Injection of Aggregated Error Flags as a Means to Guarantee Consistent Error Detection in CAN

Guillermo Rodriguez-Navas, Christian Winter and Julián Proenza
Dept. de Matemàtiques i Informàtica, Universitat de les Illes Balears, SPAIN
Contact: guillermo.rodriguez-navas@uib.es, julian.proenza@uib.es

Abstract

Although the specification of CAN states that this protocol provides data consistency, it is well know that said property does not hold for certain specific error scenarios affecting the last bits of a CAN frame, and a number of solutions have been already suggested. Morever, for a long time it has been thought that the errors affecting the initial or intermediate bits of a CAN frame cannot cause any inconsistency. In this paper we show that this assumption is false, and that such kinds of message inconsistencies are also possible for certain combinations of multiple channel errors. After describing these unreported scenarios of inconsistency, we present a mechanism that guarantees the consistent detection of said scenarios and eliminates the possibility of suffering this kind of inconsistencies. This mechanism is therefore useful for the design of highly-dependable applications over CAN.

1 Introduction

The Controller Area Network (CAN) [1] is a fieldbus communication protocol that was first devised for invehicle control applications, but it has been adopted in a much wider range of distributed embedded systems. CAN is nowadays a mature technology that has experienced a tremendous success mainly because of its error control features, low latency, network wide bus access priority and real-time response. These properties, together with the low cost of the CAN components, has turned CAN into a defacto standard for non-critical control applications.

With respect to critical applications, CAN is traditionally considered an unsuitable technology due to a number of dependability-related limitations [2]: (1) Limited data consistency; (2) Limited error containment; (3) Limited support for fault tolerance, and (4) Lack of clock synchronization. Nevertheless, several researchers state that CAN will be able to support safety-critical applications if these limitations are overcome with the proper enhancements [2].

We are currently executing a project called CANbids (*CAN-Based Infrastructure for Dependable Systems*) that purports to design, implement and validate a CAN-based infrastructure for supporting the execution of highly-dependable distributed control applications. CANbids will use as building blocks various mechanisms and enhancements intended to overcome the aforementioned CAN dependability limitations. Many proposals related to CANbids are already available in the literature [3].

This paper reviews the first of the dependability limitations of CAN, which is the lack of data consistency, describes a new (unreported) potential cause of inconsistenciy in CAN and proposes an adequate solution. This work is therefore complementary to the proposal discussed in [4].

2 Background

In this section we introduce the work carried out by several researchers in order to cope with the inconsistency scenarios of CAN.

2.1 Fault model

The fault tolerance mechanisms of CAN are intended to tolerate transient channel faults, i.e. faults suffered by the transmission medium and the transceivers. Such faults manifest as changes of the bit values transmitted, and may in fact affect one or more bits.

It is important to remark that channel errors usually have local nature, what means that some of the nodes may be affected by an error while others may not. This constitutes a severe impairment to data consistency, because the detection of errors can only be performed locally and is hence intrinsically inconsistent. However, the approach followed in CAN for guaranteing the consistent detection of channel errors is to make each node detecting a (probably local) error inject more errors in the network, such that all nodes will eventually detect at least one error and reject the frame. This technique is called CAN *error signaling*.

2.2 Error signaling and data consistency in CAN

The error signaling mechanism of CAN varies depending on the operational mode of the CAN controller. These operational modes are three, namely, *error active* state, *error passive* state and *bus-off* state. A CAN controller changes from one state to the other according to the fault-treatment strategy implemented by CAN, which is as follows. Every CAN controller has two counters of channel errors, one for errors during transmissions and another one

 doi:10.1109/ETFA.2011.6059172

for errors during reception. These counters are increased according to certain policies, whenever a channel error is detected. If one of these counters reaches a certain threshold, the node is considered potentially dangerous (yet not faulty) for the communication and enters the error passive state. If more errors are detected and the counters keep increasing and reach a higher threshold then the CAN controller is considered faulty and enters the *bus-off* state, in which the node does not take part in the communication anymore [1].

The error signaling mechanisms associated to the error active state and error passive state are different from each other. However, it is known that the error signaling performed by nodes being in the error passive state does not guarantee data consistency [5] and, for this reason, the error passive state is generally not allowed in applications that require said property. Due to this, and without losing generality, in this work we will assume that all CAN nodes stay in the error active state.

In the error active state, each node detecting an error will signal this situation to the rest of nodes by sending what is called an *error flag*. This flag starts the bit after the error that was detected. An error flag consists of six consecutive dominant bits. This flag violates CAN protocol rules, e.g. it destroys the bit fields requiring fixed form producing a form error. As a consequence all other nodes detect an error condition too and start transmitting an error flag as well. After transmitting an error flag, each node sends recessive bits and monitors the bus until it detects a recessive bit. Afterwards, it starts transmitting seven more recessive bits. The eight recessive bit chain resulting on the bus is called *error delimiter*. This error delimiter together with the superposition of error flags contributed from different nodes constitute what is called an *error frame*. After the error frame transmission, the frame that was being sent is automatically retransmitted.

In the presence of a single channel error, this mechanism guarantees the consistent rejection of any erroneous frame. However, as explained in the next subsection, several scenarios have been reported in which the combination of two or more errors may lead to an inconsistent rejection of a frame.

2.3 Reported causes of inconsistency and solutions

Rufino *et al.* identified [5] some specific scenarios in which some nodes receive a frame that some others never do. This situation is called an *inconsistent message omission* (IMO). The same authors proposed a set of protocols to be executed on top of CAN to solve the problem [5]. All these protocols require the transmission of at least an additional frame for each frame that would have been transmitted in the network, even if this frame was consistently transmitted, and therefore cause a significant communication overhead.

Afterwards, Livani *et al.* [6] presented their SHAdow REtransmitter (SHARE): a mechanism to be included in the network as a regular node and that is able to detect the bit pattern described by Rufino *et al.*, and retransmit the potentially inconsistently received frame. This approach had the significant advantage of only requiring the transmission of extra frames in case an inconsistency was possible.

In a later analysis, new scenarios of inconsistent communication were identified in which both CAN, the proposed higher-layer protocols and SHARE fail [7]. Those scenarios are characterized by the presence of multiple bits affected by errors in the channel.

An alternate way of dealing with these scenarios was proposed in [4]. This proposal suggests the use of a mechanism called CANsistant (*CAN Assistant for Consistency*), which is inspired by SHARE and works similarly, but is able to identify the scenarios of [7] as well.

It is important to remark that all the reported causes of inconsistency have something in common: they may only occur as a consequence of errors at the end of a frame transmission. The bottom problem is that in CAN the error globalization must be performed before any of the nodes has accepted the erroneous frame. Therefore, the errors happening within the last bits of a frame (particularly, within the EOF field) have a very short time for being globalized: the length of just a few bits. Thus, if the subsequent bits are also affected by errors that mask the error signaling for certain nodes, then the frame will be accepted by these nodes while being rejected by the others. In such cases, the only way to recover from the inconsistency, and hence avoid an IMO, is by message retransmission. The protocols of Rufino *et al.* perform the retransmissions preventively, whereas SHARE and CANsistant are more selective and only retransmit a frame when it is required, i.e. whenever one of the potentially dangerous error scenarios is detected.

3 New causes of inconsistency

Applying the reasoning discussed in Section 2, for a long time it has been thought that the errors affecting the initial or intermediate bits of a CAN frame cannot cause any inconsistency. In other words, it has been thought that any error signaling that starts before the EOF will always provoke errors that are consistently detected by all the (non-faulty) CAN nodes. Nevertheless, we have identified some scenarios with multiple bits affected by errors in which this is not fulfilled. In order to find out these scenarios we used a simulation tool called CANfidant [8], which simulates the behavior of several CAN controllers and allows injection of both global and local errors in the channels of these controllers. With the help of this tool, it is possible to observe the response of the CAN controllers under complex combinations of communication errors.

For illustration, Figure 1 depicts one of the inconsistency scenarios detected. In this scenario, the first node (Node 0) is the transmitter (frame with Id= 0, Data=162Dec); the second node (Node 1) detects a local channel error within a bit of its CRC (a stuff error, more precisely) and immediately injects an error flag. But Node 0 and Node 2 both suffer from another channel error that

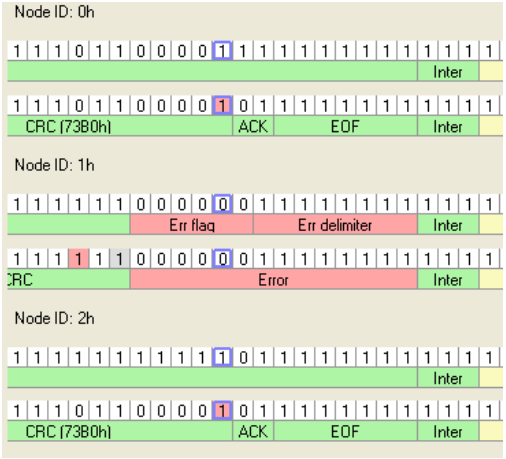

Figure 1: An inconsistency scenario with 2 channel errors

prevents the detection of the error flag and they accept the frame anyways.

Although it is not possible to discuss here all the scenarios identified with the help of CANfidant, two conditions have been found out as potential causes. On the one hand, the first error is seen by a receiver. Because if it is seen by the transmitter then it will send error flags instead of the correct bits and therefore there will be no complete frame to be inconsistently received. On the other hand, the first error detected (the actual erroneous bit could have happened before) must be a stuff error or a format error (e.g. the ACK delimiter). If it was detected only by checking the CRC, then the affected node would send an error flag in the first bit of EOF and thus all nodes would detect the problem before the last bit of EOF even in the presence of up to 5 additional errors in the first 5 errors of the error flag.

Also, we noticed that fewer errors are required for having an inconsistency if the value of the CRC ends with a long sequence of dominant bits that could, in the presence of some other channel error(s), mask the error flag as shown in Figure 1. As a final remark, it is important to say that our study has been systematic, but not exhaustive (since exhaustivity would require some kind of formal verification) and therefore some new scenarios of inconsistency might still appear. For this reason, it is important to propose a solution that addresses the source of the problem and thus may effectively deal with scenarios not yet identified.

4 Our solution

4.1 Rationale

The source of the new inconsistency scenarios is that the error flag defined by the CAN protocol is too short in the presence of multiple errors. The solution we propose is to make the error flags longer and hence increase the chances of causing errors that are noticed by all the nodes. However, there is an important issue to be taken into account: these longer error flags should not cause significant increments of the error counters of the nodes or, otherwise, the

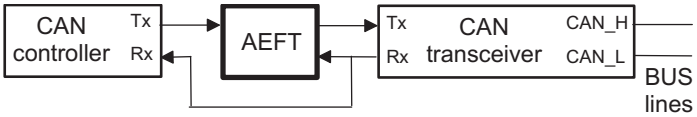

Figure 2: Location of the AEFT

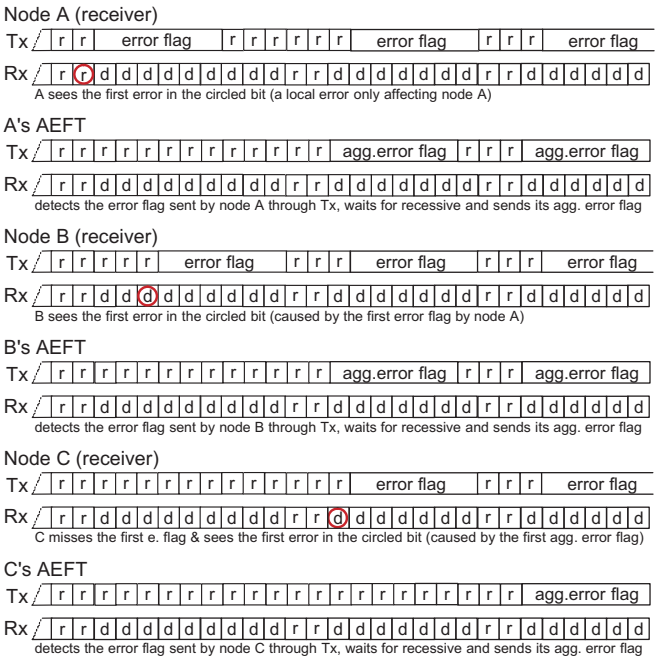

Figure 3: Example of signaling with aggregated error flags

error signaling might unfearly lead some nodes to the bus-off state. In order to avoid this problem, instead of having a single longer error flag, what we propose for error signaling is using a number (m) of consecutive error flags, but making sure that there are some recessive bits between them. For designating this new error signaling mechanism we have coined the term *aggregated error flag* (AEF).

In order to make this mechanism compatible with any kind of CAN controller, we propose to attach a specifically designed hardware device to each CAN node, and to make this device inject the AEF when required. This device is called *AEF Transmitter* (AEFT) and its location in a CAN node is shown in Figure 2.

Figure 3 shows an example of the use of aggregated error flags for signaling. The figure shows the Tx and Rx signals of three CAN controllers and the contribution of their corresponding AEFTs. In this case, the consistency is guaranteed despite the existence of three channel errors (marked with circles).

4.2 Architecture

The internal structure of the AEFT is depicted in Figure 4. It is constituted by five modules. The module *bit rate prescaler* (BRP) is basically a frequency divider that provides a clock ticking at the bit rate. The module *synchronizer* implements the bit synchronization defined by the CAN standard [1] and provides the sampling point of

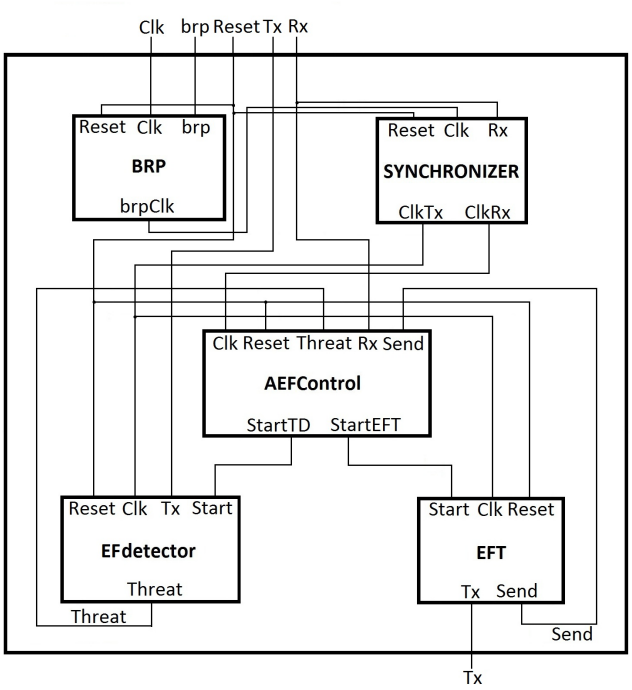

Figure 4: Block diagram of the AEFT

each bit, since this is required by the other modules. The module *AEFControl* is the control unit of the AEFT and is the one who decides when to inject an AEF. This module uses the module *EFdetector* for detecting any error flag sent by the corresponding CAN controller and the module EFT (Error Flag Transmitter) for injecting each error flag that is required for conforming the AEF.

The error-flag detection is enabled by EFTControl with the signal *startTD*. Whenever EFdetector sees that the CAN controller is starting an error flag, it is notified with the signal *threat* to AEFControl. Then, AEFControl disables the detection of EF and uses the signal *startEFT* to instruct EFT to inject the first error flag of the AEF. The signal *send*, issued by EFT, indicates that the complete error flag has been transmitted and the following one can be started.

The number of error flags actually injected in every AEF (m) is configurable, because it depends on the fault conditions assumed; typically a more aggresive environment would require a higher m. For controlling this, AEFControl counts the number of activations of *send*, and once the value m is reached no more EF is injected.

5 Conclusion and future work

In this work we studied the different cases in which a frame can be inconsistently received by the nodes of a CAN network. We reviewed the already reported scenarios of inconsistency due to channel errors in the EOF and identified new scenarios of inconsistency, caused by multiple channel errors in the intermediate bits of the frame.

In order to eliminate this new source of inconsistencies, we suggest a mechanism based on aggregations of error flags, which guarantees the consistency of any frame broadcast. This proposal is fully compatible with the solutions that address the other inconsistency problems, such as CANsistant [4]. Additionally, we have proposed the architecture of a device, the so-called Aggregated Error Flag Transmitter (AEFT), for implementing this specific signaling. Such a device can be developed with a PLD and can be easily incorporated onto any CAN node. At present, we are setting up a CAN infrastructure that incorporates CANsistant as well as nodes with AEFT; our aim is to show the usefulness of our solution as an efficient and effective way to prevent inconsistent message omissions.

Acknowledgment

This work was supported by the Spanish Science and Innovation Ministry with grant DPI2008-02195, FEDER funding.

References

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

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

[3] SRV, *CANbids project website*. (http://srv.uib.es/project/12).

[4] J. Proenza and E. Sigg, "A first design for CANsistant: A mechanism to prevent inconsistent omissions in CAN in the presence of multiple errors," in *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, sept. 2009, pp. 1 –4.

[5] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues, "Fault-tolerant broadcast in CAN," in *Proceedings of the IEEE 28th Int. Symp. Fault-Tolerant Computing. FTCS-28. Munich (Germany)*, June 1998.

[6] M. Livani, "SHARE: A transparent approach to fault-tolerant broadcast in CAN," in *Proceedings of the 6th International CAN Conference*, 1999.

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

[8] G. Rodriguez-Navas, J. Jimenez, and J. Proenza, "An architecture for physical injection of complex fault scenarios in CAN networks," in *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference*, vol. 2, sept. 2003, pp. 125 – 128 vol.2.