MajorCAN: A Modification to the Controller Area Network Protocol to Achieve Atomic Broadcast

Julián Proenza and José Miro-Julia
Dept. de Matemàtiques i Informàtica, Universitat de les Illes Balears, Palma de Mallorca, SPAIN
dmijpa0@ps.uib.es (J. Proenza) and joe@ipc4.uib.es (J. Miro-Julia)

Abstract

It has already been reported that the CAN protocol produces data inconsistencies in some scenarios that violate the Atomic Broadcast properties. It has been proposed a set of higher level protocols to achieve Atomic Broadcast on CAN based systems. This approach causes considerable overhead. We propose a more efficient solution through small modifications to CAN. Our first proposal of modified protocol is called MinorCAN. Unfortunately we have identified new inconsistency scenarios in which both CAN, MinorCAN and the mentioned higher level protocols still violate the Atomic Broadcast properties. Inspired in the MinorCAN ideas we propose a protocol, called MajorCAN, that really implements Atomic Broadcast.

1. Introduction

The Controller Area Network (CAN) protocol, a fieldbus first developed for automotive applications, is widely used in the automation industry as well. The main reason for its success is its real-time and dependable behaviour. Among the dependability properties, the specification of this standard [5] claims that CAN presents *data consistency*. This means that within a CAN network it is guaranteed that a frame is either simultaneously accepted by all nodes or by none. This roughly corresponds to the *Atomic Broadcast* definition, and has lead many authors to assume that CAN provides this service of capital importance in fault-tolerant and real-time distributed systems. But it is well known that CAN does not always accomplish the pretended data consistency. One impairment is the presence of a special state in the CAN nodes, called *error-passive*. A CAN node in the *error-passive* state signals the transmission errors in a way that cannot force the other nodes to see the error. If this node is the only one suffering the error an inconsistency appears in the network. Many authors have proposed avoiding this state to improve the dependability in CAN based systems [2, 4, 10]. Even assuming no node is error-passive, CAN presents other inconsistency problems. On the one hand it may be the case that a node receives a frame twice (*double reception of frames*) [11]. On the other hand Rufino *et al.* have recently identified a much more dangerous situation: scenarios in which some nodes receive a frame and some others do not. In [10], they introduce specifically designed higher level protocols to cope with these inconsistencies.

In this paper we identify new scenarios, in which both CAN and the mentioned higher level protocols [10] still violate the Atomic Broadcast properties. Fixing these problems using higher level protocols introduces considerable overhead. Many distributed control systems, especially in automotive applications, have to be constructed with minimum memory and CPU power requirements [1]. So we take the more efficient approach of modifying the CAN protocol. We propose two modified protocols. The first and more simple is called MinorCAN, and works in the scenarios introduced in [10] but not in the new scenarios. We explain MinorCAN because it helps to understand the second protocol. This second is called MajorCAN, and works in all the identified scenarios implementing Atomic Broadcast in the presence of up to m randomly distributed errors per frame.

2. CAN and data consistency

The CAN protocol does not implement Atomic Broadcast. In this work we use the same adaptation of the definition of Atomic Broadcast used in [10]. This definition assumes that the nodes can only present *benign failures*, that includes *crash*, *omission* and *timing failures* [3]. A node presenting a crash failure stops prematurely doing nothing from that point on. A node presenting omission failures either intermittently omits to send/receive messages it was supposed to send/receive, or stops prematurely doing nothing from that point on, or both. Finally a node presenting timing failures can fail in one or more of the following ways: it commits omission failures, its local clock drift exceeds the specified bound (*clock failure*), or it violates the bounds on the time required to execute a step (*performance failure*).

With these failure assumptions, a protocol presents

Atomic Broadcast when it exhibits the following properties:

AB1 - Validity: if a correct node broadcasts a message[1], then the message is eventually delivered to a correct node.
AB2 - Agreement: if a message is delivered to a correct node, then the message is eventually delivered to all correct nodes.
AB3 - At-most-once Delivery: any message delivered to a correct node is delivered at most once.
AB4 - Non-triviality: any message delivered to a correct node was broadcast by a node.
AB5 - Total Order: any two messages delivered to any two correct nodes, are delivered in the same order to both nodes.

The first pointed impairment to Atomic Broadcast is the existence of the error-passive state in CAN nodes. CAN protocol provides several error detection mechanisms. When a node in the so called *error-active* state detects an error, this is signalled to the rest of the nodes and the transmitter starts an automatic retransmission of the erroneous frame. Any CAN node has a transmit error counter and a receive error counter. Whenever any of these counters reaches the value 127 the node enters in the error-passive state. When in this state, the node signals the errors in a way that cannot force the transmitter to retransmit the incorrectly received frame [11]. As an example of the consequences this can have, let us consider the case of an error-passive node being the only one to detect an error in a received frame. The transmitter will not be forced to retransmit and the error-passive node will be the only one never receiving the message. In this case property AB2 is not satisfied.

Many authors have proposed avoiding this state to improve the dependability in CAN based systems [4, 2, 10]. This is easily achieved using a signal provided in most modern CAN circuits, called the error warning notification. This signal is generated when any error counter reaches the value 96, that is considered as an indication of a heavily disturbed bus [7]. This is a good point to switch off the node before it goes into the error-passive state, assuring that every node is either helping to achieve data consistency or disconnected.

In the rest of this section, we will describe other inconsistency problems reported in the literature that are more difficult to solve. All these problems appear even if no node is in the error-passive state. Prior to the description of the problems, we explain the related CAN protocol concepts.

2.1. CAN protocol fundamentals

As justified above the error-passive state must be avoided to achieve data consistency. So in the CAN networks we consider during the rest of the paper no node is in that state. All the fundamentals we are going to explain in this section correspond to nodes in the error-active state.

[1] In CAN a *message* is the information unit that can be transmitted in a single *frame*, but that can also need several frame retransmissions to be eventually transmitted

A CAN bus can take one of two values: *dominant* or *recessive*. The dominant value represents the logical '0' and the recessive value represents the logical '1'. If two nodes simultaneously transmit two bits with different values, the resulting bus value will be dominant.

The claimed data consistency in the CAN protocol is achieved thanks to its special error detection and error signalling mechanisms. CAN presents five error detection mechanisms that lead to five different kinds of errors called bit error, stuff error, CRC error, acknowledgment error and form error. Any node detecting an error by any of these mechanisms 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 was detected. An error flag consists of six consecutive dominant bits. This flag violates CAN protocol rules, *e.g.* 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. This simple mechanism allows the globalization of local errors and is supposed to provide data consistency.

All the data frames finish with eight recessive bits that is the same as the error delimiter of an error frame. So all frames, error or data, finish with the same bit pattern to permit node synchronization. The last seven of the eight recessive bits form the end of frame field (EOF). After the EOF a chain of recessive bits of variable length, called interframe space separates a frame from the next one. During the first three bits of this interframe, slow nodes can introduce an extra delay between frames. This is done using a special flag, called overload flag, that has the same format as the error flag. The rest of the nodes react exactly as with error flags by sending their own overload flags and the overload delimiter. So if an error is detected in these bits all the nodes will consider it as an overload condition.

In the last bit of the EOF the behaviour in the presence of errors is special to cope with specific error situations. If a transmitter detects an error in this bit it will handle it as if it was detected in any other bit: an error flag will be started in the next bit, frame transmission will be considered erroneous and the frame will be retransmitted. If a receiver detects an error in this last bit it will accept the frame as correct and, instead of an error flag, it will generate an overload flag. The reason of this behaviour is illustrated by the scenario in Fig. 1a. A set of receiving nodes, called the

X set, detect an incorrect dominant value in the last bit of the EOF, while the transmitter and another set of receiving nodes, called the Y set, see a correct recessive bit. The nodes of X start transmission of an overload flag in the bit after the error. The rest of nodes see the first dominant bit of the overload flag at the first bit of the interframe space and start the transmission of their overload flags as well. So both transmitter and the nodes belonging to Y will consider the frame as correctly transmitted. Thanks to the last bit rule the nodes belonging to X will also accept the frame and consistency will be achieved.

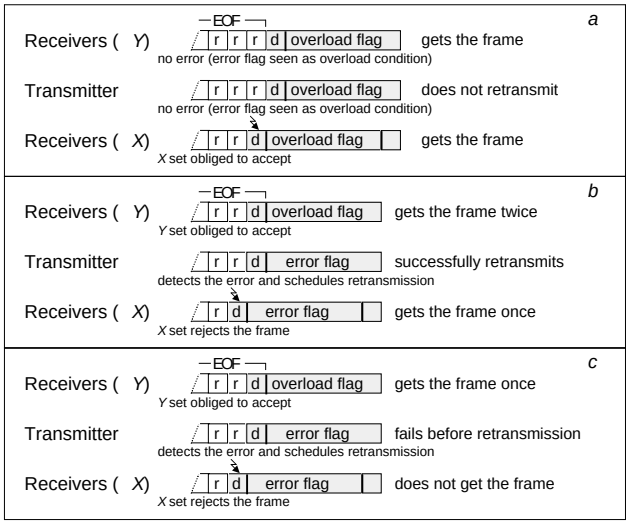

Figure 1. Error scenarios discussed [10]

2.2. Inconsistency scenarios

Unfortunately this last bit behaviour is also responsible for nodes receiving twice the same frame—the mentioned *double reception of frames*. Fig. 1b shows a scenario where this happens. A disturbance corrupts the last but one bit of the EOF of the nodes belonging to X. In the next bit these receivers start the transmission of an error frame. The dominant first bit of this error flag is seen by the transmitter and the nodes belonging to Y as an error in the last bit of their EOF. The nodes belonging to X will reject the frame, the transmitter will retransmit it, and the nodes belonging to Y will accept the frame following the last bit rule. As a result the nodes belonging to Y will receive the frame twice.

The double reception of frames is a well known phenomenon that has lead to a set of common recommendations [11], *e.g.* not to transmit messages that toggle the state of the receivers. Beyond the double reception, Rufino *et al.* have identified [10] new error scenarios in which the last bit behaviour produces *inconsistent message omissions* (IMO): some nodes receive a frame and others never do. This is illustrated in Fig. 1c. This is essentially the same case as in Fig. 1b, the difference being that in Fig. 1c after the frame's first transmission the transmitter suffers a failure that impedes the retransmission of the frame. So the nodes belonging to Y receive the frame whereas those of X do not.

Rufino *et al.* have evaluated the probability of occurrence of the inconsistent message omissions [10]. They have obtained values of the order of 10^{-6} incidents/hour. Those are larger than the reference value of 10^{-9} incidents/hour, the safety number of the aerospace industry [8], that is being adopted by the automotive industry as well [6].

According to [10] the described inconsistencies are the cause that CAN protocol properties differ from the Atomic Broadcast requirements. To establish the CAN properties they make some assumptions on the failure semantics of the system. Each node consists of a CAN controller and a process that communicates with the processes in the other nodes. The processes are considered fail-silent (they can show either crash or omission failures). The CAN controllers avoid the error passive state and never reach the number of errors necessary to enter the bus-off state (crash) in an interval of reference T_{rd}. Finally there is no permanent failure of shared network components (e.g. medium partition). Given these assumptions the CAN properties are:

CAN1 - Validity: if a correct node broadcasts a message, then the message is eventually delivered to a correct node.

CAN2 - Best-effort Agreement: if a message is delivered to a correct node, then the message is eventually delivered to all correct nodes, if the transmitter remains correct.

CAN3 - At-least-once Delivery: any message delivered to a correct node is delivered at least once.

CAN4 - Non-triviality: any message delivered to a correct node was broadcast by a node.

CAN5 - Total Order not ensured.

CAN6 - Bounded Inconsistent Omission Degree: in a known time interval T_{rd}, inconsistent omission failures may occur in at most j transmissions.

CAN6 establishes j as a measure of the probability of the inconsistent message omissions. CAN5 is explained if we consider the following example. If a frame, labeled A, is scheduled for retransmission when some nodes have received it and some others have not, a second frame, labeled B, could gain the arbitration to the retransmission. The nodes having received A the first time will see the order A, B, A, while the others will see B, A. Proofs for the rest of the CAN properties are straightforward.

Comparing these properties with the ones appearing at the Atomic Broadcast definition in Section 2, it is clear that CAN does not present Atomic Broadcast. There are two possible approaches to obtain Atomic Broadcast on CAN based systems. The first one is to add higher level protocols specifically designed to recover from the inconsistent message omissions. The second one is to modify the CAN protocol to fix the problems with the minimum overhead.

The solution proposed in [10] follows the first approach. Specifically they introduce three protocols: EDCAN, RELCAN and TOTCAN. In EDCAN all the receivers retransmit the message after reception to overcome transmitter failures. This protocol satisfies all the Atomic Broadcast properties except Total Order, thus providing *Reliable Broadcast* [3]. In RELCAN the same properties are satisfied taking a more efficient approach. The transmitter sends a CONFIRM message after the successful transmission of the main message. Only in case the CONFIRM does not reach the receivers in a specified timeout they start the retransmission of the main message. Finally TOTCAN satisfies all the Atomic Broadcast properties, including Total Order. Each time a receiver gets a duplicate of a message, it puts it at the tail of a queue. The transmitter sends an ACCEPT message after the successful transmission of the main message. When the receivers get the ACCEPT message, they fix the position of the message in the queue. In case the ACCEPT message does not reach the receivers in a specified timeout, they remove the corresponding message from the queue. A detailed description of these protocols can be found in [10].

This approach is cheap because it does not require to make any change in the CAN controllers. But it is clear that it also wastes communication bandwidth, a scarce resource in CAN, requires extra program memory space and implies a computational overhead for the nodes. Many control systems, especially in automotive applications, have to be constructed with minimum memory and CPU power requirements [1]. This justifies to propose changes in the standard CAN protocol.

3. A first solution: the MinorCAN protocol

We propose to slightly modify the CAN protocol to obtain a new protocol called MinorCAN. MinorCAN provides Atomic Broadcast avoiding both double reception of frames and inconsistent message omissions in the scenarios described in Fig. 1. This is done using resources already available in standard CAN. Furthermore the performance achieved by MinorCAN is better than that of the standard CAN protocol. Although we have warn that MinorCAN is going to fail in the new scenarios we are going to introduce in Section 4, it is important to describe it because MajorCAN is inspired in the same ideas.

The main modification in MinorCAN is the behaviour of the protocol when processing errors detected in the last bit of EOF. For errors detected in the bits previous to the last bit of EOF, frames will be always rejected/retransmitted as in standard CAN. For those detected in the bits following the last bit of EOF, frames will be always accepted/not retransmitted also as in standard CAN. Finally, for those detected in the last bit of EOF, both receivers and transmitter will decide to reject/retransmit or to accept/do not retransmit the frame according to the same criterion: If the node x, either transmitter or receiver, is the first to detect an error in the last bit of a frame then no one has yet rejected the frame or scheduled it for retransmission, so x will not do so either; but if x is the second one, some other node has already rejected the frame or scheduled it for retransmission so x must do the same.

Using this simple criterion consistency is assured and performance is improved: in MinorCAN if the transmitter detects an error in the last bit of EOF retransmission might be avoided, depending on the other nodes, while in CAN it always takes place.

The behaviour described above can be easily implemented using the *Primary_error* message that is interchanged between the Medium Access Control (MAC) sublayer and the Fault Confinement Entity (FCE) in the CAN protocol. This message signals that the MAC sublayer has detected a dominant bit after sending an error flag, corresponding to the last bit of the error flag (or overload flag) of another node. This indicates that the MAC sublayer has detected a primary error and not an error that is caused by the error flag of another node. In MinorCAN a node that detects an error in the last bit of EOF signalled by the *Primary_error* message accepts/does not retransmit the frame, while a node that detects an error in the same position but not signalled by *Primary_error* rejects/retransmits the frame. Due to this implementation if all the nodes detect an error in the last bit of EOF, MinorCAN will consider all the errors not primary and the frame will be unnecessarily but consistently retransmitted/rejected.

MinorCAN works properly in the critical scenarios described before as is illustrated in Fig. 2. Furthermore, it can be proven, by checking all the possible cases, that MinorCAN achieves consistency in the event of a permanent failure of any of the nodes after the bit error detection. Finally it is important to note that the *Primary_error* message is not a CAN message interchanged between the CAN nodes, but a signal internal to the CAN controller and generated within the CAN frame transmission time. This means that this solution introduces no overhead, while any of the higher level protocols proposed in [10] implies the transmission of more than a CAN frame per message.

4. New inconsistency scenarios

One of our main contributions is that we have identified new error scenarios leading to inconsistent message omissions. In these situations neither standard CAN nor MinorCAN nor the mentioned higher level protocols fulfill the Atomic Broadcast requirements. Let us consider the case in Fig. 3a for standard CAN. As in Fig. 1b, a disturbance corrupts the last but one bit of the EOF of the nodes belonging to X. In the next bit these receivers start the transmission

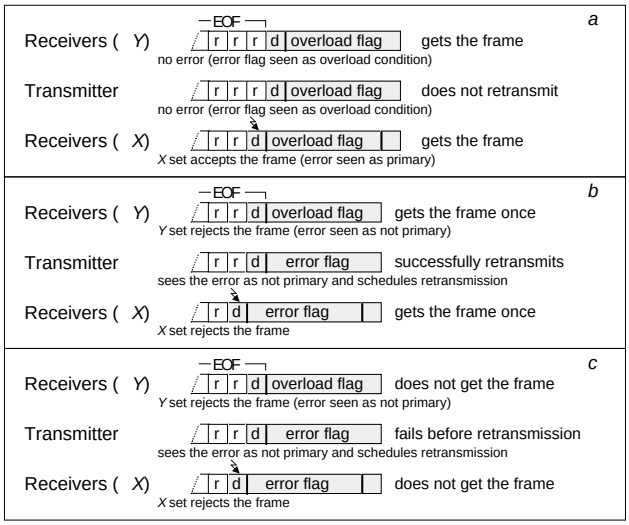

Figure 2. Achieving consistency in MinorCAN

of an error frame. The dominant first bit of this error flag is seen by the nodes belonging to Y as an error in the last bit of their EOF, while, in this case, the transmitter can not see the error flag in the last bit of EOF due to an additional disturbance in that bit. The nodes belonging to X will reject the frame, the nodes belonging to Y will accept the frame following the last bit rule, and the transmitter will consider it as correct, and no retransmission will be performed. As a result an inconsistent message omission will happen.

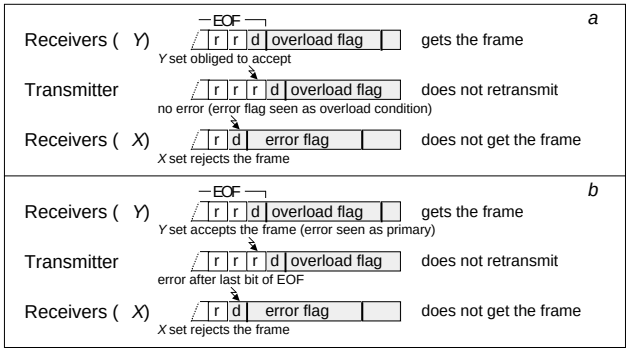

Figure 3. New scenario (CAN and MinorCAN).

Note that an additional disturbance in one single bit is enough to produce the inconsistency. This scenario produces the same result in MinorCAN as can be seen in Fig. 3b. After the disturbance in the last but one bit of EOF of the nodes belonging to X, these reject the frame and start the transmission of an error flag. The nodes belonging to Y detect an error in the last bit of EOF, transmit an overload flag, sample the channel after this transmission, and decide they have detected a primary error. Thus they accept the frame. The transmitter can not see the error until the bit after the EOF, so it assumes everybody has the frame and does not retransmit it.

Among the higher level protocols proposed in [10] to solve the inconsistency problems, only EDCAN operates properly in the new scenarios. But it is important to remember that EDCAN does not provide Atomic Broadcast anyway—it does not guarantee Total Order. Moreover EDCAN is precisely the one which has a lower performance, as it needs at least one retransmission per frame (each frame is transmitted at least twice). The rest of the protocols do not work because they only perform recovery actions in case the transmitter fails, and as we have proven, inconsistencies can appear even if the transmitter does not fail.

To justify the importance of these new scenarios it is necessary to estimate their probability of occurrence in standard CAN. The model proposed in [10] is not suitable for our purposes, because it uses as a basic parameter the *bit error rate* (ber). This parameter indicates the probability for a bit to be erroneous in any part of the network. In our case it is necessary to model the probability for a node to be affected by the error in its particular view of the bit. According to Charzinski[1] the spatial distribution of errors in the network is modeled by p_{eff}, the probability for a bit error occurring somewhere in the network to appear at a certain node. For each bit time the following equation holds:

$$p_{eff} = \mathrm{P}\{\mathrm{A} \mid \mathrm{B}\}$$
$$\text{where A = error affects considered node; and}$$
$$\text{B = error somewhere in the network} \qquad (1)$$

Following the same assumptions of Charzinski, we will consider the effectivity of bit errors to be randomly distributed over the nodes with $p_{eff} = 1/N$, where N is the number of nodes. If we identify $ber = \mathrm{P}\{\text{error somewhere in the network}\}$ it is easy to derive:

$$\mathrm{P}\{\text{error affects considered node}\} = \frac{1}{N}\, ber \qquad (2)$$

We will consider for ber the same constant values as in [10] in order to have a common reference. Expression 2 calculates the value of the new parameter we need for our model. We will call this parameter ber^*. So:

$$ber^* = \frac{1}{N}\, ber \qquad (3)$$

Now we can obtain the expression of the probability of occurrence per frame of the scenario presented in Fig. 3a. In this expression τ_{data} is the number of bits in a frame. The expression is obtained considering all the cases in which at least one of the receivers is affected by an error in the last but one bit of the frame, while the rest of them—at least one—is not affected. In all the cases the transmitter suffers

an error in the last bit that impedes it to detect the error flag.

$$\begin{aligned}&\mathrm{P}\{\text{new scenario in a frame}\} = \\ &= \textstyle\sum_{i=1}^{N-2} \binom{N-1}{i} \left((1-ber^*)^{\tau_{data}-2}\ ber^*\right)^i \ \times \\ &\qquad \left((1-ber^*)^{\tau_{data}-1}\right)^{N-1-i} \ \times \\ &\qquad\quad (1-ber^*)^{\tau_{data}-1}\ ber^* \end{aligned} \tag{4}$$

We can estimate the number of inconsistencies that these new scenarios produce in an hour by multiplying by the number of frames transmitted in an hour. Again we use the same data as in [10] for comparison purposes. So we consider a network at 1 Mbps, with 32 nodes, an overall load of 90% and frames with a length of $\tau_{data} = 110$ bits. We calculate the number of inconsistencies for bit error rates (ber) going from aggressive to benign environments. In Table 1 the results of these calculations are summarized. Column IMOnew/hour shows our results for the new scenarios, column IMO/hour shows the maximum values for the scenarios described in Fig. 1c obtained by Rufino *et al.* using their own model and, to make a fair comparison, column IMO*/hour shows the corresponding values for the scenarios in Fig. 1c obtained by our model based on the ber^* parameter. For these last set of values the following expression has substituted expression 4 for the probability of occurrence per frame:

$$\begin{aligned}&\mathrm{P}\{\text{old scenario in a frame}\} = \\ &= \textstyle\sum_{i=1}^{N-2} \binom{N-1}{i} \left((1-ber^*)^{\tau_{data}-2}\ ber^*\right)^i \ \times \\ &\qquad \left((1-ber^*)^{\tau_{data}-1}\right)^{N-1-i} \ \times \\ &\qquad\quad (1-ber^*)^{\tau_{data}-2}\ (1-e^{-\lambda\Delta t}) \end{aligned} \tag{5}$$

where $(1 - e^{-\lambda\Delta t})$ represents the probability for the transmitter to crash impeding the retransmission of the frame. For Δt we will take the value of 5 ms as in [10] and for λ, 10^{-3} failures per hour that is the maximum value considered in [10].

ber	IMOnew/hour (Fig. 3a)	IMO/hour (Fig. 1c)	IMO*/hour (Fig. 1c)
10^{-4}	8.80×10^{-3}	3.94×10^{-6}	3.92×10^{-6}
10^{-5}	8.91×10^{-5}	3.98×10^{-7}	3.96×10^{-7}
10^{-6}	8.92×10^{-7}	3.98×10^{-8}	3.96×10^{-8}

Table 1. Probabilities of the scenarios

Analyzing the results in Table 1, it is obvious that the model we have introduced based in ber^* permits to reproduce the results obtained in [10] for the old scenarios. This legitimates the comparison of the results obtained for each kind of scenarios. Thus it is clear that the new scenarios have probabilities larger than the reference value (10^{-9}), and also larger than the previously reported scenarios. This justifies to review the CAN properties. Making the same failure assumptions as in Section 2.2 we propose a definitive set of CAN properties that are listed below. Only properties that have suffered a modification when considering the new scenarios are listed and marked with a '.

> **CAN2' - Agreement not guaranteed**: if a message is delivered to a correct node, then the message may never be delivered to all correct nodes, even if the transmitter remains correct.
> **CAN6' - Bounded Inconsistent Omission Degree**: in a known time interval T_{rd}, inconsistent omission failures may occur in at most j' transmissions.

CAN2' is immediately justified by the scenario in Fig. 3a. In that scenario, discussed at the beginning of this section, the transmitter can not see at the last bit of EOF the error flag transmitted by some receivers, and then considers the frame correctly transmitted while some receivers reject it. In CAN6' j' is larger than previous j: the probability of the inconsistent omission failures is larger when the new scenarios are considered.

5. The CAN modification for Atomic Broadcast: the MajorCAN protocol

As seen in the previous section, the probability of multiple errors affecting the nodes within the same frame is considerably high. Any modification we propose to the original protocol must consider this possibility. We first must decide how many errors we must be able to cope with. We denote this number as m and of course it must be larger than 2, as with 2 errors the scenario that leads to property CAN2' (Fig. 3a) could happen. Our proposal is $m = 5$ because standard CAN uses a CRC code that allows the detection of up to 5 randomly distributed bit errors [11]. Therefore it makes sense to guarantee Atomic Broadcast at the same level. In any case, this decision strongly depends on the ber value. If ber is larger then larger values of m should be considered. So the new protocol we propose is designed to be parametrisable in m to make the upgrade simpler. We call this new protocol MajorCAN [9] and we will use the notation MajorCAN$_m$ to indicate the value chosen for m in specific cases.

MajorCAN is based in the mechanism described before for MinorCAN. Once again nodes detecting an error at the end of the frame must check if there are nodes detecting the error later and then accepting the frame, and nodes detecting the error later must notify that they accept the frame through their error flags. The main changes to cope with multiple errors are two. First that this notification of acceptance must be performed with an extended error flag to admit errors during the notification. And second that to avoid situations like the one illustrated in Figures 3a and 3b, not only the nodes detecting an error in the last bit of the CAN's EOF

must check if other nodes are notifying the acceptance of the frame, but also nodes detecting errors before that.

Before describing in detail the structure of the extended error flags, and the way the nodes check if others are accepting, it is necessary to realize that the structure of the EOF can also be modified to better adapt to the new approach, avoiding an excessive prolongation of the MajorCAN frames. The EOF contains no relevant data. If a frame contains errors only in the bits of the EOF it is a correct frame and could be accepted. Whether it is accepted or not is just a question of agreement between the different nodes. The MajorCAN EOF field will be divided in two sub-fields. Nodes detecting an error in the second sub-field must accept the frame and notify through an extended error frame, while nodes detecting an error in the first sub-field must check if there are nodes detecting the error in the second one and then accepting the frame.

The number of bits of the two sub-fields of the MajorCAN's EOF must be chosen to be minimum. According to the CAN specification [5] whenever a CRC error is detected, transmission of an error frame starts at the bit following the ACK delimiter—that is at the first bit of the EOF. This means that a node starting an error flag in the first bit of EOF should never accept/do not retransmit that frame. To achieve agreement between nodes, it must be impossible for other nodes to first detect this error flag in a bit of the second sub-field which would produce the acceptance of the frame. As in the worst case $m-1$ additional errors can delay the detection of the error flag, the first sub-field of EOF must have m bits. On the other hand if the node that first detects the error does it in the mth bit of EOF—the last bit of the first sub-field—and there are only two nodes in the network, it is necessary that the other node notifies the acceptance of the frame with an extended error frame. Given that additional errors can delay the detection of the error flag generated by the first node up to $m-1$ bits, this error flag can be detected by the other node at the $2m$th bit that will be, in consequence, the last bit of the second sub-field of EOF. So the second sub-field must also have m bits.

As said before, a node y detecting an error in the second sub-field of EOF must notify the acceptance through an extended error frame. This extension must be enough to permit a node x detecting an error in the first sub-field of EOF to know, in the presence of up to $m-1$ additional errors, that y is notifying acceptance. If there is only one error and x has detected it in the mth bit of EOF, y will see the x's error flag as an error condition in the $(m+1)$th. Then for x to know some other node is accepting—like y in this case—it would be enough to sample the last bit of the regular 6-bit error flag that this node would generate, that is the $(m+7)$th bit. This is essentially what MinorCAN does. If there is an additional error it could affect precisely the sampling of the $(m+7)$th bit. To cope with this, x should sample not only the $(m+7)$th bit but also the $(m+8)$th bit and the $(m+9)$th bit, and perform a majority voting on these samples. Of course this implies that y must extend two bits its error flag. In the presence of m errors, $m-1$ of them could affect the sampling. So x must sample $2m-1$ bits from the $(m+7)$th to the $(3m+5)$th, and y must extend its error flag to all these bits. If instead of having detected the error in the $(m+1)$th bit, y detected it in any of the bits from the $(m+2)$th to the $2m$th it must also extend its error flag up to the $(3m+5)$th bit to notify the rest of the nodes that it is accepting. If x detected the error in any of the bits from the 1st to the $(m-1)$th, instead of in the mth, it must also sample the bits from the $(m+7)$th to the $(3m+5)$th to know if any other node is accepting the frame.

It is important to remark that if any node detects its second error during the bits corresponding to the EOF and the extended error flags, this is not signaled with any additional error flag. Otherwise error flags of second errors could spoil the agreement process.

Fig. 4 shows the behaviour of a MajorCAN$_5$ node when an error is detected in different bits of EOF. Vertical arrows indicate the bits where the sampling is performed.

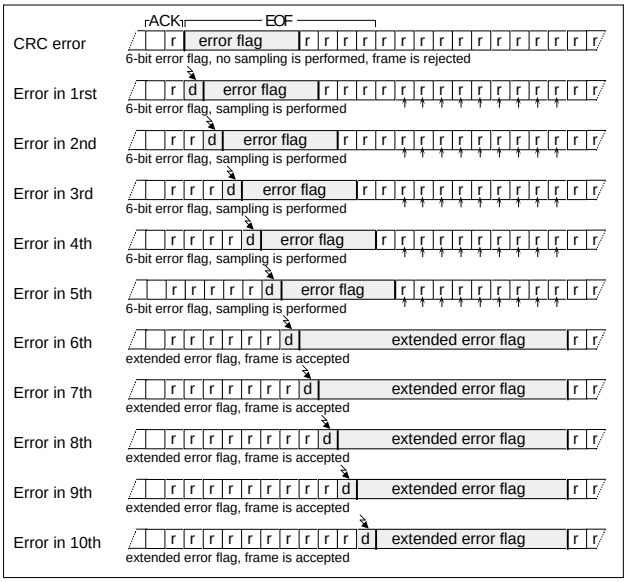

Figure 4. Behaviour of a MajorCAN$_5$ node.

To finish the description of the MajorCAN protocol it is important to mention some details. First, errors detected after the last bit of EOF will be treated as in standard CAN. Second, also as in standard CAN the MajorCAN frames finish with an ACK field of two bits followed by the EOF field described above. As the ACK field ends with a recessive bit and EOF is a sequence of $2m$ recessive bits, any MajorCAN frame finishes with $2m+1$ recessive bits. This must match the form of the error delimiter to permit node synchronization. So the MajorCAN error delimiter will be

a chain of $2m + 1$ recessive bits, instead of the 8 recessive bits of the standard CAN. This means that in the best case—when there are no errors during the EOF—the overhead introduced by MajorCAN compared with standard CAN is of $2m-7$ bits. For our proposal of $m = 5$ we have an overhead of 3 bits. In the worst case—when there are errors during the last m bits of EOF—the MajorCAN frame is extended $2m - 2$ bits more. For our proposal of $m = 5$ this means an increment of 8 bits, so a total overhead of 11 bits. Even in this last case, the overhead is negligible compared with any of the higher level protocols proposed in [10] that require the transmission of more than a CAN frame per message.

Fig. 5 illustrates the way MajorCAN$_5$ nodes achieve consistency in front of up to five errors. Nodes belonging to X detect a dominant bit in the 3rd bit of EOF and transmit a 6-bit error flag. Nodes belonging to Y detect the error flag in the 4th bit and signal it with their own 6-bit error flags. All of them will perform the sampling to decide whether accept or not the frame. In the meantime the transmitter suffers two additional errors that delay the error flag detection until the 6th bit of the EOF. Then it accepts the frame and notifies the acceptance by transmitting an extended error flag. All the rest of the nodes perform the sampling and, even in the presence of the remaining two errors, decide to accept the frame.

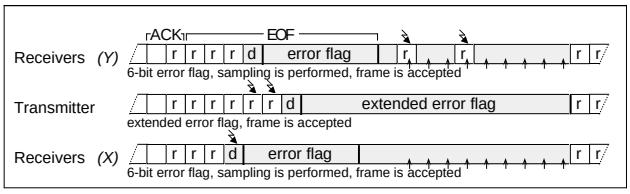

Figure 5. MajorCAN$_5$ consistency example.

6. Conclusions and future work

We have identified new scenarios where the CAN protocol violates the Agreement property of the Atomic Broadcast definition. Previous solutions for the until now known inconsistencies were higher level protocols [10]. Such an approach wastes communication bandwidth, a scarce resource in CAN, requires extra program memory space and implies a computational overhead for the nodes. In many control systems, especially in automotive applications, there are systems which have to be constructed with minimum memory and CPU power requirements. Moreover, we have shown that in the new scenarios none of these higher level protocols fulfill the Atomic Broadcast requirements. We take the different approach of modifying the CAN protocol and we introduce a new protocol called MajorCAN$_m$, that provides Atomic Broadcast in the presence of up to m errors in the communication channel, when the nodes present fail-silent behaviour. We show that the maximum communication overhead introduced by MajorCAN$_m$ compared with standard CAN is $4m - 9$ bits, that for our proposal of $m = 5$ means 11 bits. This overhead is negligible compared with any of the higher level protocols proposed in [10] that require the transmission of more than a CAN frame per message.

We are implementing the MajorCAN protocol in VHDL to develop a prototype for a MajorCAN controller. This will permit to obtain experimental results on the functionality of the protocol. We plan to do model checking on the VHDL description to achieve a formal verification. We are also planning to use a more accurate model to estimate the probability of the new scenarios introduced.

References

[1] J. Charzinski. Performance of the error detection mechanisms in CAN. In *1st International CAN Conference. ICC'94. Mainz, Germany*, September 1994.

[2] P. Ferriol, F. Navio, J. J. Navio, J. Pons, J. Proenza, and J. Miro-Julia. A double CAN architecture for fault-tolerant control systems. In *5th International CAN Conference. ICC'98. San Jose CA*, November 1998.

[3] 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, 2nd edition, 1993.

[4] H. Hilmer, H.-D. Kochs, and E. Dittmar. A fault-tolerant communication architecture for real-time control systems. In *Proc. IEEE Int. Workshop on Factory Communication Systems. Barcelona (Spain)*, October 1997.

[5] ISO. *International Standard 11898 - Road vehicles - Interchange of digital information - Controller Area Network (CAN) for high-speed communication*. 1993.

[6] H. Kopetz. Automotive electronics - present state an future propects. In *Digest of Papers of the IEEE 25th Int. Symp. Fault-Tolerant Computing FTCS'95 - Special Issue*, pages 66–75, Pasadena, California-USA, June 1995.

[7] Philips. *CAN bus specification 2.0. Parts A and B.*

[8] D. Powell. Failure mode assumptions and assumption coverage. In *Digest of Papers of the IEEE 22th Int. Symp. Fault-Tolerant Computing FTCS'92*, pages 386–395, Boston, Massachusetts-USA, July 1992.

[9] J. Proenza and J. Miro-Julia. Fixing the Atomic Broadcast impairments in the Controller Area Network (CAN) field-bus: the MajorCAN protocol. Technical Report 1-99, Universitat de les Illes Balears, Departament de Ciéncies Matemátiques i Informática, April 1999.

[10] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcast in CAN. In *Proc. IEEE 28th Int. Symp. Fault-Tolerant Computing. FTCS'98. Munich (Germany)*, June 1998.

[11] H. Zeltwanger. Failure detection and error handling in CAN-based networks. In *Seminario Anual de Automática, Electrónica Industrial e Instrumentación. SAAEI'98. Pamplona (Spain)*, September 1998.