Using FTT and stars to simplify node replication in CAN-based systems

Julián Proenza, Manuel Barranco, Joan Llodrà
SRV-DMI. Universitat de les Illes Balears. Spain
{julian.proenza, manuel.barranco}@uib.es

Luís Almeida
Instituto de Telecomunicações, Universidade do Porto, Portugal
lda@fe.up.pt

Abstract

Nodes, among the components of distributed embedded systems, exhibit the greatest permanent failure rate among. Thus, providing tolerance to nodes faults is mandatory whenever high-reliability is required, being node replication the most common technique for that purpose. This paper proposes a novel technique suitable for CAN-based systems that simplifies existing approaches taking advantage of a star topology and the FTT protocol. [1]

1. Introduction

Despite the success of the Controller Area Network (CAN) [7] protocol in many application areas, there is a general belief that CAN is unsuited to critical applications [10]. One the reasons is the protocol limited support for fault tolerance, particularly tolerance to nodes faults, which impacts strongly the final reliability [5].

The CANbids project [12] presented a solution for node replication in which nodes periodically exchange the results of their computations and vote on them to achieve fault masking. This is a complex solution given the absence of knowledge on the actions timings (event-triggered approach). In particular, it relies on complex redundant internal nodes structures that use non-trivial protocols to ensure a system-wide consistency [12].

This paper proposes adopting a star topology and the Flexible Time-Triggered over CAN (FTT-CAN) [1] protocol to significantly simplify the approach for nodes replication. The a priori knowledge on the channel traffic provided by FTT-CAN together with a central hub with error containment capabilities can restrict the nodes failure semantics without overly complicating their internal design and simplifying the voting protocol. A special module called *Node Replication Manager* (NRM) allows reducing the attrition caused by faults by recovering nodes that cannot recover by themselves.

[1]This work was partially supported by the Portuguese Government through FCT - Fundação para a Ciência e a Tecnologia, in the scope of project Serv-CPS -PTDC/EEA-AUT/122362/2010.

2. System and fault models

Our system model assumes using the FTT-CAN [1] protocol, in which a master divides the communication time into rounds of fixed duration called Elementary Cycles (ECs), each of which is divided into an asynchronous window (for event-triggered traffic) and a synchronous window (for time-triggered traffic). Each EC starts with the master sending a Trigger Message (TM), which synchronizes the slaves and tells them the synchronous messages they have to transmit in the next synchronous window, according to the System Requirements Database (SRDB). Slaves may request changes to the SRDB, which are subject to an admission control.

On this FTT protocol, our design uses active replication to provide node fault tolerance, i.e. several nodes execute replicas of the same program and, after each partial computation, they exchange their results and vote on them. We assume that the replicas are identical pieces of software, allowing tolerating hardware faults but not software (design) ones. Yet, we will generally follow the replication terminology of *N-Version Programming* (NVP) [2], despite NVP assuming replicas with design diversity.

Each replica is partitioned into a set of *segments*. In execution, each time a replica finishes a segment, it issues a vector of results of this segment, called *cc-vector*. Then a *decision algorithm* is executed to obtain a *consensus cc-vector* which is sent back to all replicas to be used in the following computations. This mechanism, called *cc-point*, provides fault masking in a minority of replicas. We will use *a-replica* to refer to any application program replica.

Regarding the fault model, the proposed mechanisms tol-

erate temporary hardware faults in the channel and both temporary and permanent hardware faults in the nodes. Intentional and design faults are out of the scope of this paper.

3. System Architecture

Our architecture is based on error containment boundaries enforced around each node that prevent error propagation. These boundaries are built by restricting the nodes failure semantics and preparing the other nodes to deal with the errors that may still occur.

Since a node is essentially executing the application task, performing cyclic votings on its results can easily handle *incorrect computation failure semantics* [3], i.e. failing by providing incorrect results either in the value or in the time domains. Conversely, coping with more arbitrary failure modes, e.g. impersonating other nodes, sending different values to different nodes or babbling in the channel, requires more complicated voting protocols. To avoid such complication, we decided to design mechanisms to deal with such modes.

A key component in the architecture is the central hub that enforces nodes failure semantics without complicating their design, e.g. without internal duplication and comparison. The hub *polices* its ports and uses the traffic a priori knowledge provided by the FTT periodic TM for consistency checks. For example, the hub allows each node to send one cc-vector per segment, only, in the synchronous window of the EC, thereby preventing *babbling idiot failures* [8]. Likewise, the hub makes sure that each cc-vector from each node is sent in broadcast eliminating *two-faced failures* [11], i.e. a node sending different versions of the same messages to different nodes. Finally, the hub prevents the transmission of cc-vectors that are sent with a wrong identifier thereby eliminating *impersonation failures* [11],i.e. a node impersonating other nodes.

Given the high criticality of the hub in the architecture, its circuits are internally duplicated and compared thus enforcing crash failure semantics. Nevertheless, hub replication to deal with the single point of failure that it represents is left for future work.

Beyond these mechanisms included in the hub, enforcing the desired failure semantics still requires certain properties of the underlying communication protocol. Specifically, we need a protocol providing *Reliable Broadcast* (RB) [6] as well as the ability to prevent babbling idiot behaviour generated by the channel itself.

Despite the known limitations of CAN in providing RB, we will assume this property holds and we will deal with RB in the specific scope of FTT-CAN in future work.

Concerning the babbling idiot behaviour, CAN normally prevents it using its native error counting and node isolation mechanisms. However, these are also known to be ineffective when there are direct electrical connections between multiple nodes without proper error containment, as in buses [4]. For example, a transceiver that continuously sends a dominant value prevents further communication.

To overcome this limitation, we use the CANcentrate [4] hub given its high capacity to diagnose typical CAN physical layer permanent failures, i.e. stuck at dominant, stuck at recessive or bit flipping failures, and disconnect the affected ports.

Beyond the mechanisms presented before for restricting the nodes' failure semantics and contain certain errors, it is also important to deal with the errors that may cross the containment boundaries. For each received message, nodes have to determine if it originates from the FTT master, the hub or from another node. In the first two cases, it is assumed correct since the FTT master and the hub exhibit crash failure semantics. The latter case is dealt with voting, given the incorrect computation failure semantics of nodes. Finally, nodes, the FTT master and the hub must consider the potential omission of any message.

3.1 Organization of the fault-tolerance operations

In this section we focus on the fault tolerance operations performed at this architecture's upper layer, which deals with the execution of the application software. Other lower-level fault tolerance operations (e.g. the ones performed by the CAN controllers) are out of the scope of this paper. Nevertheless, for some of these lower layers some requirements on what they should do will be discussed.

The main operation performed in this upper layer to achieve fault tolerance is the voting on the cc-vectors issued by the a-replicas at the end of each segment. This voting, which is executed at each node after the cc-vectors have been exchanged at the synchronous window of the scheduled EC, provides *error compensation* [9] (i.e. fault masking). Since all nodes are to execute the same voting, it is necessary that this replicated operation presents replica determinism [11]. In this architecture this can be achieved by ensuring that non-faulty replicas of the voting do no use non-deterministic constructs in their programming and that they vote on the same set of cc-vectors. This is very easy since the channel provides RB and there is an a priori agreement that all cc-vectors for a specific segment have to be exchanged in a specific EC.

To improve the global dependability, each node performs the following three additional fault-tolerance operations on its own faults: *error detection* (i.e. detection of its own errors); *fault passivation* [9]; and *recovery* of its a-replica when it has suffered a temporary fault. Each node performs detection of its own errors by comparing its own cc-vector with the consensus cc-vector it calculates by voting. Each node performs fault passivation by disconnecting itself from the network when it is affected by faults. To prevent a quick

attrition of redundancy, disconnection should not be permanent if the node is only affected by temporary faults. Therefore, each node maintains an error counter for itself and its a-replica which is increased each time it detects a new error after a voting, and decreased when no error is detected. Only when the error counter reaches a pre-established threshold, the node considers itself permanently faulty and disconnects itself from the rest of the system. Concerning recovery after temporary faults, for a-replicas we organize the voting of their cc-vectors in such a way that the a-replica receives by means of the consensus cc-vector the information that it needs for recovering from its errors (i.e. the a-replica uses the results of the voting as the input of its next segment).

Obviously no guarantee on the correct operation of these mechanisms can be obtained. A node can fail to properly detect its own errors, and even if it does not fail to do it, it can later on fail to disconnect itself from the rest or even incorrectly recover and continue providing erroneous cc-vectors. This is due to the lack of failure restriction of the nodes. More accurately, only the failure behaviour seems restricted to the other nodes after the hub prevents the transmission of certain messages but each node can still fail arbitrarily, send arbitrary values to the hub and internally make arbitrary erroneous actions. In order to increase the chances of errors to be properly detected and recorded and of nodes to be correctly recovered after failures, a device called *Node Replication Manager* (NRM) will also execute the same voting as the nodes. This should provide it with all the necessary information for being able to detect the errors of all nodes, keep an error counter for each of them, order the disconnection of one of them and also help a node to recover, in case the severity of its failure does not allow it to recover by simply using the last result it obtained in the voting. The NRM will be internally duplicated and compared to force it to present crash failure semantics.

4. The role of the NRM

As indicated above, the NRM is a new device that in our architecture retains the ultimate responsibility for managing the node replication, so as to increase the chances of node recovery while at the same time making unnecessary for the nodes to present costly internal structures for restricting their failure semantics. Note that if there was no NRM the alternative would be that regular nodes help others in recovery and thus their failure semantics should be restricted.

The NRM has been introduced as a mechanism to detect faulty nodes and order their recovery, in such a way that it either performs these tasks properly or it crashes without affecting the nodes. Since the NRM acts when the node is severely faulty and cannot recover by itself, we will call this process *reintegration* instead of simply recovery.

Despite the prominence of the NRM in this process, it is important that the nodes are as active as possible in their reintegration. Thus when a node sees that its own error counter has reached a threshold (meaning that it has registered a series of consecutive errors in the last votings) it should carry out a complete reset. After finishing the reset it should wait for the next TM to get a basic resynchronization with the rest of the nodes and then it should send a *reintegration request* at the beginning of the asynchronous window of that EC. The NRM should then transmit in the same asynchronous window the so-called *reintegration information* that should include all that is required for a proper reintegration of the node, e.g. the value of the node's error counter.

If the node is more seriously damaged, e.g. using a wrongly low value of its error counter, it can be unable to realize that it has to ask reintegration. This is why the NRM will send a reintegration order to the nodes that have reached their error thresholds. The order will be sent during the asynchronous window and the affected nodes should react as described above: performing a reset and then sending a reintegration request.

Obviously a node can be even more seriously damaged and ignore the reintegration order from the NRM. This is the reason for introducing a watchdog timer in the design of the node. Thereby the node's software should be organized in such a way that a periodic restart of the watchdog is timely executed. If this timer is not restarted it would mean that the node is lost and thus may be unable to obey the NRM orders. Upon watchdog expiration a hardware-induced reset will occur and the node will resume computation requesting reintegration as described above.

Faulty nodes may wrongly send reintegration requests, e.g. in a continuous manner, thereby causing a babbling idiot scenario. The hub would be the responsible for ensuring that these wrong behaviours do not affect the other nodes. Thus the hub will only permit the transmission of requests during the asynchronous window and only one request per node.

Whereas the recovery potentially achievable with the simple feeding of the consensus cc-vector to the local a-replica takes place just after a segment is executed and before the next one starts, the reintegration we are describing now is triggered in such a way that we cannot guarantee that the node was executing the segment that corresponds to the consensus cc-vector provided with the reintegration information. Therefore this reintegration is much more likely to succeed if the application is structured in such a way that a node cyclically executes the same segment and, thus, it is always possible for it to use the consensus cc-vector provided as starting point of the next iteration. Fortunately this is the case of control loops such as those implementing a PID.

Due to the addition of the NRM, a third level of error containment is introduced in the system. More specifically, the hub will perform 3 different levels of error containment. First the CANcentrate-like mechanisms devoted to contain

the errors generated at the channel's physical layer. Second the mechanisms based on the a priori knowledge on the traffic that is provided by the FTT protocol. And third, those based on the information provided by the error counters of the NRM. Indeed, when the NRM decides that a node is not recoverable anymore because its error counter has reached another higher threshold, it will indicate its permanent disconnection from the rest. The hub will receive the corresponding indication from the NRM and will physically carry out the disconnection.

Both the NRM and the FTT master could be separated nodes connected to the hub or directly placed inside the hub, given the need that the latter has of some critical information provided by them. This decision has to be made taking into account that all of them (hub, FTT master and NRM) are single points of failure that need to be replicated. If all are together in an enhanced hub, the replication and the layout of the system could be simplified and the final cost reduced. Investigating whether such a configuration provides enough reliability will be the subject of future work.

5. Conclusions and future work

We have presented the basic ideas of a new architecture devised to provide simplified node replication on the basis of a star topology and the FTT protocol. We have also introduced a novel hardware module, called NRM, that concentrates all the complexity required to provide a reliable recovery of the nodes that cannot recover by themselves.

Our approach has a number of additional advantages. Since both the nodes (locally) and the NRM (globally) perform the voting, we have a recovery mechanism that adapts to the severity of node failures. If the failure is not so severe, the node recovers by itself using the consensus cc-vector that it calculates. If it is more severe, the NRM helps the node to recover. Moreover the NRM uses the asynchronous window of FTT for the reintegrations, in such a way that it is not necessary to reserve specific slots in the synchronous window that would be wasted in the absence of reintegrations. Finally it is important to remember that the use of FTT not only gives us support for simplifying the fault tolerance implementation but also for achieving the flexibility FTT is originally designed for, what results in an added value for the final system.

Future work will address the replication of the communication channel (provide replicated links for the nodes and tolerance to hub faults) and of the new NRM. For the latter is is possible to use schemes similar to the ones proposed in the past for FTT masters. As indicated above, the possibility of integrating the FTT master and the NRM into the hub and then replicate the resulting device will be considered, as long as it provides enough reliability. Moreover it is possible to study increasing the functionality of the NRM to include features such as the dynamic allocation of replicated task to specific nodes, depending on the changing fault tolerance needs of the application.

References

[1] L. Almeida, P. Pedreiras, and J. A. Fonseca. The FTT-CAN protocol: Why and how. *IEEE Transactions on Industrial Electronics*, 49(2), December 2002.

[2] A. Avižienis. The N-Version aproach to fault-tolerant software. *IEEE Transactions on Software Engineering*, 11(12):1491–1501, December 1985.

[3] M. Barborak, M. Malek, and A. Dahbura. The consensus problem in fault-tolerant computing. *ACM Computing Surveys*, 25(2):171–220, June 1993.

[4] M. Barranco, J. Proenza, and L. Almeida. Boosting the robustness of Controller Area Networks: CANcentrate and ReCANcentrate. *IEEE Computer*, 42(5):66–73, May 2009.

[5] M. Barranco, J. Proenza, and L. Almeida. Quantitative comparison of the error-containment capabilities of a bus and a star topology in CAN networks. *IEEE Transactions on Industrial Electronics*, 58(3):802–813, Mars 2011.

[6] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. J. Mullender, editor, *Distributed Systems*, ACM-Press, chapter 5, pages 97–145. Addison-Wesley, second edition, 1993.

[7] ISO. *International Standard 11898 – Road Vehicles – Interchange of Digital Information – Controller Area Network (CAN) for High-Speed Communication*. 1993.

[8] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Real-Time Systems. Engineering and Computer Science. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[9] J.-C. Laprie, editor. *Dependability: Basic Concepts and Terminology*. Springer-Verlag Wien New York, 1992.

[10] J. Pimentel, J. Proenza, L. Almeida, G. Rodríguez-Navas, M. Barranco, and J. Ferreira. Dependable automotive CAN networks. In N. Navet and F. Simonot-Lion, editors, *Automotive Embedded Systems Handbook*. CRC Press.

[11] S. Poledna. *Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism*. Kluwer Academic Publishers, 1996.

[12] J. Proenza, M. Barranco, G. Rodríguez-Navas, D. Gessner, G. Guardiola, and L. Almeida. The design of the CANbids architecture. In *Proceedings of the 17th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2012), Krakow, Poland*, September 2012.