ACHIEVING FAULT TOLERANCE IN FTT-CAN

Joaquim Ferreira[2], Paulo Pedreiras[1], Luis Almeida[1], Jose A. Fonseca[1]

jjf@est.ipcb.pt, pedreiras@alunos.det.ua.pt, {lda,jaf}@det.ua.pt

[1]*DET / IEETA - Universidade de Aveiro*

3810-193 Aveiro, Portugal

[2]*EST - Insituto Politécnico de Castelo Branco*

6000 Castelo Branco, Portugal

Abstract: In order to use the FTT-CAN protocol (Flexible Time-Triggered communication over Controller Area Network) in safety-critical applications, the impact of network errors and node failures must be thoroughly determined and minimized. This paper presents and discusses fault-tolerance techniques to limit that impact. The particular configuration of the communication system can be more or less complex and fault-tolerant as desired by the system designer. The paper includes the fault hypothesis and presents a replicated network architecture using bus guardians. An important aspect is the replication of the master node that schedules the time-triggered traffic. In this case, it is particularly important to assure a correct synchronization of the master replicas. The mechanisms that support masters' replication and synchronization are described and their performance is evaluated. The resulting architecture allows reducing the conflict between safety and flexibility, supporting the use of FTT-CAN in safety critical applications.

Keywords: Real-time distributed systems, Fieldbus systems, Flexible real-time communication, Real-time scheduling, Fault tolerance, Safety-critical systems.

1 INTRODUCTION

Flexibility and safety have been considered as conflicting concepts basically because the former implies the ability to deal with changing requirements that, in turn, can lead to unpredictable and possibly unsafe operating scenarios. However, it is also commonly accepted that flexibility is a desired property in a system in order to support evolving requirements, simplify maintenance and repair, and improve efficiency in the use of system resources. The issue, then, is to find the compromise that allows exploiting as much flexibility as possible without jeopardizing the system safety. For obvious reasons, this is particularly

important in safety-critical systems, where uncontrolled failures cannot be tolerated. On the other hand, it is even more important in systems where the resources are scarce and have to be efficiently used. This is the case, for example, with automotive distributed computer control systems, which are subject to heavy pressure towards cost-reduction. In these systems, the communication infrastructure deserves particular attention because of the current trend to encapsulate single functions in separate nodes. This trend leads to an increased level of traffic, which requires an efficient use of the available bandwidth in order to fulfill the traffic temporal and throughput requirements.

1.1 ABOUT THIS PAPER

This paper is organized as follows; section 2 gives an overview of FTT-CAN protocol, in section 3

2 FTT-CAN BRIEF PRESENTATION

The basis for the FTT-CAN protocol (Flexible Time-Triggered communication on CAN) has been first presented in [1]. Basically, the protocol makes use of the dual-phase elementary cycle concept in order to combine time and event-triggered communication with temporal isolation. Moreover, the time-triggered traffic is scheduled on-line and centrally in a particular node called master. This feature facilitates the on-line admission control of dynamic requests for periodic communication because the respective requirements are held centrally in just one local database. With on-line admission control, the protocol supports the time-triggered traffic in a flexible way, under guaranteed timeliness (dynamic planning-based scheduling paradigm).

Furthermore, there is another feature that clearly distinguishes this protocol from other proposals concerning time-triggered communication on CAN [11][6] that is the exploitation of its native distributed arbitration mechanism. The protocol relies on a relaxed master-slave medium access control in which the same master message triggers the transmission of messages in several slaves simultaneously (master/multi-slave). The eventual collisions between slaves' messages are handled by the native distributed arbitration of CAN. Moreover, the protocol also takes advantage of the CAN arbitration to handle event-triggered traffic in the same way as the original protocol does. Particularly, there is no need for the master to poll the slaves for pending event-triggered requests. Slaves with pending requests may try to transmit immediately, as in normal CAN, but just within the respective phase of each elementary cycle. This scheme, similar to the arbitration windows in TT-CAN, allows a very efficient combination of time and event-triggered traffic, particularly resulting in low communication overhead and shorter response times.

In FTT-CAN the bus time is slotted in consecutive Elementary Cycles (ECs) with fixed duration (*E* time units). All nodes are synchronised at the start of each EC by the reception of a particular message known as *EC trigger message*, which is sent by the master node. The transmission of this message takes *LTM* (constant) time units.

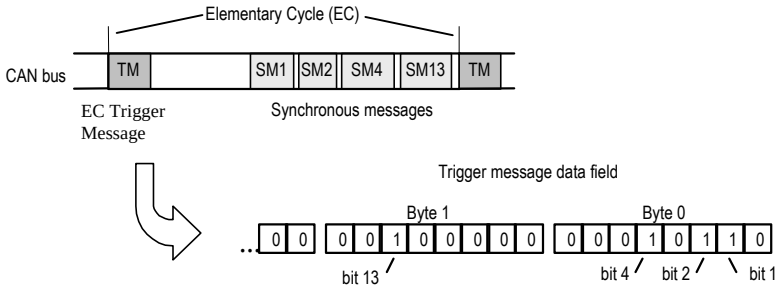

Figure 1 - Contents of the trigger message in FTT CAN

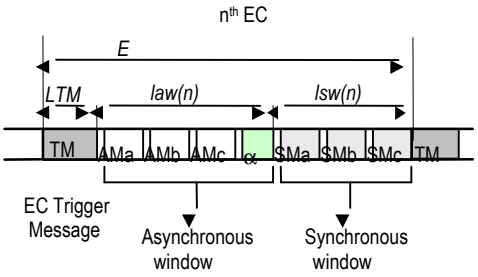

Figure 2 - The Elementary Cycle in FTT-CAN

Within each EC the protocol defines two consecutive windows, asynchronous and synchronous, that correspond to two separate phases (see Figure 2). The former one is used to convey event-triggered traffic, herein called *asynchronous* because the respective transmission requests can be issued at any instant. The latter one is used to convey time-triggered traffic, herein called *synchronous* because its transmission occurs synchronously with the ECs. The synchronous window of the n^{th} EC has a duration *lsw(n)* that is set according to the traffic that is scheduled for it. The schedule for each EC is conveyed by the respective EC trigger message (see Figure 1). Since this window is placed at the end of the EC, its starting instant is variable and it is also encoded in the respective EC trigger message. The asynchronous window has a duration *law(n)* equal to the remaining time between the EC trigger message and the synchronous window. The protocol allows establishing a maximum duration for the synchronous windows (*LSW*) and correspondingly a maximum bandwidth for that type of traffic. Consequently, a minimum bandwidth can be guaranteed for the asynchronous traffic.

In order to maintain the temporal properties of the synchronous traffic, such as composability with respect to the temporal behaviour, it must be protected from the potential interference of asynchronous requests. Thus, a strict temporal isolation between both phases is enforced by preventing the start of transmissions that could not complete within the respective window. This is achieved by removing from the network controller transmission buffer any pending request that cannot be served up to completion within that interval, keeping it in the transmission queue. As a consequence, a short amount of idle-time may appear at the end of the asynchronous window (α in fig. 1). At the end of the synchronous window, another short amount of idle-time may appear. In this case, it is due to variations in the stuff-bits used in the physical encoding of CAN messages.

The communication requirements concerning the time-triggered traffic are held in a table located in the master node, the System Requirements Table (SRT). This table contains the description of the periodic streams of messages. The relevant parameters are an identification, the data length in bytes, the corresponding maximum transmission time, the period, the deadline, a value that reflects the importance of the stream in the application, the attributes that indicate the operations that are acceptable over this stream and a list, or a range, of possible values for any specified parameter. The attributes allow limiting the level of flexibility in a per stream basis. They may specify that no changes are accepted for a given stream, or that the stream may be removed from the SRT, or that the period may assume a value in a given range, or one value from a list of possible values. On the other hand, it may also specify that a stream is unconstrained.

The communication services of FTT-CAN are delivered to the application by means of two subsystems, the Synchronous Messaging System (SMS) and the Asynchronous Messaging System (AMS). Each of these subsystems manages the respective type of traffic. The SMS offers services based on the producer-consumer model whilst the AMS offers send and receive basic services, only. Both subsystems are described in more detail in the following two sections.

3 FAULT TOLERANCE ISSUES

In order to use FTT-CAN in safety-critical applications, the impact of network errors and node failures must be thoroughly determined and minimized. Fault-tolerance techniques are used to limit that impact, which, however, increase the system cost. The particular configuration of the communication system can be more or less complex and fault-tolerant as desired by the system designer. Therefore, it is important to show that using common fault-tolerance techniques in FTT-CAN is possible.

3.1 FAULT HYPOTHESIS

The fault hypothesis considers physical faults such as those caused by electromagnetic interference, defects or damage. Concerning spatial proximity faults, FTT-CAN, as any bus-based topology, is subject to this particular kind of faults, even with bus replication, since the destruction of a single node usually also disrupts the network interface and the network connectivity itself.

As common in all bus-based systems, physical partition is not tolerated unless the bus is replicated and the partition affects just one of the busses. Also, every node in the system is considered to be fail-silent, i.e. either transmits correctly or it does not transmit at all. In this matter, a node can be fail-silent in the time domain, i.e. transmissions occur at the right instants, only, or in the value domain, i.e. messages contain correct values, only. FTT-CAN enforces fail-silence in the time domain in all nodes by using bus guardians. In what concerns the value domain, the protocol considers fail-silence in the master node, only. In this case, the scheduler and the SRT are replicated internally (see Figure 3). Whenever the EC schedule built by the replica does not match the one built by the primary then no EC trigger message is generated.

The remaining nodes are application specific and if this behavior is desired it must also be implemented within the node.

With fail silent node interfaces, fault tolerance at the network level may be achieved by replicating the nodes and the communication channels between the nodes. Being FTT-CAN a master/multi-slave architecture, special attention is put on the master replication. So, master node replication can be done both internally to implement fail silent behavior in the value domain (either software or hardware based) and externally to provide fault tolerance in case one master should fail. The overall architecture, based in a replicated broadcast bus, is depicted in Figure 3.

Membership service can be implemented via heartbeat messages scheduled by the master node according the application requirements.

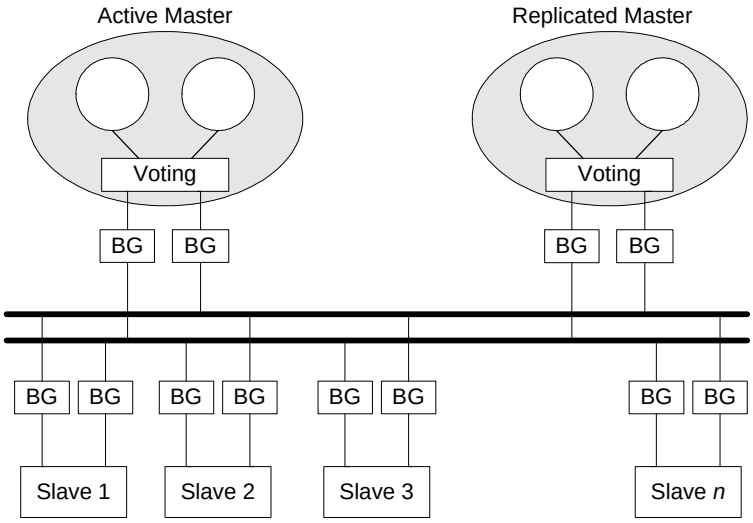

Figure 3 - Network architecture based in a replicated broadcast bus.

3.2 USING BUS GUARDIANS

Bus guardians are autonomous devices with respect to the node network controller and host processor that enforce an adequate timing in the node transmissions. In the case of FTT-CAN, two types of bus guardians can be used in the slave nodes, with two levels of complexity, performance and cost. The simpler ones receive the EC trigger message, decode the timing information concerning the start of the synchronous window and enforce the duration of both windows. Then, they block any transmission from the node that is not coherent with the respective window, e.g. the transmission of an asynchronous message in the synchronous window.

The more complex bus guardians contain the identification of the messages that are produced by that node. At run-time, they decode the EC trigger message and extract both the timing and schedule information and block any transmission from the node that is not scheduled for that EC.

In the case of the master node, the bus guardian does basically the same function. Notice that the master node may also transmit synchronous and asynchronous messages. However, there is an extra functionality that prevents the next EC trigger message of being transmitted later than a given tolerance.

The bus guardians can either be placed on the same chip of the controller although having a separate oscillator or be physically located in a separate chip outside the node, having in this case both separate oscillator and power supply. The fault coverage of the last solution is higher, but it is also more expensive.

3.3 STARTUP AND RESTART

In FTT-CAN, the active master node (higher priority one) alone controls the startup sequence, on power up; the slave nodes only start to transmit messages when explicitly instructed to do so by the master. The backup masters will also wait for the first triggers message produced by the active master, after that the replicated masters will issue the active master a synchronization request. According to this startup protocol there is no competition for the bus either from the slaves or from the backup masters.

Restarting during operation might be necessary if electromagnetic interference or other external influences lead to a fault hypothesis violation and, probably, to a complete failure of the fieldbus if the never give up strategy adopted also fails to deliver an acceptable quality of service. The fieldbus failure, once detected by the master node automatically triggers the restarting of the system without the needing to download any dispatching tables into the slave nodes, since all relevant information is convoyed in the trigger message.

3.4 IMPACT OF TRANSMISSION ERRORS IN FTT-CAN

Even if the network interfaces are guaranteed to be fail silent and fault tolerance at the network level is achieved by replication, disturbances on the bus may still occur. These disturbances, most likely caused by electromagnetic interference, become the major source of errors, since internal errors cannot affect other nodes.

In what concerns transmission errors, the CAN protocol includes a feature known as automatic retransmission. As soon as an error is detected all nodes transmit error frames in order to resynchronise. Then, the bus is again available to the transmission of messages and the node that was transmitting when the error occurred automatically retransmits the same message (providing no higher priority message is pending in other node). This feature is desirable from a point-of-view of reliable communication. However, from a timeliness point-of-view, this feature is not so interesting because the automatic retransmission takes place independently of the temporal validity of the respective message. Furthermore, this feature is also incompatible with a distributed time-triggered approach (e.g. TT-CAN) since it may cause an extension of message transmission times beyond the duration of the respective pre-allocated time slots. Thus, TT-CAN requires the use of *single shot transmission*, a feature available in most current CAN controllers that corresponds to disabling the automatic retransmission.

On the other hand, FTT-CAN does not require the disabling of the automatic retransmission since the protocol limits its maximum extent to the duration of the window where the error took place. This is achieved implicitly through the same mechanism used to enforce temporal isolation between SMS and AMS, i.e. all transmission activity is suspended at the end of each window. This characteristic of FTT-CAN leads to a desirable error confinement within both subsystems, i.e. any error in SMS does not affect

the AMS and vice-versa. Within each subsystem, extra time can be allocated in order to cope with the delays caused by errors as forecasted by an appropriate error model (e.g [9][4]).

The number of tolerable transmission errors per elementary cycle (error model) strongly depends on the target application. Stochastic [9]and deterministic [13] error models for CAN have been proposed to mimic the car environment.

In our approach an error model is feed into the dynamic online scheduler so in can incorporate slack time in the elementary cycles to accommodate possible transmission errors (fault tolerant scheduling). This passive (in the way that defensive measures are adopted before the actual error occurrence) approach guarantees the system operation under "normal" error situations as anticipated by the error model (level-1 faults). Errors beyond the error model are handled by an active mechanism (never give up strategy), based on the master node, that monitors the bus looking for missing synchronous messages and rescheduling them for future elementary cycles (whenever possible) [2]. This is made possible since the temporal resolution for the synchronous messages is the EC duration; so within the same elementary cycle, a message can be retransmitted and still maintain its timeliness. This model is based on the intrinsic properties of FTT-CAN, namely the centralized knowledge of all system and on the online dynamic scheduling that allows the use of corrective measures in case of error occurrence. The system can, thus, be designed to operate statically, triggering to a dynamic activity either to the existence of new operational requirements or in an error situation.

3.5 MASTER REPLICATION AND SCHEDULING SYNCHRONIZATION

The whole FTT-CAN based distributed system is synchronized by the reception of the EC trigger message. When this message is omitted, a temporary loss of connectivity happens. Therefore, in order to prevent such situation, a backup master must be used. This backup master monitors the network looking for EC trigger messages. Whenever the next trigger message is delayed more than a given tolerance the backup enters into action and transmits the missing EC trigger message. From that moment on, the backup becomes the primary master and the previous primary, if still active, will become a backup. Notice that more than one backup master may be used, as long as each one is assigned a different priority.

Besides internal replication of the scheduler and the SRT that prevents the master node to generate a trigger messages whenever the EC schedule built by the replica does not match the one built by the primary[1] (value redundancy), fault tolerance is also implemented by the replication of the master node itself (space redundancy). The master nodes are replica determinate if they, starting from the same initial conditions and feed by identical inputs, produce the same result. So it is of utmost importance that all replicated masters process the same data. In this perspective the synchronization process between the master nodes is critical, since a fault in the communication infrastructure, during this process, could result in the replicas obtain different values or even no values at all. This general problem of distributing data consistently even in the presence of faults is commonly know as Byzantine agreement [7] or distributed consensus. When a node broadcasts a message to the bus, Byzantine agreement requires that all non-faulty

nodes receives the same message (agreement) and that if the transmitter is non-faulty, then all non-faulty nodes receives the message actually sent (validity). Atomic broadcast is a way to achieve consensus however, native CAN does not support atomic broadcast, since if an error occurs in the last bit of the seven-bit end of frame delimiter of a message, that message may be correctly received by some controllers, erroneously by others and even in a duplicate fashion by others [14]. A suite of protocols is available [14][12] to complement CAN's functionality in order to overcome this problem.

Another possible way of reaching consensus is through the use of sequence numbers in messages, however this does not prevent messages from being received in different order, not guaranteeing total order of atomic broadcasts.

Concerning the impact of message omission and/or duplicates in FTT CAN, it can be regarded in three perspectives according to the criticality of the exchanged messages and on the nature of the messages (synchronous or asynchronous):

1. **Master replica synchronization messages and trigger messages**. In this case a 3-bit cycle sequence number conveyed in the trigger message and in the master synchronization messages eliminates the need to use more elaborate mechanisms [14][12] to impose atomic broadcast transactions in the bus.**Synchronous messages** are centrally scheduled by the master with a given periodicity and convey state variables. Omissions and/or duplicates not masked or detected by master node based error masking and detection mechanisms (see further) are not a threat to the system since control loops, for example, are generally able to maintain stability and work effectively if a small number of data values are lost (vacant sampling) [5].**Asynchronous messages** transmission is performed according to CAN standard without any supervision by the master node. Since asynchronous messages may convey safety critical data, e.g. alarms, one of the available protocols to enforce atomic broadcast may be used without restrictions, provided the overhead introduced is taken into account and the enough bandwidth is reserved.

4 SCHEDULING SYNCHRONIZATION PROTOCOL

An important aspect is the scheduling synchronization between primary and backup masters. Since the schedulers that run on the masters are dynamic, it must be guaranteed that in each EC they generate similar schedules. Thus, in every EC all backup masters compare their own schedules with the schedule conveyed in the trigger message, moreover, they also compare a short cyclic sequence number (3-bit) that is also encoded in the trigger message. Whenever an inconsistency is detected the backup master issues a synchronization request, causing the current primary master to download the SRT as well as the relative phasing information necessary to resume scheduling synchronously. This process (see Figure 4) may take a few ECs depending on the size of the SRT and on the current network utilization. The synchronization process is a time critical task since, during its execution modifications of the SRT are not allowed and the system may work without a backup master (if just one replica exists). Furthermore the overhead introduced

[1] The remaining nodes are application specific and if this behaviour is desired it must also be implemented within the node.

by the synchronization protocol also affects the performance of the asynchronous messaging system, since the synchronization protocol is based in asynchronous messages (see section 4.3).

The quantity and nature of the data that has to be received by a backup master to enable its synchronization with the active master depends on the adopted scheduling algorithm. However, usually it can be divided in two groups, one containing static message properties and other containing scheduling state dependent properties. Considering, as an example, either Rate Monotonic (RM) or Earliest Deadline First (EDF) scheduling policies, the static properties consist in the data size, period and deadline. On the other hand scheduling state dependent data of consists in the messages phases at the beginning of the next plan. The message identifier is always sent with the pertinent data.

The timeline of the synchronization process is depicted in Figure 4. Once the active master receives the synchronization request (MST_DATA_QRY), it starts to download the SRT table and the relative phasing data in two rounds. In the first round, the SRT is split and convoyed into several messages (MST_DATA_MSGPROP). These messages only carry the static information (e.g. period, deadline, message IDs, etc). Once the first state transfer round is complete, the dynamic scheduling dependent data (e.g. relative phasing) is also split into several messages (MST_DATA_SCHINF). The transmission of this last state transfer round must be enclosed within a single plan and only after the scheduling of the next plan is completed in order to assure the consistence of the time dependent scheduling data. Once the scheduling dependent data is fully received by the backup master, it waits for the beginning of the next plan to start the scheduling. After completing the scheduling of the next plan, the backup master is ready to monitor the trigger messages produced by the active master and replace it in case of failure, as soon as a new plan begins. Notice that the start of a new plan is encoded in the trigger message.

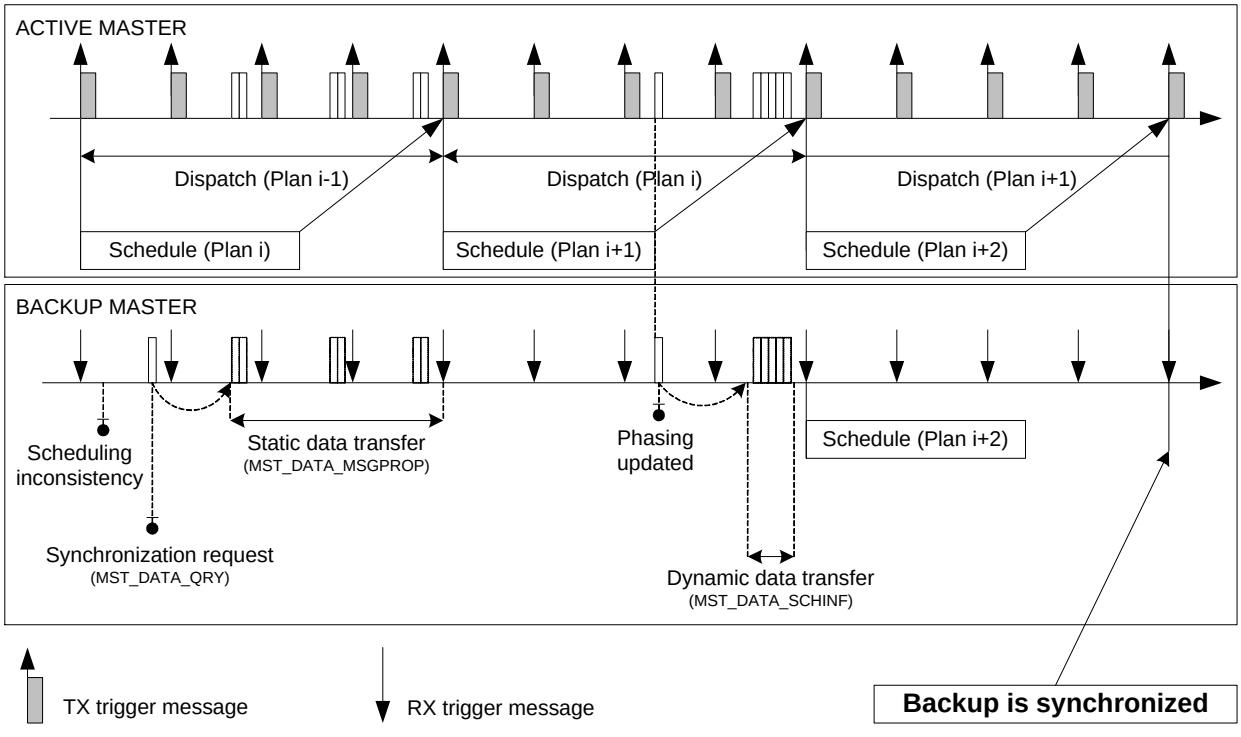

Figure 4 – Timeline of the scheduling synchronization process.

In order to tolerate the failure of the active master during the synchronization process of at least 3 masters (1 active and 2 backups) must be present and one of the backup masters would have to be fully synchronized with the primary master while the other has lost synchronicity or has been just connected to the network. This is made possible because all operator commands to change the communication requirements are distributed to the active and all backup masters by the operator console module via an atomic broadcast protocol implemented in the asynchronous messaging system. Notice that the operator console module can either be a part of the master node or it can be implemented in a separate node, as in the existing prototype.

Notice that, from the master synchronization point of view, the existence of the planning scheduler is quite convenient since it gives room for state transfer without compromising the normal network activity and also without imposing an excessive load on the masters.

If the scheduling is made in EC by EC basis or if the plan duration is not enough, the synchronization algorithm depicted in Figure 4 cannot be used, since the active master would enough room to transfer the scheduling dependent date (phasing). A more generic solution to this problem is to transfer first the static scheduling information, exactly as in the as in algorithm depicted in Figure 4, being the phasing information encoded in just a single asynchronous message convoying the EC number within the macro cycle that the active master is currently scheduling. With this data the replicated master can rebuild the scheduling (timeline) since the beginning of the macro cycle until the correct EC and synchronize after that point. Solving this problem with conventional microcontrollers may be very hard (if not impossible), however using specialized dedicated hardware [8] capable of scheduling a plan in a fraction of EC duration this might be achievable.

4.1 ACTIVE ERROR CONFINEMENT AND MASTER REPLICATION

The schedule built by the active master, convoyed in the trigger message, may differ from the one generated by the replicas without the need of resynchronization. This is due to the fact that only the active master monitors the bus looking for missing synchronous messages and, if possible, reschedules them for future elementary cycles. When this occurs, the backup masters will not issue a synchronization request, because a bit is encoded in the trigger message meaning that a transient rescheduling is under way due to an error occurrence in one (or several) of the slave nodes. If the active master based membership protocol detects a permanent failure of a slave node, then this bit is reset forcing then the synchronization of all master replicas. When a backup master becomes the active master, besides producing the trigger messages it also starts to monitors the bus looking for missing synchronous messages and reschedules them for future elementary.

4.2 TRIGGER MESSAGE TRANSMISSION BY A BACKUP MASTER: IMPLEMENTATION DETAILS

The transmission of the trigger message by the backup master after the active master node fails is implemented in an 80592 microcontroller-based prototype. It is based in a timer and in the normal CAN transmission and receive interrupts. At the backup master end, upon the reception of a trigger message, a timer is programmed to generate an interruption during the reception of the next trigger message. During the interrupt service routine (ISR) associated with the reception of a trigger message the backup master writes on the transmission buffer its own trigger message, orders its transmission and immediately after the transmission is aborted. If the active master is already transmitting a trigger message in the bus, then the abort operation is successful, otherwise the abort operation fails and the trigger message produced by the backup master is effectively transmitted. In the latter situation the backups master will become the active master (this is detected because an transmit interrupt will then be issued). This process is depicted in Figure 5.

If there were several backup masters presented in the network the situation would be similar, since possible backup master competition would be handled by native CAN arbitration. This implementation is quite efficient since the master replacement delay is a fraction of the trigger message duration, and so the induced jitter due to master replication is very low. Notice that this scheme, one can implement a single shot like operation mode.

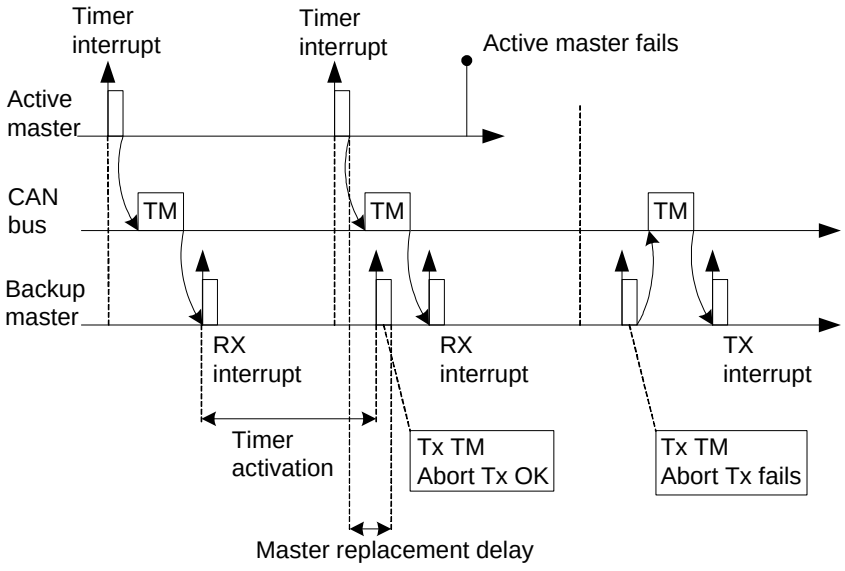

Figure 5 - Master replacement process.

4.3 COMPUTING THE WORST-CASE SECHEDULING SYNCHRONIZATION TIME

As it was previously described, the synchronization of a backup master node requires the proper reception of a set of data from the active master. During this process the backup master is unable to replace the current active master, since it does not have enough information either in the time or value domain, to

build schedules in parallel. Therefore, to assess the system reliability it is important to compute an upper bound for the time required by the synchronization process. The subsequent calculations do not include the overhead introduced by any of the available protocols to enforce atomic broadcast.

The number of CAN frames required to send either static or scheduling dependent data depends both on the quantity of messages (N_{RT}) and on the amount of data required to represent the respective set of properties for each one (MP_{Len}). Knowing that the maximum number of data bytes that can be carried in each CAN message is 8, equation 1 gives the number of CAN data frames and their respective size, needed to transmit either static or scheduling dependent data of the whole set of synchronous messages.

$$\begin{cases} \lfloor (N_{RT} * MP_{Len})/8 \rfloor_{:DLC=8} + 1_{:DLC=x} & \text{if } x \neq 0 \\ \lfloor (N_{RT} * MP_{Len})/8 \rfloor_{:DLC=8} & \text{otherwise} \end{cases}$$

$$\text{with } x = (N_{RT} * MP_{Len}) - \lfloor (N_{RT} * MP_{Len})/8 \rfloor_{:DLC=8} \quad (1)$$

Besides the data frames, the synchronization process also requires two more data frames, one sent at the beginning of the synchronization process (MST_DATA_QRY), requesting the data from the active master and other to signal the successful end of the transaction (MST_DATA_OK) or the need to update the state-dependent data frames (MST_DATA_SCHINF_REFRESH). Notice that none of these messages carry any data bytes.

The FTT-CAN protocol supports real-time asynchronous message, with guaranteed response time [10]. Giving the set of asynchronous messages exchanged on the system, the minimum bandwidth reserved for the asynchronous window and the relative priority of the asynchronous messages used in by the scheduling synchronization protocol, it is possible to obtain an upper bound for the transmission time required to send the complete set of messages, by applying the equation 2 in the calculus of the response time of the last message of the scheduling synchronization protocol.

$$R_i = s_i + w_i + C_i \quad (2)$$

where:

s_i is a blocking factor

w_i is the level-*i* busy factor

C_i is the worst-case transmission time of message *i*

However, in worst-case, the transmission of the last message containing scheduling dependent data happens just after the beginning of a new plan. In this case, the set of messages containing this type of data must be transmitted again, just after the current invocation of the scheduler has finished (see Figure 4). After receive the updated data, the out-of-sync backup master needs to wait for the beginning of the next plan to start the scheduler with same data as the active master. After having the schedule built, the

beginning of a new plan sets the instant from which the backup master becomes fully synchronized and able of acting as a master if necessary (Figure 4). Therefore, an upper bound for the time required (ST_{WC}) for a master to become fully synchronized can be computed by calculating the set of messages required by the process (M_{SP}) and applying the following equation:

$$ST_{WC} = R_{SP} + 2 * PLAN_W \quad (3)$$

where:

R_{SP} is the response time of the last message in M_{SP}

$PLAN_W$ is the plan duration (in *ms*)

4.3.1 EXPERIMENTAL RESULTS

To assess the feasibility and correctness of the proposed synchronization process, some experiments have been carried out using a 5-node network made of CANivete boards [3], based on the Philips 80C592 microcontroller. The EC duration was 8.9ms, the trigger message had 2 data bytes, supporting a maximum of 8 synchronous messages, and the maximum duration of the synchronous window was set to 4.5ms. The plan duration was 30 ECs. Apart from the synchronous messages and the scheduling synchronization messages, asynchronous messages with up to 8 data bytes, were also injected in the bus. The message set used in this experimental set up is represented in Table 1. The implemented scheduling policy was Rate Monotonic. As referred in previous sections, in this case the static data consist of the message identifier, data size, period and deadline, while the scheduling dependent data consists only in the phases of the messages at the beginning of the next plan. All these properties are encoded in one byte.

ID	*Period*	*Deadline*	*Init phase*	*Size*
1	1	1	0	1
2	1	1	0	3
3	2	2	0	3
4	3	3	0	2
5	4	4	0	5
6	4	4	0	5

Table 1 – Synchronous message set properties.

(Period, Deadline and Init phase in ECs; Size in bytes)

Using equation 1, the resulting number of messages is three 8 byte messages for the static data and one 8 byte plus one 4 byte messages for sending the state-dependent data.

The response time calculated from equation 2 is 23.062 *ms*, resulting in an upper bound for the synchronization time (equation 3) $ST_{WC} = 557.062\,ms$.

The experiment was repeated several times in different conditions and, on average, the time until full synchronization was around 385*ms*. In some of the experiments the value was considerably high 550*ms*. This can be explained by the use of a large plan, leading to a high probability of the synchronization requests being served before the end of the plan. When this situation occurs, the measured value approaches the theoretical worst-case bound. Notice that due to low processing power of the microcontrollers used in the test platform, the use of such a large plan is a requirement.

5 CONCLUSION

6 REFERENCES

[1] Almeida, L., J. Fonseca, P. Fonseca, Flexible Time-Triggered Communication on a Controller Area Network. *Proc. of Work-In-Progress Session of RTSS'98 (19th IEEE Real-Time Systems Symposium*), Madrid, Spain, December 1998.

[2] Ferreira, J., P. Pedreiras, L. Almeida, J. Fonseca. FTT-CAN Error Confinement, *Proc of FET 2001 FeT'2001 - 4th FeT IFAC Conference Fieldbus Technology*, Nancy, France, November 2001.

[3] Fonseca P., F. Santos, A. Mota and J. A. Fonseca, A Dynamically Reconfigurable CAN System, Proc. 5th International CAN Conference, San José, California, USA, 1998.

[4] Hanson, H., C. Norstrom, S. Punnekkat. Integrating Reliability and Timing Analysis of CAN-based Systems. *Proc. of WFCS 2000, 3rd IEEE Workshop on Factory Communication Systems*, Porto, Portugal, September 2000.

[5] Hong S., Scheduling Algorithm of Data Sampling Times in the Integrated Communication and Control Systems}, IEEE Transactions on Control Systems Technology, vol. 1, no. 3, June 1995, pp 225-230.

[6] ISO - International Standards Organization: Time-Triggered Communications on CAN: TC 22 / SC 3 / WG 1 / TF 6

[7] Lamport L., R. Shostak and M. Pease, The Byzantine Generals problem, ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3 July 1982, pp 382-401.

[8] Martins E., J. Fonseca. Improving Flexibility and Responsiveness in FTT-CAN with a Scheduling Coprocessor. *Proc. of FeT'01 (IFAC Conf. on Fieldbus Technology)*. Nancy, France. November 2001.

[9] Navet, N., Y.-Q. Song. Design of Reliable Real-Time Applications Distributed over CAN (Controller Area Network). *Proc. of INCOM'98 (IFAC Symp. on Information Control in Manufacturing)*, Nancy, France. June 1998.

[10] Pedreiras P., L. Almeida. Combining Event-triggered and Time-triggered Traffic in FTT-CAN: Analysis of the Asynchronous Messaging System. *Proc. of WFCS 2000, 3rd IEEE Workshop on Factory Communication Systems*, Porto, Portugal, September 2000.

[11] Peraldi, M.A. and J.D. Decotignie. Combining Real-Time Features of Local Area Networks FIP and CAN. *Proc. of ICC'95 (2nd Int. CAN Conference).* CiA – CAN in Automation, 1995.

[12] Pinho, L. and Vasques, F., Atomic Multicast Protocols for Reliable CAN Communication. In *Proc. of the 19th Brazilian Symposium on Computer Networks (SBRC 2001*), Florianópolis, Brazil, May 2001, pp. 194-209.

[13] Punnekkat S., H. Hansson and C. Norström, Response Time Analysis under Errors for CAN, Real-Time Technology and Applications Symposium (RTAS'2000), 2000

[14] Rufino J., P. Veríssimo, G. Arroz, C. Almeida, L. Rodigues }, Fault-tolerant broacast in CAN, Digest of Papers, 28th International Symposium on Fault Tolerant Computer Systems}, 1998}, pp 150-159.