Back to Table of Contents

Error Recovery in Time-Triggered Communication Systems Using Servers

Luis Marques, Verónica Vasconcelos
Dep Eng Electrotécnica
ISEC – Inst. Politécnico de Coimbra
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*— In communication systems, transient faults will eventually occur. Thus, some mechanism is necessary to handle them and achieve appropriate levels of reliability, particularly in safety-critical systems. One possibility is to rely on temporal redundancy, i.e., using message retransmissions. General requirements for such a mechanism would include a parsimonious use of extra bandwidth while guaranteeing the schedulability of the message set. In this paper we propose using on-line traffic scheduling together with scheduling servers to recover message errors in time-triggered systems on Controller Area Network (CAN), taking advantage of the Flexible Time-Triggered CAN protocol. This novel mechanism is shown to offer a desired error recovery latency using much less extra bandwidth than typical approaches used in time-triggered systems. In this paper we present this novel error recovery mechanism, including a thorough characterization as well as configuration guidelines, namely concerning how to choose the server parameters (type, period and capacity). The correctness of the proposed approach and its superior performance are validated with simulation using several communication benchmarks available in the literature.**

Keywords—time-triggered communication; traffic scheduling; scheduling servers; Control Area Network; error recovery

I. INTRODUCTION

Bus networks are still the prevalent way of connecting network nodes in Distributed Embedded Systems (DES) [1]. Many of the applications supported by DES architectures (e.g., vehicular, medical devices) have strict timeliness and dependability requirements, thus the networks must provide reliable and timely communication between nodes. Since its introduction in the 80's, CAN [2] has gained wide acceptation, being one of the most used protocols in several areas.

One important application field of DES is the automotive domain, namely in Drive-by-wire systems [3]. These systems are obviously safety-critical, since a system failure could cause the death or serious injury to people. Therefore, these systems must possess high levels of reliability and integrity. These levels can be achieved using fault tolerance techniques, e.g. using replicated nodes and buses, as well as retransmission mechanisms.

Faults in any electronic system can be classified as permanent or transient, depending on their persistence. Transient faults are temporary by nature and are due mainly to Electromagnetic Interference (EMI), radiation or temperature variation for instance. On the other hand permanent faults are persistent and result from physical defects in system parts and last until repair or replacement. The incidence of transient faults is far greater that the permanent ones, with a ratio of transient to permanent faults bigger than 100 [4]. Therefore, for this kind of systems, the prompt detection of transient transmission errors can boost significantly the system reliability.

Communication errors can be handled either on the temporal or in the spatial domain. Temporal domain techniques imply the transmission of messages and replicas in distinct time instants. The transmissions can be done systematically or be triggered by error occurrence. On the other hand, spatial domain techniques require the existence of multiple communication channels, each one transmitting a message replica or copy. These methodologies are orthogonal and may be used simultaneously in the same system.

This paper focuses on the use of temporal redundancy, only, to handle communication errors in time-triggered communication systems. We claim that static traffic scheduling, commonly used by current time-triggered protocols, is not bandwidth-efficient and we propose an online traffic scheduling approach to improve such situation. In particular we make use of the unique flexibility features of FTT-CAN (Flexible Time-Triggered communication on CAN) [5], namely the capability to schedule the time-triggered traffic online, to reschedule omitted messages whenever necessary, and with full control over the respective scheduling. This approach not only brings important gains in bandwidth efficiency with respect to the conventional offline-scheduled time-triggered communication protocols but also gives the system designer more freedom in controlling the compromise between timeliness and reliability inherent to the scheduling of retransmissions.

The main contribution of this paper is the use of a periodic server to handle all the retransmissions. We show how to configure such server so that it can serve all the retransmission requests within the messages deadlines and we validate this mechanism with extensive simulation using benchmarks reported in the literature as well as one from a small electrical vehicle prototype. The bandwidth needed to recover all errors in an aggressive environment is two orders of magnitude lower

978-1-4799-0658-1/13/$31.00 ©2013 IEEE

than the one required using static off-line scheduling of retransmissions.

The paper is organized as follows. The following section discusses some related work while Section III briefly introduces the FTT-CAN protocol. Section IV presents the fault model while Section V presents the system model and the recovery server configuration method. Section VI shows the experimental validation of our approach and Section VII concludes the paper and refers to future work.

II. Related Work

Communication protocols can be classified according with different criteria, one of them being the triggering of message transmissions. In event-triggered protocols, e.g. CAN and Ethernet, message transmissions are triggered at arbitrary instants, by express application request. In time-triggered protocols, e.g., TTP/C [6], messages are transmitted at predefined time instants. There are also mixed event/time triggered protocols, e.g. TT-CAN [7], FlexRay [8], FTT-CAN and FTT-SE [9], in which both types of communications coexist. In the automotive industry the trend is using the mixed paradigm, with the FlexRay protocol being pointed out as a potential de facto standard for high speed communications in cars [10].

For safety-critical applications, the use of networks with a time-triggered phase is usually preferred, due to the predictability of the time-triggered traffic pattern that permits a prompt detection of message omissions. Nevertheless, despite the promptness of detection, the error correction is more complicated and imposes an undesired compromise between bandwidth efficiency, timeliness and reliability related to when and how it is carried out.

Depending on the system type, several temporal redundancy schemes can be used to guarantee the message delivery in the presence of transient faults. The periodic sampling and transmission associated with real-time monitoring and feedback control already implies a train of semantically equivalent messages, i.e., referring to the same real-time entity. When one of such messages is lost, a trivial recovery scheme consists on waiting for the following transmission one period later. This is a simple procedure but with long recovery latency, typically penalizing the performance of the application, e.g., the quality of control in a feedback control loop. Such impact can be reduced by using over-sampling, a technique frequently used in industry in which a higher sampling and transmission rate is used. In this case, the recovery latency is shortened in the same way as the period is reduced at the cost of a corresponding increase in the use of bandwidth. This extra bandwidth is wasted whenever there are no errors, which is the most frequent condition in a properly designed system.

An alternative technique consists on sending multiple copies of each message instance (replicas), over time, via the same bus. The problem, however, is that classic communication time-triggered protocols are based on static offline built schedules, from TTP/C to TT-CAN, FlexRay or Profinet-IRT and Ethernet Powerlink [11]. Thus, message retransmissions must be decided offline, without factual knowledge about the concrete existence of errors in each particular message instance. Since errors are infrequent, chances are that the offline-decided message retransmissions are, most of the times, useless or even omitted, depending on the nature of the protocol, causing a bandwidth waste that penalizes the efficiency of the bus utilization. Recent work proposes optimizing the time-triggered schedule in FlexRay to reserve the minimum static slots for retransmissions so that a reliability goal is met [12]. Despite the improvement in recovery latency and impact on bus bandwidth with respect to the case of simply adding slots for retransmissions, this mechanism is still static and scheduled off-line, thus suffering from the same problem of poor bandwidth efficiency utilization.

Another alternative could be using the multiple operational modes that some protocols, e.g. TTP/C, offer, and switch between them online. In this case, different modes with different slot arrangements for different error scenarios could bring in a benefit in terms of bandwidth use but in reality the number of modes is always too limited to allow representing all error patterns of interest. Thus, this approach is not effective for handling transient faults.

Finally, some protocols have been proposed for CAN that allow the coexistence of time-triggered slots with event-triggered traffic in which the latter has lower priority and can share the bandwidth left free by the former [13]. In this case, when there are no errors the bus bandwidth is still usable by the event-triggered traffic, thus alleviating the negative impact on bandwidth efficiency of the previous time-triggered approaches. However, they typically use the native retransmissions mechanism of CAN. This introduces two negative aspects: firstly they work only on CAN and secondly, the scheduling of the retransmissions is constrained to that mechanism and cannot be consistently integrated with the scheduling of the time-triggered traffic. FTT-CAN allows such bandwidth efficient combination of time and event-triggered traffic while suffering from none of those negative aspects. In fact, the online scheduling of retransmissions can follow any desired policy and it is intrinsic to the FTT paradigm, thus independent of CAN and available in any FTT protocol.

In this paper we will use the omissions detection mechanism of FTT-CAN proposed in [14], which bears some resemblance with ServerCAN [15] for event-triggered communication, and schedule the retransmissions through a periodic server that is then integrated with the time-triggered traffic.

III. FTT-CAN brief presentation

The FTT-CAN communication protocol [5] is the implementation of the FTT paradigm on the CAN protocol. This protocol uses the dual-phase Elementary Cycle (EC) concept in order to combine time and event-triggered communication with temporal isolation. The EC is composed by two windows, each one conveying the corresponding communication traffic – asynchronous window (AW) for the event-triggered messages and the synchronous window (SW) for the time-triggered one. Moreover, the time-triggered traffic is scheduled online by the master node using any scheduling

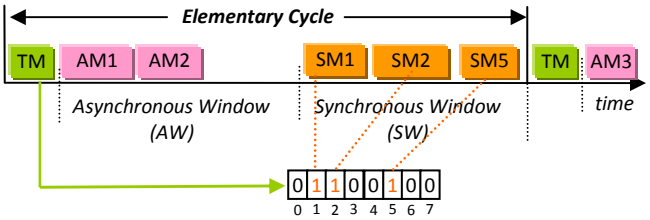

Fig. 1. Elementary Cycle in FTT-CAN (the TM payload defines which synchronous messages should be transmitted in current EC).

policy. Since scheduling is carried out dynamically, it is possible to change the communication requirements at runtime, adapting to changing environments, a property called operational flexibility. All those changes are subject to a schedulability test, thus guaranteeing the timeliness of all admitted messages.

The FTT-CAN protocol uses a master/multi-slave transmission control scheme, to reduce the overhead normally associated with normal master-slave protocols. According with this scheme, the TM triggers the transmission of several messages on one or more slave nodes. As can be seen in Figure 1, the EC starts with the TM that possesses the scheduling information for the time-triggered traffic in that EC and also defines the windows durations in order to force isolation between the AW and the SW. The duration of the SW may change from EC to EC, depending on the traffic scheduled for it, being upper limited to give minimum service quality to event-triggered traffic.

The FTT-CAN protocol has a set of characteristics that clearly distinguish it from other protocols currently used in DES. These are the online scheduling and message triggering done centrally by the master node, with any given scheduling policy, in combination with the support to asynchronous and synchronous traffic with temporal isolation.

The FTT-CAN has a centralized architecture where the master node controls message transmission activities. This aspect facilitates the use of elaborated scheduling schemes and allows for prompt updates to the traffic schedules, on an EC-by-EC basis. On the other hand introduces a single point of failure, the master, because if this node, for some reason, cannot send the TM all the communication activities end and the system suffers a serious failure, with possible catastrophic consequences. This obstacle can be overcome by the use of master replication [16], that can cope with master failures with very short replacement latency. Another single point of failure, but also common to other DES, is the bus permanent failure. This issue can be resolved in the FTT-CAN scope using replicated buses, as in [17].

Operational flexibility, meaning the capacity to cope with evolving scenarios, where is possible to add, remove or update synchronous messages, is made possible by the existence of an online scheduler. An admission control module, implemented in the master node, verifies if new messages can be added (or updated) while maintaining system schedulability. If not, the message is rejected by this admission control service, otherwise the correct system operation would be at risk.

By default the CAN controller retransmits messages in case of errors. In order to guarantee that all scheduled messages for the current EC fit inside the SW, in the current implementations of FTT-CAN this mechanism is disabled. This poses a problem with transient message errors in the synchronous window, diminishing the reliability of the system when compared with CAN. This can be overcome using an error signaling protocol or by using specific controllers, e.g. majorCAN [18].

IV. Fault Model

In this work we consider a distributed embedded system based on a simplex CAN bus in which messages can be corrupted by sporadic bit errors, e.g., due to EMI. The arrival rate of the errors, i.e., the Bit-Error Rate (BER), is a function of the environment and it is known.

The occurrence of faults is assumed to follow a Poisson process with average fault arrival rate equal to λ, where the mean depends on the environment. Other more complex arrival processes that can, for example, model bursts are left for future work. We also consider that one fault per EC is a reasonable assumption since the EC duration is typically small compared to $1/\lambda$, being rather unlikely that two or more faults occur in such a short time period. For instance, if we assume a BER of the order of 10^{-7} and a baud rate of 500kbps, an aggressive assumption according to [19], with an EC of 2.5ms, the probability of having more than one fault per EC is of the order of 10^{-8} which we consider low enough to be neglected.

The work in [19] shows an experimental assessment of the BER in CAN in different environments that are, then, classified as *benign*, *normal* and *aggressive* (Table I). Here we complement this classification with one further category, called *ultra-aggressive*, which we will use to stress the mechanisms we propose in this paper with a BER that is typically higher than found in practice.

Since we are focusing on transient faults and temporal redundancy techniques, we do not consider bus partitions. These can be handled with spatial redundancy techniques, as in [17], that are orthogonal to the current work. Similarly, we do not consider Master crashes or even TM omissions, as these can also be handled with the mechanisms presented in [16]. Moreover, we also consider a few simplifying assumptions that we expect to drop in future work, namely that all nodes are fail silent, i.e., transmit correctly or nothing at all, and that all faults are symmetric, thus all nodes detect all faults in the same way. This last assumption also implies that any node in the network can listen to and record all the messages in the bus. Recalling that the TM contains the schedule for the EC, it becomes clear that any node can detect a synchronous message omission.

TABLE I. BER in CAN – Experimental Values

Environment	***BER***
Benign	3.0×10^{-11}
Normal	3.1×10^{-9}
Aggressive	2.6×10^{-7}
Ultra-aggressive	$5{,}0 \times 10^{-6}$

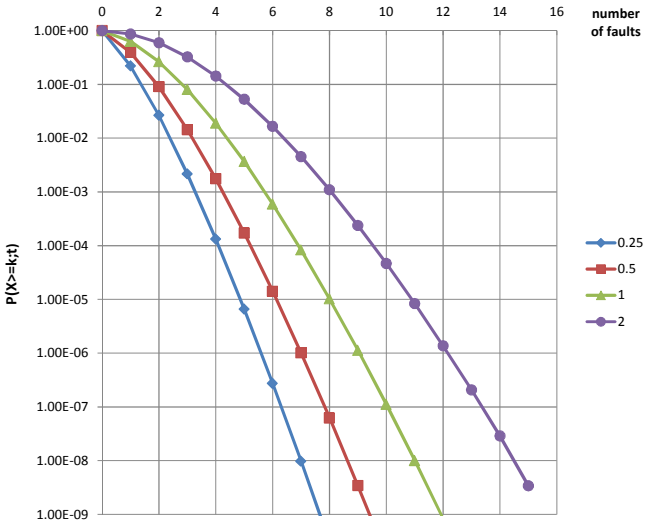

Fig. 2. Probability of N or more faults in intervals 0.25/λ, 0.5/λ, 1/λ and 2/λ.

Note that we consider message errors (or omissions) in the synchronous window, only. This is a situation in which a certain synchronous message should arrive in a particular EC, according to the EC-schedule in the respective TM, but it is not received by the Master, or possibly another node, depending on the method used. Since a message error or omission can make the system inconsistent for a period of time, which can correspond to several ECs, we must handle these situations promptly.

A. *Fault incidence*

A central point in our work is the knowledge of the fault incidence rate in a specified period, i.e., how many faults happen in such period and with which probability**.** As known, the Poisson process is characterized by an average arrival rate λ, and we can calculate the probability of arrival of *k* instances in a defined interval *t*, using equation (1).

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

We can use this equation in order to obtain the probability of arrival of *k* or more instances, P(X≥k, t), by summing the terms in equation (1).

Figure 2 shows, as an example, the probability of different numbers of faults in four different intervals relative to the average interval between faults (1/λ), namely 1/4, 1/2, 1 and 2. This probability is an important requirement that will guide the design of our fault handling mechanism. For example, consider that the desired probability of not recovering from a transient fault is 10^{-8}. This means that, considering an average arrival rate of 1 fault per second, our mechanism must have the capacity to recover 11 faults in that interval.

V. System Model and Servers Configuration

We consider a system with a message set M as in expression (2), with *n* messages, in which each message is periodic with a maximum transmission time C_i, a period T_i, a deadline $D_i \leq T_i$ and an offset O_i.

$$M = \{m_i(C_i, T_i, D_i, O_i,), i = 1..n\} \quad (2)$$

Any of these messages can be subject to errors and we wish to recover such affected messages with retransmission as long as it is within the respective deadlines. The main proposal in this work is to handle such retransmissions using a periodic server S as in expression (3), the *error server*, in which C_S is the server capacity, T_S its replenishment period and O_S its offset. The message set M, together with server S, form an extended set M_S that must be schedulable.

$$M_S = M \cup S(C_S, T_S, O_S) \quad (3)$$

Currently, for simplicity of implementation, we use a deferrable server which, despite a small penalty in schedulability, presents a good response time, as needed to serve the retransmission requests within their deadlines. Moreover, it is given highest priority in the system.

The schedulability of the extended M_S set can be assessed using any of the methods existing in the literature [20] from fast utilization-based tests to accurate response-time tests. Note that the bandwidth of the error server is shared among all messages that need being recovered, which further contributes to the system bandwidth efficiency. However, the main reason why this approach is highly efficient is because it decouples the response time needed to serve retransmissions in time from the replenishment period that is computed based on the fault incidence rate. Typical retransmission slots in static schedules work as polling servers, which couple response time with replenishment period. This leads to periods that are much shorter than actually needed by the fault incidence rate in order to meet the response time requirements. As shown in the following section, this difference in bandwidth requirements to recover a given set of errors can be as high as two orders of magnitude, or even higher.

The architecture of our system has been presented in [14], being all the process represented in Figure 3.. Essentially, the FTT-CAN Master is enhanced with an omissions detection unit that reports missing synchronous messages at the end of the respective window and just before invoking the scheduler for the next EC. These missing messages are then inserted in the error server queue for retransmission. When the Master scheduler is invoked at the end of the EC it takes the pending retransmissions into account when building the schedule for the following ECs. If the scheduling criteria so dictates, a retransmission can occur as early as in the following EC, thus with 1EC minimum latency.

A. *Error server design*

We start by computing the server replenishment period T_S based on the average interval between consecutive errors (1/λ), expressed as integer number of ECs, which is obtained

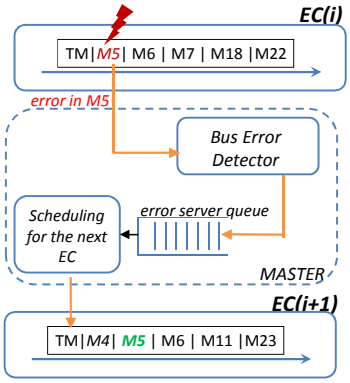

Fig. 3. Error detection and recovery process, using a server

by means of expression (5) considering the BER, bus bit-rate and EC length (LEC).

$$T_S = \alpha \left\lceil \frac{1}{\frac{BER * bit\_rate}{LEC}} \right\rceil \quad (5)$$

The bandwidth assigned to the server depends on both T_S and C_S. Though the same bandwidth can be achieved by different (T_S,C_S) pairs, the server behavior, namely in what regards interference on lower-priority messages and consequently on the global system schedulability, is quite different. Parameter α, in equation 5, allows adjusting the period in order to control, if needed, the impact on the schedulability of the extended set M_S.

On the other hand, C_S is determined through the Poisson curves in Figure 2. We start with a desired residual probability of not being able to recover from errors and we determine how many errors can occur in that interval, for that probability, using the curve corresponding to ($\alpha * 1/\lambda$). Then we compute C_S so that the server can handle that number of messages with maximum length. Recalling the same example as stated below Figure 2, for a residual probability of 10^{-8} with $T_S=1/\lambda$, the server needs to handle 11 messages, i.e., 11 retransmissions, and thus $C_S=11*\max_i(C_i)$. This simple method, as shown through simulations in the following section, is rather effective.

VI. Simulation and Results

This section presents the simulator that was developed to assess the error-recovery mechanism presented in this paper, as well as simulation results obtained with several benchmarks and BER values described in the literature.

A. Error simulator for FTT-CAN

Simulation is a well-known and widely used technique for assessing and validating an immense variety of engineering problems. For the particular case of error recovery protocols, simulation is invaluable, given the difficulty of reproducing error scenarios that occur rarely, thus making experiments with real hardware extremely hard, cumbersome and long.

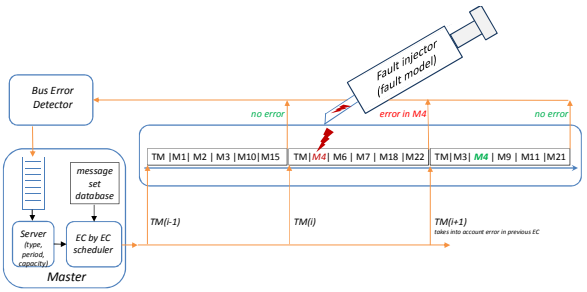

Fig. 4. Simulator Block Diagram

***Inputs**: file with network configuration, server parameters, BER and message set definition*
***Outputs**: text file with collected statistics*

1. Load network parameters, BER and message set
2. Initialize internal data structures
3. Construct the error vector
4. **For** EC number 1 to Max_Cycle
   a. Schedule next EC
   b. Bus processing
   c. Update server
   d. Collect statistical data
5. Write all statistics gathered to a text file

Algorithm 1: Simulator algorithm

For this reason, we developed a simulator specifically tailored for testing the error-recovery mechanism proposed in this paper. Figure 4 presents the simulator block diagram, which operates according to Algorithm 1.

At step one, the simulator reads the network configuration parameters, such as CAN bitrate, EC length and LSW (Length of SW). Then, it loads the message set, defined by the number of messages, their periods, offsets, sizes and priorities. All this data is loaded from a text file.

At step two, the simulator internal structures are initialized.

At step three it generates the error vector. The errors are generated according to the Poisson distribution, with average λ, corresponding to the defined BER. The error vector contains the index of the ECs in which faults occur, as well as the time instant, inside each ECs, where the faults actually occur.

At step four the traffic is scheduled and processed. The traffic is scheduled, EC by EC, according to the rules defined by the FTT-CAN protocol. At this moment, the simulator implements a fixed priority scheduler, and the errors are managed by a Deferrable Server with period T_S and capacity C_S. Step four is repeated Max_Cycle times, once for each simulated EC.

At step 4.a the EC-schedule is built and the existence of deadline misses is verified. During the bus processing phase (step 4.b), the error vector and EC-schedule are cross-checked to detect if any message is affected by an error. If so, that message is marked for rescheduling by inserting its ID in the error server queue. Finally, the error server state is updated and statistical data collected.

At step five the simulator generates a report file, which includes all relevant information about the message set used and statistics, namely: BER, worst case message response times, number of errors and missed deadlines of individual messages. It also has information about error incidence.

B. Simulation results

In order to assess the performance of the scheme proposed in this paper, we conducted a series of simulations based on benchmarks described in the literature. The selected benchmarks are: SAE benchmark [21][22], used in several projects and tool analysis, like the ones reported in [23] or [6]; Peugeot Benchmark [24]; and an extension to the SAE Benchmark (Updated_SAE) that updates the original message set to take into account modules available in today's cars [25].

Table II describes in detail the Updated_SAE benchmark message set, for illustrative purposes, where $\overline{T}$, *DLC* and *Offset* are given in number of LEC's, bytes and number of LEC's, respectively. Unless stated otherwise, all simulations use a BER of 3.0×10^{-7}, i.e., slightly higher than the aggressive profile reported in Table I. For the sake of simplicity, we assume that each EC can accommodate up to 10 synchronous messages and the server capacity is expressed in number of messages. Finally, we considered a bit-rate of 500Kbps.

Table III reports the results obtained with the Updated_SAE benchmark. The server period is kept constant (T_S=2500*LEC), while the server budget C_S varies from 1 to 10 messages. As can be seen, no deadline misses are observed during the retransmission processes with C_S equal to or greater than 5. The same experiment was repeated using the message sets defined by SAE and Peugeot benchmarks. The results are very similar to the ones reported in table III and so they are not repeated here.

TABLE II. UPDATED_SAE BENCHMARK MESSAGE SET

Bit rate = 500 kbps			LEC = 2.5 ms			LSW = 78%	
ID	**T**	**DLC**	**Offset**	**ID**	**T**	**DLC**	**Offset**
1	20	1	0	19	4	6	0
2	2	2	0	20	4	2	0
3	2	1	0	21	4	3	0
4	2	2	0	22	4	2	0
5	2	1	0	23	5	2	0
6	2	2	0	24	5	2	0
7	2	1	0	25	5	2	0
8	2	1	0	26	5	2	0
9	3	1	0	27	5	4	0
10	3	1	0	28	5	5	0
11	3	1	0	29	5	3	0
12	3	1	0	30	20	1	0
13	3	1	0	31	40	4	0
14	3	4	0	32	40	1	0
15	3	4	0	33	40	1	0
16	3	4	0	34	400	3	0
17	4	1	0	35	400	1	0
18	4	2	0	36	400	1	0

TABLE III. SIMULATION RESULTS (T_S=2500 ECS)

Nº Cycles – 20×10^6	BER=3×10^{-7} (agressive)	
C_S	**Nº missed deadlines**	**Message Error Recovery rate**
1	1810	59.2%
2	216	92.2%
3	22	99.2%
4	6	99.8%
5..10	0	100.0%

Looking carefully to table III, two main conclusions can be taken. The first one is that, as expected, the bandwidth allocated to the server is a critical design parameter. For example, when C_S=1, corresponding to BW=0.004%, only 59.2% of the errors are recovered. As C_S, and consequently the server bandwidth, grows, the error recovery rate increases monotonically until reaching 100% for C_S=5.

The second conclusion is that full error recovery rates are attained even with a very small amount of bandwidth allocated to the server. In this particular scenario, C_S=5 corresponds to a bandwidth of 0.02%, only.

The simulations have been carried out for 20×10^6 ECs, corresponding, with the Updated_SAE benchmark, to almost 200 million transmitted messages and 14 working hours. We also repeated the simulation for C_S=5 messages during a number of ECs equivalent to 15 working days and the message error recovery was steady at the 100% level.

To get a good indicator of the efficiency of the recovery method proposed in this paper, it is important comparing it with the classical approach used in time-triggered systems, which consists in reserving slots dedicated to message retransmissions during the system design phase. The static slot allocation was modeled with a polling server, using C_S=1 and T_S varying between 1 and 2500. Table IV reports the results that we obtained.

The most immediate conclusion is that full error recovery is only attainable with one retransmission slot per EC. To put this result in perspective, this scenario corresponds to a bandwidth utilization of 10%, which is around 500 times bigger than the one required by the method proposed in this paper. It should be remarked that, for classic statically scheduled time-triggered systems, the performance gap shall be even bigger, since in these systems the slots are reserved for specific messages or to specific nodes, while in this experiment the error retransmission slots were shared by all messages.

As the previous results indicate, bandwidth plays a crucial role in the attainable error recovery rate. In the FTT-CAN protocol, the bandwidth allocated to a server is expressed as a function of its period and capacity. Thus, a given bandwidth can be expressed by different combinations of C_S and T_S. To evaluate the impact of using a different period, we repeated the simulations with T_S=1000*LEC instead of T_S=2500*LEC. Table V reports the results obtained for this scenario.

TABLE IV. SIMULATION RESULTS - CLASSIC TDMA – 1 SLOT PER T_S

Nº Cycles – $20x10^6$	BER = $3x10^{-7}$ (agressive)	
T_S /LEC	**Nº missed deadlines**	**Message Error Recover rate**
1	0	100.00%
2	1	99.98%
5	2	99.96%
10	3	99.9%
25	14	99.7%
50	23	99.5%
75	39	99.1%
100	40	99.1%
200	111	97.5%
500	256	94.3%
1000	562	87.5%
2500	1612	64.1%

TABLE V. MESSAGE RECOVERY RATE USING BER=$3X10^{-7}$

CS	TS=250*LEC	TS=500*LEC	TS=1000	TS=2500
1	97.56%	94.40%	86.82%	59.20%
2	*100.00%*	99.55%	98.74%	92.20%
3	100.00%	*100.00%*	99.91%	99.20%
4	100.00%	100.00%	*100.00%*	99.80%
5..10	100.00%	100.00%	100.00%	*100.00%*

As the previous results indicate, bandwidth plays a crucial role in the attainable error recovery rate. In the FTT-CAN protocol, the bandwidth allocated to a server is expressed as a function of its period and capacity. Thus, a given bandwidth can be expressed by different combinations of C_S and T_S. To evaluate the impact of using a different period, we repeated the simulations with T_S=1000*LEC instead of T_S=2500*LEC. Table V reports the results obtained for this scenario.

In this case full error recovery requires that the server capacity U_S is at least 0.04%, thus slightly higher than for T_S=2500*LEC. As can be seen also in table V, simulation for T_S equal to 500*LEC (U_S=0.06%) and for 250*LEC (U_S=0.08%) point in the same direction. Therefore, apparently there are no advantages in opting by lower T_S values.

To stress the error recovery method proposed in this paper, we carried out a set of simulations with a BER of $3x10^{-6}$, corresponding to the ultra-aggressive environment in Table I, which is well above the ones reported in the literature.

Both the Updated_SAE and Peugeot benchmark message sets were simulated under these conditions. Figure 2 reports the results obtained. It is immediately observable that a non-null probability of non-recovered errors remains, even when the server capacity is raised well beyond the error incidence probability. A meticulous search in the log files generated by the simulator allowed tracing the origin of such phenomenon, which happens when both the transmission and the retransmission of a message instance are affected by faults. Since each retransmission has a latency of one EC, when such succession of faults affects a message with a short period, a second retransmission may not be possible before the deadline. For instance, messages with a period with 2 ECs fail the deadline in such scenario, independently of the capacity, period and type of server. The probability of this occurrence grows with the BER value, and in the simulations we conducted it was only observable for BER equal or greater than $3x10^{-7}$ (aggressive environment). Nevertheless, we can point out that message failure rate, defined as the ratio between missed messages and total transmitted messages, is always below 10^{-7}, even in an ultra-aggressive environment.

Finally, we simulated a scenario in which, during a short period of time, the BER is ten times greater than the one corresponding to the aggressive environment, corresponding to a situation in which a system is subject to a transient strong interference from the environment. The objective is to find out how much extra capacity should the server have to recover from such ultra-aggressive transient environment. New simulations, whose results are reported in Table VI, were done using a BER equal to $3x10^{-6}$, and C_S varying between 5 and 25.

Note that the source of the 2 missed deadlines reported in Table VI for C_S = {20, 25} was traced back to the scenario with faults in consecutive ECs that we referred previously, and thus is not recoverable. Hence, in this scenario, a C_S of 15 messages seems a good value. We can say, then, that handling an ultra-aggressive BER requires 3 times more bandwidth than an aggressive environment. However, it is important to realize that this translates to 0.06% of the available bandwidth, only, thus being a reasonable way to deal with unforeseen scenarios.

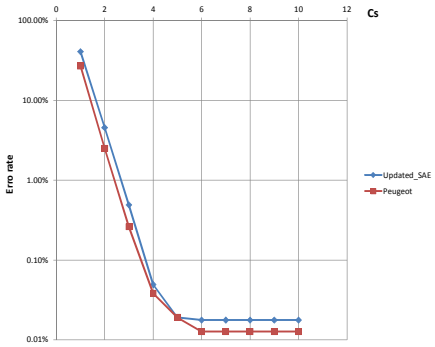

Fig. 5. Error recovery rate as a function of C_S – ultra-aggressive BER.

TABLE VI. SIMULATION RESULTS (ULTRA-AGGRESSIVE ENVIRONMENT)

Nº Cycles - $20x10^6$	T_S = 2500*LEC ; BER = $3,0x10^{-6}$	
C_S	**Nº missed deadlines**	**Message Error Recover rate**
5	44578	0.1%
10	388	99.1%
15	6	99.9866%
20	2	99.9955%
25	2	99.9955%

VII. Conclusions & Future Work

This paper presented mechanisms based on servers to deal with time-triggered message omissions due to transient faults. The devised mechanism takes advantage of distinguishing features of the FTT-CAN protocol, namely online scheduling.

One distinguishing feature of the proposed methodology is that bandwidth is only consumed when errors actually occur. This clearly contrasts with the fault-tolerance mechanisms present in classic time-triggered protocols, which are based on offline reservations that consume bandwidth independently of the actual occurrence of errors.

The performance of the proposed mechanism was extensively assessed via simulation. Several benchmarks and BER environments have been considered. The results show that, for the benchmarks found literature, near full error recovery can be attained using typically less than 0.02% of the available bandwidth. This value is orders of magnitude below the bandwidth that a classic method would require to have an equivalent performance.

A limitation was identified. The error recovery mechanism requires rescheduling, incurring in a latency of one EC. Thus, e.g. a message with a 2 ECs deadline only permits a single retransmission. If a fault affects the retransmission, the message deadline is violated. The simulations show that this scenario may become observable for a $BER \geq 3x10^{-6}$ (ultra-aggressive profile).

Future work includes the development of an analytic framework to compute the server parameters that minimize the amount of resources required to fulfill the requirements of applications and the impact of the server in the global system schedulability. A more thorough comparison with related error recovery techniques found in the literature will also be carried out. The use of a more accurate fault model, by assuming more than one fault in each EC and also the existence of burts of errors, with modelization like in [26], using Markov and Gilbert Cells, will also be investigated. Finally, it will be studied the adaptation of this work to other dual elementary cycle protocols, such as the HaRTES Ethernet switch [27] and FlexRay.

References

[1] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications (Real-Time Systems Series), Springer, 2nd Edition. Edition, 2011

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

[3] R. Isermann, R. Schwarz and S. Stolzl, "Fault-tolerant drive-by-wire systems", IEEE Control Systems, 22(5), 2002

[4] P. Peti, H. Kopetz, R. Obermaisser and N. Suri, "From a federated to an integrated architecture for dependable embedded systems", Technical Report 22 Technissche Universitat Wien, 2004

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

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

[7] G. Leen, D. Heffernan, TTCAN: a new time-triggered controller area network, Microprocessors and Microsystems, Volume 26, Issue 2, 17 March 2002, Pages 77-94

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

[9] P. Pedreiras, L. Almeida and P. Gai, "The FTT-ethernet protocol: merging flexibility, timeliness and efficiency", Proceedings. 14th Euromicro Conference on Real-Time Systems, 2002

[10] P. Pop, P. Eles and Z. Peng, "Bus access optimisation for FlexRay-based distributed embedded systems", DATE '07 Proceedings of the conference on Design, automation and test in Europe, San Jose, USA, 2007

[11] M. Felser and T. Sauter, "Standardization of industrial Ethernet - the next battlefield?", Proceedings of IEEE International Workshop on Factory Communication Systems, 2004

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

[13] 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

[14] L. Marques, V. Vasconcelos, P. Pedreiras and L. Almeida, "Tolerating Transient Communication Faults with Online Traffic Scheduling", ICIT 2012, Athens, Greece, 2012

[15] T. Nolte, M. Nolin, and H.A. Hansson, "Real-time server-based communication with CAN", IEEE Transactions on Industrial Informatics, 2005

[16] J. Ferreira, P. Pedreiras, L. Almeida and J Fonseca, "Achieving fault tolerance in FTT-CAN", Proceedings of 4th IEEE Workshop on Factory Communication Systems, 2002

[17] V. F. Silva, J.A. Fonseca and J. Ferreira, "Adapting the FTT-CAN Master for multiple-bus operation," Proceedings of 5th IEEE International Conference on Industrial Informatics, vol.1, pp.305,310, June 2007

[18] J. Proenza and J. Miro-Julia, "MajorCAN: A modification to the Controller Area Network protocol to achieve Atomic Broadcast.", Proceedings of IEEE Int. Workshop on Group Communications and Computations. IWGCC. Taipei, Taiwan, April 2000

[19] 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 (RTN2004)

[20] G. C. Buttazzo, Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications, Kluwer Academic Press, 1997

[21] SAE, "Class C Application Requirement Considerations", SAE Technical Report J2056/1 (June 1993)

[22] K. Tindel and A. Burns, "Guaranteeing Message Latencies On Control Area Network (CAN)", Real-Time Systems Research Group, Department of Computer Science, University of York, England, 1994

[23] R. I. Davis , A. Burns, R. Bril and J. Lukkien "Controller area network (can) schedulability analysis: Refuted, revisited and revised", Real-Time Systems, Volume 35, Number 3, April 2007 , pp. 239-272(34)

[24] 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, pp. 187-194

[25] 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

[26] M. Short and I. Sheikh, "Computing optimal window sizes to enforce dependable communications in time-triggered Controller Area Networks", Proceedings of the 9th International Workshop on Real-Time Networks (RTN 2010), pp. 38-43, Brussels, Belgium, 2010

[27] R. Santos, R. Marau, A. Oliveira, P. Pedreiras and L. Almeida, "Designing a costumized ethernet switch for safe hard real-time communication", Proc. IEEE International Workshop on Factory Communication Systems, Dresden, Germany, May, 2008