broster, A. Burns, G. Rodríguez-Navas, "Comparing Real-time Communication under Electromagnetic Interference", Proceedings of the 12th Euromicro Conference on Real-Time Systems (ECRTS'04)

[10] J. Ferreira, A. Oliveira, P. Fonseca and J.A. Fonseca, "An Experiment to Assess Bit Error Rate in CAN", Proceedings of 3rd International Workshop of Real-Time Networks , 2004

[11] G. C. Buttazzo, *Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications*, Springer, 2011

[12] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", J. ACM 20, 1 (January 1973), 46-61

[13] M. G. Harbour and JC Palencia, "Response time analysis for tasks scheduled under EDF within fixed priorities", 24th IEEE Real-Time Systems Symposium, 2003.

[14] U. Mohammad and N. Al-Holou, "Development of An Automotive Communication Benchmark", Canadian Journal on Electrical and Electronics Engineering Vol. 1, No. 5, August 2010

[15] P. Castelpietra, Y.-Q. Song, F. Simonot-Lion and O. Cayrol, "Performance evaluation of a multiple networked in-vehicle embedded architecture", Proceedings of IEEE International Workshop on Factory Communication Systems, 2000

[16] F. Santos, J. Trovão, A. Marques, P. Pedreiras, J. Ferreira, L. Almeida, M. Santos, "A modular control architecture for a small electric vehicle", 11th IEEE International Conference on Emerging Technologies and Factory Automation, Prague, Czech Republic, 2006

[17] M. Short, "Improved task management techniques for enforcing EDF scheduling on recurring task sets," Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium, Stockholm, Sweden, 2010.