Enforcing Consistency of Communication Requirements Updates in FTT-CAN

Joaquim Ferreira, Luís Almeida, José A. Fonseca
DET-IEETA
Universidade de Aveiro, Portugal
jjcf@alunos.det.ua.pt,{lda, jaf}@det.ua.pt

Guillermo Rodríguez-Navas, Julian Proenza
Departament de Matemátiques i Informática
Universitat de les Illes Balears, Spain
{vdmigrg0, dmijpa0}@uib.es

Abstract— **Traditional design approaches to safety-critical distributed systems, due to fault-tolerance reasons, have typically considered static cyclic table-based traffic scheduling. However, there is a growing demand for flexibility and integration, mainly to improve efficiency in the use of system resources, with the network playing a central role to support such properties. This calls for dynamic on-line traffic scheduling techniques so that dynamic communication requirements are adequately supported. The FTT-CAN protocol (Flexible Time-Triggered communication over Controller Area Network) has been developed specifically to deliver that kind of support with timeliness guarantees. It uses a master-slave approach with one or more master replicas for fault-tolerance reasons. The communication requirements are held in a table, that is replicated in all masters. This paper considers the problem of updating the communication requirements while maintaining coherency and synchronization between the master and all its replicas. The paper also discusses the generalization of the proposed mechanism which can easily be adapted to other dynamic master-slave protocols.**

I. Introduction

Many safety-critical embedded systems used today, e.g. in transportation systems, are distributed and rely on a fieldbus network that interconnects sensors, actuators and controllers in a reliable and timely way. One popular network access control paradigm that is used in many of these applications is the master-slave paradigm, in which a single node controls the traffic on a bus using a cyclic traffic dispatching table. Some of the interesting properties of this type of networks are the inherent global synchronization with respect to the master since this node explicitly tells each slave when to transmit, as well as the relative simplicity of supporting dynamic communication requirements because these are concentrated on a monolithic structure, the communication requirements database, in the master node, only. This simplifies the tests to assess if a system has enough resources to accept change requests (admission control) and reduces the respective reaction time with respect to distributed transmission control alternatives. With this type of operational flexibility, one can turn on and off the transmission of message streams, or vary the respective transmission rates, according to the run-time needs of the system, or even change the traffic scheduling policy on-line. This results in a higher efficiency of network utilization, freeing bandwidth that can be used to serve more streams or to facilitate error recovery.

On the other hand, master replication is essential to avoid the single point of failure and achieve fault-tolerance. Upon active master failure, a backup master immediately enters into action, replacing the failed one in a short interval of time and maintaining the network activity in a transparent way. However, when replicating the master node, the fact that the current communication requirements can vary on-line increases the difficulty in assuring the synchronization between active and backup masters. This paper presents a replication protocol which supports sustained coherent and synchronized operation of replicated masters with dynamic communication requirements. This protocol is built on top of Controller Area Network (CAN) using FTT-CAN [1], which can also be categorized as a master-slave communication protocol that was specifically developed to support dynamic communication requirements with guaranteed timeliness.

The paper is organized as follows, in the next section the fundamentals of FTT-CAN are presented, existing replication solutions are discussed and the issue of atomic broadcast in CAN is analyzed. Section III describes the system assumptions and the fault model. In section IV, the replication protocol is described, and its generalization is briefly discussed. Section V concludes the paper.

II. Problem statement and related work

A. FTT-CAN basics

In the FTT-CAN [1] protocol, bus time is broken in consecutive fixed duration cycles called Elementary Cycles (ECs). Each EC is triggered by a special message called Trigger Message (TM) that not only triggers the start of a new EC but also carries the so-called EC-schedule, which is the list of messages that must be transmitted in the respective EC by the slave nodes. The native medium access control layer of CAN sorts out collisions at bus access within the EC.

By sending only one control message per EC, the bandwidth efficiency of the protocol is substantially improved with respect to conventional master-slave access control. This has been referred to as master/multislave transmission control.

The protocol also supports synchronous (periodic) and asynchronous (aperiodic) traffic within two disjoint windows in the EC (Fig. 1). The master schedules the synchronous traffic, only. Finally, the synchronous communication requirements, e.g. period, relative phasing, deadline and transmission time

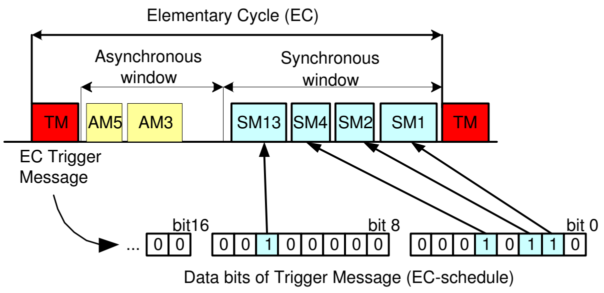

Fig. 1. Master/multislave access control. Slaves produce synchronous messages according to an elementary-cycle schedule conveyed by the trigger message. If the x data bit is 1, then message x is produced in this EC; if it is 0, then message x is not produced.

of message streams, are stored in a table called Synchronous Requirements Table (SRT). On-line update requests to the SRT go through an admission control that accepts them only if the overall traffic schedulability is guaranteed. These requests use the asynchronous windows, which can be seen as a CAN network with reduced bandwidth.

Despite the gaps between the consecutive asynchronous windows, FTT-CAN enforces that message arbitration and retransmissions occur in a logically continuous manner, so that priority inversions do not occur in the transitions.

In terms of fault-tolerance in FTT-CAN, several aspects were already covered in previous work, such as the mechanism for master failure detection and replacement [2], the mechanism to resynchronize master replicas after asynchronous start/restart [3] and the enforcement of fail silence both in the master and in slave nodes [4]. In this paper we deal with enforcing consistency between master replicas during communication requirements updates.

B. Replication techniques

When replicating the master, each of the replicas will have its copy of the SRT and will run its version of the traffic scheduler to trigger bus transactions. A transparent replacement of the active master can only occur if all master replicas have consistent SRTs and their traffic schedulers are synchronized so that, for each EC, they generate similar bus transactions. This sustained consistency and synchronization must be enforced specially during SRT updates, which are moments for potential incoherencies to build up.

As for related work, there are two main groups of replication protocols and derivatives to enforce consistency between replicas in distributed systems: active and passive replication.

Active replication: This technique, also called state machine approach [5] is fully distributed in the sense that every replica, in parallel, receives and processes the same sequence of client requests and sends back the reply. Consistency is enforced because when fed with the same inputs in the same order (usually using atomic broadcast [6]), replicas will produce the same output. The requests are handled independently but must be processed in a deterministic way. This method is simple and possible node failures are transparent to the clients because other replicas also process the requests. However the determinism constraint may be difficult to enforce (e.g. in a multi-threaded node).

Passive replication: In this technique, also known as primary backup [7], clients send their requests to the primary replica that is responsible for processing them and returning the responses back to the clients. The backup replicas only interact with the primary and apply the respective updates. No determinism constraint is necessary but special care must be put on the mechanisms that enforce agreement between primary and backups (usually a membership service). A failure in the primary before sending the reply to the client cannot be masked by passive replication. In this case the client will time-out and re-issue the request.

Two variants of these replication protocols are semi-active and semi-passive replication. In the former, the replicas do not need to process client requests in a deterministic way. Each time replicas need to make non-deterministic choices, one of them (leader) makes the decision and informs the others (followers). Semi-passive replication [8] can be implemented in the asynchronous system model without requiring a membership service to agree on a primary.

C. Atomic broadcast

A particularly important feature, especially for active replication, is atomic broadcast. This means that all transmitted messages are consistently delivered by all the correct nodes of the network, in the same order [9]. Nevertheless, the CAN protocol does not guarantee this property in all the possible circumstances [10] [11] [12]. Although solutions to achieve atomic broadcast in CAN have been already suggested, none of the proposed solutions seems to be directly applicable to FTT-CAN. Solutions in [12] and [10] are based on asynchronous message confirmation/retransmission and need certain adaptations to comply with the timing of the FTT-CAN protocol. Moreover, further adaptation is required to take advantage of the synchronous characteristics of this protocol, particularly the regular transmission of the TM, in order to achieve higher efficiency. The approach in [13] is based in extra hardware and uses continuous retransmission trials in case of network errors, interfering with the timing definitions of FTT-CAN. The MajorCAN protocol [11] enforces atomic broadcast at the frame level and would solve most of the consistency and synchronization problems related with replica management. However it goes beyond the CAN standard, since it proposes a new format for the CAN error frames that copes with the last but one bit error problem.

Due to this incompatibility between FTT-CAN and the previously suggested solutions for atomic broadcast, in the present work the atomic broadcast property is not assumed for the CAN network. In particular, our protocol will tolerate inconsistent message omissions as well as inconsistent message duplicates. Nevertheless, since the use of the MajorCAN would lead to a significant simplification of the replica management protocol, its integration with FTT-CAN is being considered as a future extension of this work.

III. System assumptions and fault model

In order to better explain the replication protocol proposed in the next section some underlying properties related to fault-tolerance aspects of both CAN and FTT-CAN are shown here. The following properties of CAN state the lack of support for atomic broadcast caused by a possible error in the last but one bit of a CAN frame [10].

CAN.p1 - Whenever a node successfully transmits a CAN frame, such frame is consistently received by all correct CAN controllers connected to the network.
CAN.p2 - Whenever a node attempts to transmit a CAN frame and it does not succeed due to an error, then some correct CAN controllers may receive the frame correctly while others may reject the same frame. This inconsistency can only occur if there was an error in the last but one bit of the frame. In all other situations, all CAN controllers consistently reject the frame. The following two properties are direct corollaries of this one.
CAN.p3 - Upon message retransmission caused by a network error, a correct CAN controller may receive the same frame correctly more than once.
CAN.p4 - Whenever a correct CAN controller receives a valid frame, without further information there is no way to tell whether the sender successfully sent that frame or failed in sending it due to network errors.

In what concerns FTT-CAN, the transmission of the Trigger Message is particularly relevant since it conveys the present state of the active master, mainly the EC-schedule, and serves as a synchronization mark. Its transmission mechanism is the following: the active master transmits the TM and, in case of error, tries to retransmit it during a given window (typically about half the EC duration). The backup masters try to transmit a TM with lower priority and a small delay relative to the active one and in single-shot mode with immediate abort after transmission request (i.e. transmission takes place if bus is idle, only). Only one can succeed and only if the active master failed, resulting in the following properties:

FTT-CAN.p1 - Only one master can transmit a TM successfully within each EC. This property results directly from the TM transmission mechanism referred above. A simple corollary is that there is never more than one master in the active state because a master enters this state upon successful transmission of a TM, only.
FTT-CAN.p2 - Whenever there are errors in the transmission of the TM, since its retransmission window is limited, there is a residual probability that its delivery is inconsistent (properties CAN.p2 and CAN.p3). In this case, in a single EC some nodes will either not receive any valid TM or receive TM duplicates. Notice that as long as the active master tries to transmit, no backup master will take over. If one does, only omissions can take place due to using the single-shot transmission mode. However, since the TM conveys state information that is idempotent, neither TM omissions nor duplicates cause inconsistencies to the protocol. Omissions prevent slaves from transmitting during that EC and duplicates have no impact except that the timings relative to the TM reception, such as the synchronous window start, must be corrected according to an estimation of the TM delay.
FTT-CAN.p3 - The TM reflects the active master state and is broadcast in the system, thus it is considered the reference system view. Whenever a backup master finds an inconsistency with the TM, due to some unforeseen reason beyond the fault model or during restart or startup, the state of the active master prevails and the backup reverts to unsynchronized state, requesting resynchronization as soon as possible [3].

A. Fault model

Every node in the system is considered to be fail-silent. FTT-CAN may enforce fail-silence in the time domain in slave nodes by using bus guardians [4]. Fail-silence in the value domain is considered in the master node, only. In this case, two independent master nodes share the same network interface. Transmission of the TM is allowed when both nodes try to transmit the same EC-schedule within a short time window. Otherwise, the TM is not generated [4].

Physical bus partition is not tolerated unless the bus is replicated and the partition affects just one of the buses. The addition of bus replication to FTT-CAN is also possible but it is out of the scope of this paper.

No masquerading faults are considered, e.g. a malicious node forcing the transmission of incorrect TMs.

IV. SRT update protocol

The replication protocol herein proposed (see Figures 2 and 3) is a semi-active one in the sense that all requests are issued to every replica that processes them in parallel, however atomic broadcast is not enforced and inconsistencies may arise. The protocol uses a leader-followers approach, with the current active master playing the leader role and synchronizing all replicas by means of the TM (basically, maintaining property FTT-CAN.p3). This approach also allows replicas to detect loss of consistency with the active master.

The protocol is divided in three phases. The first one is the request phase in which update requests concerning the communication requirements are sent by the application, without an atomic broadcast protocol, and are queued internally in the active master and its replicas. The second phase is the processing of requests, one by one. This phase lasts for a pre-determined number of ECs per request, setup at configuration time, depending on the time required for admission control. In this case we will consider two. Finally, the third phase is the reply phase, during which the active master notifies the application of the request result. An important aspect is that the ID of the request being processed, as well as a code corresponding to the current phase, are piggybacked on the

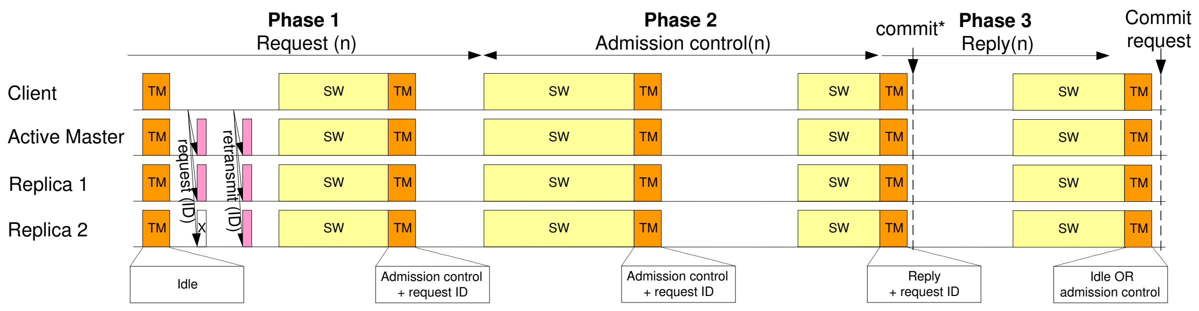

Fig. 2. Timeline of the SRT update protocol.

TM, which then serves as an acknowledge for the application and as a synchronization mark for the master replicas.

The FTT-CAN message ID encoding schema allows for 64 different IDs (6 bits) for SRT update requests. The requests queue within each master is sorted by decreasing CAN IDs mainly to enforce the fixed priority philosophy of CAN and facilitate the detection of request duplicates. As referred above, the request ID together with the status of the request process are piggybacked onto the TM, using one byte of its data field (with CAN 2.0A this leaves 48 bits to encode the EC-schedule). The bit encoding is the following:

- The six least significant bits encode the request ID.
- The two most significant bits are used to indicate the current state of the request process

00	- idle, no request being processed
01	- admission control
10	- reply accept
11	- reply reject

A. Description of the SRT update protocol phases

1 - Request phase: This phase starts when a client (an application node) issues an SRT update request. This request is issued with retransmissions enabled. Therefore, there is a high probability that the request will end up being successfully transmitted and queued consistently in all masters (property CAN.p1). However, the retransmission process may be interrupted, either because the requesting node crashed or because the request has been acknowledged within the TM. In both cases, some master replicas may have received the request while others have not (property CAN.p2) leading to inconsistencies in the queues. These inconsistencies will be cleared in the following phases.

2 - Admission control phase: When idle or after processing the last request and successfully transmitting the reply, the active master fetches the request in the head of the SRT update requests queue, inserts its ID in the TM, sends the TM out at the appropriate instant and triggers the admission control process at the start of the synchronous window. Upon receiving the TM, all master replicas scan their requests queue for the request indicated in the TM.

If they find it, they dequeue it and trigger the admission control process also at the start of the synchronous window. In case the current request is not in the queue, the master replica waits for the request until the end of the asynchronous window. This waiting time is referred as timeout, in the flowcharts of the client and of the backup master, depicted in Fig. 3. The application, in its turn, receives the TM, identifies the request ID and considers the request as correctly transmitted but continues with the respective retransmission up to the end of that EC to increase the chances that the request will reach all master replicas. If in the meanwhile that request is effectively received, the replica handles it and triggers the admission control process and remains synchronized with the active master. Otherwise, it declares itself unsynchronized and issues a resynchronization request (see [3]).

3 - Reply phase: When the admission control phase finishes in the active master (notice that it should also be finished in all replicas), the TM is piggybacked with the request ID plus a reply code meaning whether the request is to be accepted or rejected. This is detected in all replicas that should agree on the result of the admission control. When the result is "accept", the request is committed to the SRT in the EC corresponding to a transition of state from "reply" to "admission control" (if there are other requests pending) or from "reply" to "idle" (if the requests queue is empty).

The protocol described above is based on a synchronization enforced by the transmission/reception of the TM. This fact requires that its transmission must be consistent. The technique used to enforce such consistency is based on property CAN.p1 according to which the active master knows when the TM was consistently transmitted. Thus, whenever the TM transmission is not successful, the active master maintains the SRT update protocol in the same state for the next EC. The protocol only advances from state to state upon successful TM transmission. Due to the idempotency of the TM (property FTT-CAN.p2), inconsistent duplicates cause no problem to the protocol and all replicas follow the active master.

In the last part of the protocol, the request is committed to the SRT as soon as the state transition referred above is detected. This means that the commit is carried out as soon as a TM is received indicating such transition. If there are

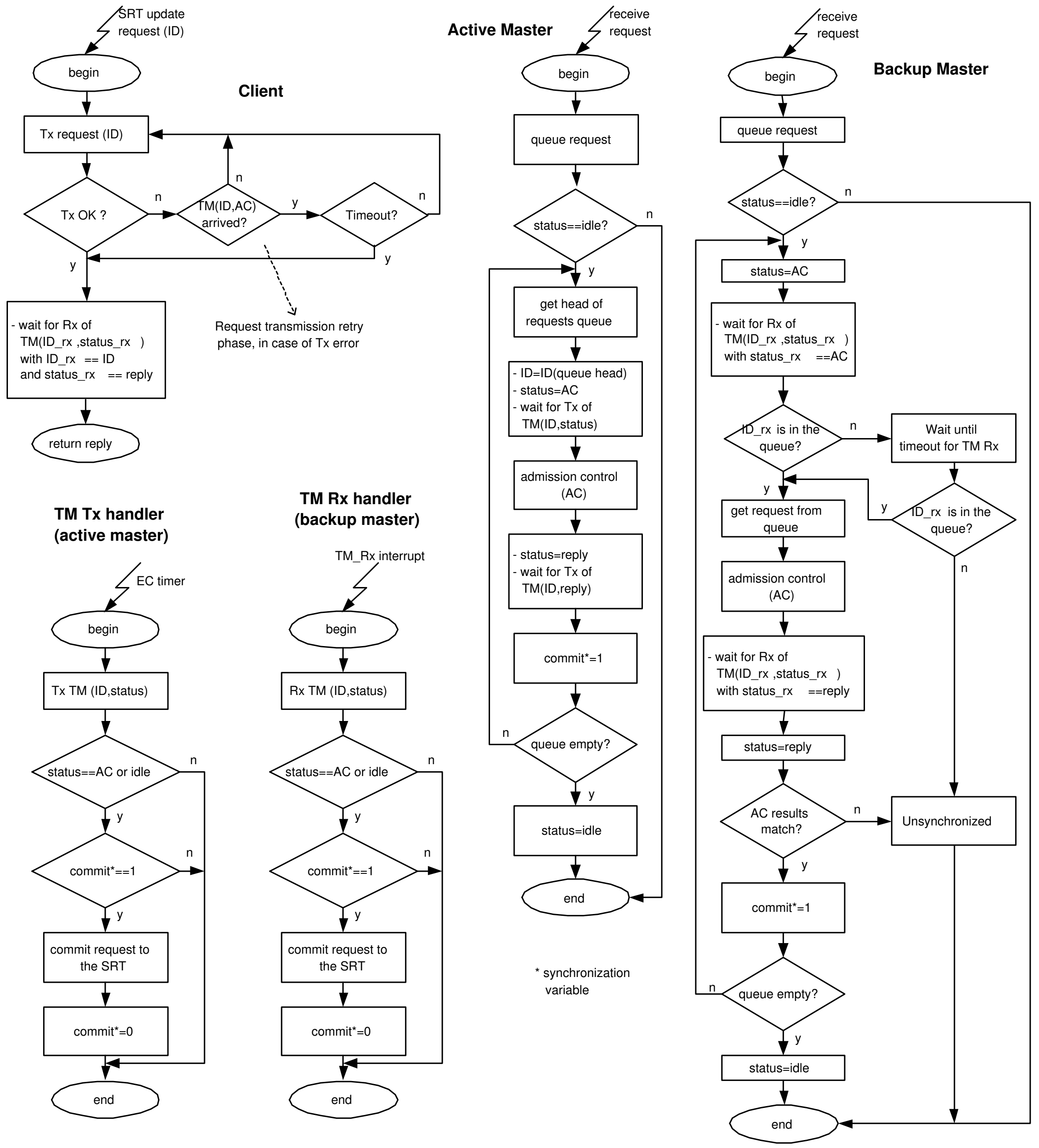
SRT update request (ID)
begin
Client
Tx request (ID)
Tx OK ?
TM(ID,AC) arrived?
Timeout?
Request transmission retry phase, in case of Tx error
- wait for Rx of TM(ID_rx ,status_rx) with ID_rx == ID and status_rx == reply
return reply
Active Master
receive request
queue request
status==idle?
get head of requests queue
- ID=ID(queue head)
- status=AC
- wait for Tx of TM(ID,status)
admission control (AC)
- status=reply
- wait for Tx of TM(ID,reply)
commit*=1
queue empty?
status=idle
end
* synchronization variable
Backup Master
status=AC
- wait for Rx of TM(ID_rx ,status_rx) with status_rx ==AC
ID_rx is in the queue?
Wait until timeout for TM Rx
get request from queue
- wait for Rx of TM(ID_rx ,status_rx) with status_rx ==reply
status=reply
AC results match?
Unsynchronized
TM Tx handler (active master)
EC timer
Tx TM (ID,status)
status==AC or idle
commit*==1
commit request to the SRT
commit*=0
TM Rx handler (backup master)
TM_Rx interrupt
Rx TM (ID,status)
y
n

Fig. 3. Simplified flowcharts of the SRT update protocol as seen by the client, the active master and by the backup master.

inconsistent TMs in this part of the protocol, some replicas may commit before others. This could cause a transitory discrepancy in the EC-schedules of replicas and active master. Therefore, all replicas switch off the policing mechanism, that compares their own EC-schedule with the one conveyed within the TM, between the reply phase, if the request is accepted, and the request commit. This prevents replicas from issuing costly resynchronization requests, when they are only temporarily unsynchronized. Notice that the active master will not start processing other request while the reply from the previous was not successfully transmitted.

If the active master crashes during the SRT update process after inconsistently transmitting a TM, the replica that takes over the network control proceeds with the update protocol according to its view of the system. If the new active master did not receive the inconsistent TM just before the crash, inconsistencies may arise leading to resynchronization requests issued by other replicas. Notice that only replicas that are synchronized with the previous active master can replace it.

If several requests occur in a short interval they are queued and handled one by one. Notice that, as referred above, after finishing processing each request, if there are requests pending, the active master fetches the one in the head of the queue and continues the process with the admission control phase. This results in a sequence of admission control and reply phases until all queued requests are processed.

The worst case response time to an SRT update request depends on whether there were inconsistencies in the transmission of TMs during the request processing. To determine an upper bound it is necessary to use an appropriate error model that allows estimating the number of extra ECs that each phase may require for consistent TM transmission. Moreover, it is also necessary to establish a minimum inter-transmission time of the requests from the same node, which can be easily enforced by the protocol. With an estimation of such worst-case response time, a timeout can be setup in the requesting nodes, after which they give up waiting for a reply.

B. Generalization of the protocol

Despite being specific to FTT-CAN, the proposed protocol seems to be adaptable to other dynamic master-slave protocols, such as WorldFIP (with dynamic bus arbitrator table) or FF-H1 (with dynamic LAS scheduling profile), or more generically to any master-slave network that support:

- Dynamic or multiple traffic dispatching tables. This is essential to support dynamic communication requirements. In the former case, such as in FF-H1, there are already mechanisms (dynamic scheduling profile) to support on-line changes to the traffic dispatching table. In the latter case, such as in WorldFIP, several traffic dispatching tables can co-exist and be swapped on-line.
- Periodic and aperiodic traffic in a cyclic framework. The operational flexibility mentioned above refers to the periodic traffic, only. The requests for changes in the periodic traffic are conveyed using aperiodic messages and thus, support for this type of traffic is required. The cyclic framework, using micro-cycles or elementary-cycles (ECs), is also common in the referred networks and it is used in the protocol to set-up checkpoints for synchronization purposes.
- Master replication mechanism. This is necessary to circumvent the single point of failure formed by the single master and most of the existing master-slave networks do provide such a mechanism.

V. Conclusions and future work

This paper addressed the problem of enforcing consistency during updates of the Synchronous Requirements Table (SRT) in FTT-CAN, using replicated masters. Since the SRT is dynamic, it is fundamental to assure that replicas remain synchronized with the active master while handling update requests from the application. The proposed protocol is a semi-active one in the sense that all requests are processed in parallel in every replica. Possible local inconsistencies arising from not using an atomic broadcast protocol, are consistently cleared during the protocol execution by the active master in a leader-followers approach.

The protocol uses periodic master messages to broadcast the master's view to its replicas and application, in a bandwidth efficient way. Moreover it also takes advantage of the CAN property that allows a transmitter to know when it has consistently transmitted a message.

The formal validation of the proposed protocol is currently being considered as well as the use of MajorCAN together with FTT-CAN. This seems to be a promising future alternative to enforce consistent updates of the communication requirements with a considerable simplification of the proposed protocol due to the atomic broadcast support at the frame transmission level.

References

[1] L. Almeida, P. Pedreiras, and J. A. Fonseca, "The FTT-CAN Protocol: Why and How," *IEEE Transactions on Industrial Electronics*, vol. 49, no. 6, 2002.

[2] J. Ferreira, P. Pedreiras, L. Almeida, and J. Fonseca, "Achieving fault tolerance in FTT-CAN," *Proceedings of the* 4^{th} *Workshop on Factory Communication Systems (WFCS 2002)*, 2002.

[3] E. Martins, J. Ferreira, L. Almeida, P. Pedreiras, and J. Fonseca, "An Approach to the Synchronization of Backup Masters in Dynamic Master-Slave Systems," *Proceedings of the WiP Session of* 23^{rd} *IEEE International Real-Time Systems Symposium (RTSS)*, 2002.

[4] J. Ferreira, L. Almeida, E. Martins, P. Pedreiras, and J. Fonseca, "Components to Enforce Fail-Silent Behavior in Dynamic Master-Slave Systems," *Proceedings of SICICA'2003* 5^{th} *IFAC International Symposium on Intelligent Components and Instruments for Control Applications*, 2003.

[5] F. B. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial," *ACM Computing Surveys*, vol. 22, no. 4, 1990.

[6] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding Replication in Databases and Distributed Systems," *Proceedings of* 20_{th} *International Conference on Distributed Computing Systems (ICDCS'2000)*, pp. 264–274, 2000.

[7] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg, "The primary Backup Approach," *S. Mullander, ed., Chapter 8, second edition*, pp. 199–216, 1993.

[8] X. Defago, A. Schiper, and N. Sergent, "Semi-Passive Replication," *Proceedings of Symposium on Reliable Distributed Systems*, pp. 43–50, 1998.

[9] P. Veríssimo and L. Rodrigues, *Distributed Systems For System Architects*. Kluwer Academic Press, 2001.

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

[11] J. Proenza and J. Miro-Julia, "MajorCAN: A modification to the Controller Area Network to achieve Atomic Broadcast," *IEEE Int. Workshop on Group Communication and Computations. Taipei, Taiwan*, 2000.

[12] L. Pinho and F. Vasques, "Atomic multicast protocols for reliable CAN communication," *Proceedings of the* 19^{th} *Brazilian Symposium on Computer Networks (SBRC 2001)*, pp. 194–209, 2001.

[13] J. Kaiser and M. Livani, "Achieving Fault-Tolerant Ordered Broadcasts in CAN," *Proceedings of the European Dependable Computing Conference*, pp. 351–363, 1999.