Developing TOBE-CAN: Total Order Broadcast Enforcement in CAN

Manuel Barranco, Julián Proenza
Dpt. Matemàtiques i Informàtica, Universitat de les Illes Balears, Spain
{manuel.barranco, julian.proenza}@uib.es

Abstract

One of the drawbacks of the Controller Area Network (CAN) that must be overcome to make it suitable for critical applications is its incapacity for providing a Total Order Broadcast (TOB) communication service. A number of mechanisms were proposed to solve this problem, but each one of them only addresses a specific TOB limitation. Thus, this paper introduces TOBE-CAN, the first solution that comprehensively overcomes all the TOB flaws these previous mechanisms deal with. TOBE-CAN takes advantage of some of these mechanisms and further provides TOB while tolerating faults that are beyond their capabilities.

1. Introduction

The Controller Area Network (CAN) is a mature, robust and low-cost fieldbus with a tremendous success in a wide variety of distributed embedded control systems. Due to this, there is a significant amount of on-going work towards overcoming some dependability limitations of CAN, in order to make it suitable for critical applications [4]. One of such limitations is its incapability of providing *data consistency*, which is a fundamental service to achieve a consistent state in any distributed control system. It is specially relevant in systems that must attain a high degree of reliability, as they typically need to consistently manage redundancy. This service includes both *Reliable Broadcast* (RB) and *Total Order* (TO) [2]. Basically, RB states that every message transmitted by a correct node will be eventually delivered to all correct nodes; whereas TO means that a correct node N1 receives message m1 before m2 if and only if each one of all the other correct nodes also receives m1 before m2. *Data consistency* is also referred to as *Atomic Broadcast* (AB) [8]. But to stress the TO property, we prefer to use the term *Total Order Broadcast* (TOB).

When CAN was released, it was supposed to provide TOB by means of its error signaling and globalization mechanisms [1]. However, as discussed over the years, those mechanisms cannot enforce TOB under certain fault scenarios. First, when a CAN controller gets into the *error passive state* [1], it can no longer globalize the local errors it detects during a frame. If so, TOB cannot be ensured because a receiver can reject the frame and the others accept it for not being aware of the problem. In particular, if the sender does not retransmit the frame, then an *Inconsistent Message Omission* (IMO) occurs, i.e. only a subset of the nodes definitively receive the frame. A second

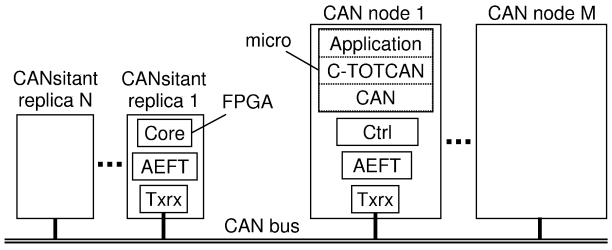

Figure 1. TOBE-CAN architecture

type of scenarios which can provoke IMOs were identified in [8, 5]. They are called the *last-bit scenarios* and happen when errors affecting the last bits of the frame lead a subset of nodes to not reject it. Additionally, newer scenarios were identified in [7], in which errors in the intermediate bits of the frame also lead to the same kind of inconsistencies.

Several mechanisms have been proposed for addressing the impairments to TOB in CAN [8, 3, 5, 6, 7], but none of them covers all the fault scenarios identified above. Thus, the objective of this work is to develop a solution for CAN that comprehensively treats all of them, taking advantage of some of the features of the previously proposed mechanisms. We refer this solution to as TOBE-CAN, i.e. a protocol that provides Total Order Broadcast Enforcement in CAN. Moreover, TOBE-CAN does not only integrate and adapt previous mechanisms, but further tolerates faults that would lead those mechanisms to fail in achieving TOB.

2. Groundwork of TOBE-CAN

To achieve TOB in TOBE-CAN, we propose to combine, modify and extend some of the previously proposed mechanisms cited above. The first one is very simple and consists in each node simply disconnecting its CAN controller when the controller's error counters reach a threshold prior to its change to the error passive state.

The second mechanism we adapted is the one CANsistant [5] provides for detecting all the last-bit scenarios. In its original form, CANsistant is an specifically designed hardware module that monitors the channel to indicate when an IMO due to these scenarios can happen. Note that any other solution for detecting IMOs at the frame's end, e.g. [3], does not cover all the last-bit scenarios.

The third mechanism consists in a circuit called *Aggregated Error Flag Transmitter* (AEFT) [7], which can be inserted between each node's CAN controller and its transceiver. The AEFT prolongs the duration of the CAN error signaling in order to enforce that any error detected in the intermediate bits of the frame is globalized and, then,

 doi:10.1109/ETFA.2012.6489700

that the affected frame is consistently rejected by all. No other solution has been proposed so far to address inconsistencies caused by errors in these bits.

Finally, to also provide TO, we adapt some of the mechanisms of a protocol called TOTCAN [8], which executes between the application and the standard CAN layer.

3. TOBE-CAN design

3.1 System architecture

Figure 1 sketches the architecture of TOBE-CAN. It consists of several CAN nodes and N CANsistant replicas. We will use the term *replica* to refer to a CANsistant replica, whereas we call *node* to both a CAN node and a CANsistant replica. Each CAN node can be implemented using a *Commercial-Off-The-Shelf* (COTS) micro, CAN controller (*ctrl*) and transceiver (*txrx*). A COTS txrx can be also used for the interface of each replica to the channel. But the replica's core must be implemented as a custom-made circuit, e.g. as a *Field-Programmable Gate Array* (FPGA), that carries out the functionalities of CANsistant. As it will be described, we extended the original features of CANsistant in TOBE-CAN; so that apart from detecting IMOs, each replica also transmits and receives frames. Thus, each replica must be provided with buffers to store frames, as well as with the capacity of signaling and globalizing channel errors it detects locally. Hence, an AEFT unit is attached not only to each CAN node, but also to each replica to ensure that they can globalize errors.

Finally, the software of each CAN node is divided into the application itself, a new protocol we call *core-TOTCAN* (C-TOTCAN), and a regular CAN driver. C-TOTCAN is based on TOTCAN [8], from which it basically keeps the strategy each recipient uses for ordering the frames. Basically, in TOTCAN, each recipient inserts any received frame into a FIFO queue and marks it as *UNSTABLE*, meaning that it has not confirmation of that frame being consistently received. The sender retransmits a copy of the frame as long as it detects a possible inconsistency. Every time a recipient receives the copy it shifts the original frame to the tail of its own queue; so that when the sender does not retransmit it anymore, the final position of the frame is the same in the queue of every recipient. Then, the sender compels all recipients to eagerly retransmit an ACCEPT message associated to that frame, so that all of them eventually receive this ACCEPT. Each recipient uses the ACCEPT to mark the original frame as *STABLE* and delivers it to the application when it reaches the head of the queue. As it will be discussed, the way in which TOBE-CAN modifies this strategy makes the system more robust and efficient.

3.2 Fault model

TOBE-CAN includes mechanisms to tolerate faults that can prevent it from enforcing TOB. Faults affecting other communication aspects, e.g. a babbling-idiot node blocking the channel, must be treated with other mechanisms.

Specifically, TOBE-CAN deals with faults that manifest as follows. First, a faulty CAN node can send any kind of information, as long as it does not impersonate a replica by sending frames that are reserved for the TOBE-CAN protocol itself. This is reasonable since, conversely to TOTCAN, CAN nodes do not send any frame related to the protocol itself. Second, a fault affecting the internal circuitry of a replica leads it to exhibit a crash failure, so that it does not transmit any further message. This can be enforced by means of internal duplication and comparison. In contrast, the fault model includes situations in which external transient disturbances, or temporary faults affecting the replica's interface, lead the replica to encounter a number of channel errors greater than the maximum number of errors CANsistant can deal with [6]. Thus, a replica can temporarily fail by not detecting a potential IMO. Finally, a faulty AEFT leads its corresponding node to stop communicating, thereby exhibiting a crash failure too. This can be achieved by providing the AEFT with self-checking circuitry that disconnects its node upon fault detection.

3.3 Functional description

Similarly to TOTCAN, TOBE-CAN broadcast a given frame in two phases. In the first one, the frame is retransmitted until it is consistently received by all the recipients. Then, an ACCEPT message associated to that frame is transmitted, as many times as needed to be consistently received too, to indicate to both the recipients and the sender that the frame has been successfully broadcast.

3.3.1 Sender behavior

When the C-TOTCAN layer of the sender receives a frame transmission request, it instructs the transmission of that frame through its CAN controller. As in TOTCAN, the CAN controller automatically retransmits the frame as long as it encounters an error. In TOTCAN the sender assumed that the frame is consistently exchanged when the controller notifies of its successful transmission. However, the inconsistency scenarios identified in [5] make it impossible to rely on the sender's controller for taking this decision. Thus, when the controller notifies C-TOTCAN of a successful transmission, the sender relies on the CANsistant replicas for further retransmitting the frame as long as they consider that this frame suffers from a potential IMO. The sender only considers the frame to be successfully transmitted (and then notifies it to the application) when it receives an ACCEPT frame from the CANsistant replicas, confirming that the frame has been consistently broadcast.

3.3.2 Replica behavior

As concerns each replica, when it detects that a new frame is correctly transmitted until the bit just before the last-but-one bit of the *End Of Frame* (EOF) field, the replica stores that frame in its *Frame Retransmission Buffer* (FRB).

If the replica detects an error at any of the two last bits of the EOF, or that the frame suffers from a potential IMO, it marks it as not consistently received (*notConsRx*). Then, it tries to retransmit it until it is consistently received. Specifically, at the beginning of each bus idle period, each

replica tries to retransmit the *notConsRx* frame of its FRB that has the highest priority, i.e. the frame with the lowest CAN identifier (CAN-ID). However, to prevent collisions among the sender and different replicas willing to retransmit the same frame, a different delay is assigned to each one of them for accessing the channel. Consider N replicas where each one of them has an univocally numerical label $r \in [1, N]$, and let $D_{s,f}$ and $D_{r,f}$ be the delay with which the sender and the replica r try to transmit the frame f respectively. Initially, $D_{s,f} = 1$ and $D_{r,f} = r+1$. Then, a node nod, either the sender or a replica, with $D_{nod,f}$ is allowed to access the channel for transmitting f at the $D_{nod,f}$th bit of bus idle.

If the sender or a replica does not transmit the frame in the bit of bus idle when it is expected to do so, then the following replica in delay takes over. However, since each replica waits for its predecessors to access the channel, there may be situations in which a node (another sender or replica) accesses the channel for transmitting a frame with a priority lower than f. To mitigate this priority inversion problem, the replicas decrease their delays when needed. Specifically, if at the k-th bit of bus idle the replica r monitors a frame with a lower priority than the one it wants to retransmit, and $1 <= k < D_{r,f}$, then the replica can assume that every node, nod, either the original sender, or a replica whose $D_{nod,f} \leq k$, either crashed or did not detect the IMO. Thus, the replica decreases its $D_{r,f}$ by k. In this way, in the next bus idle, all replicas with $D_{r,f} > k$ will try to retransmit f without waiting for the sender nor for the previously preceding k replicas. Finally, due to synchronization differences at bit-level, the sender and different replicas may start transmitting simultaneously the same frame, f. If this collision provokes an error, each replica, r, sets the $D_{r,f}$ to its original value. This avoids a potential deadlock due to the sender and/or different replicas continuously colliding when trying to retransmit the same frame.

When a replica does not detect a potential IMO in a given frame, it marks it as consistently received (*consRx*) within the FRB and, then, it inserts an ACCEPT frame associated to that frame into a buffer we call the *Frame Acceptance Buffer* (FAB). From then on, the replica will continuously try to transmit that ACCEPT frame in the 1st bit of bus idle until it is also consistently received. The replica gives priority to the frames within the FAB over the ones located at the FRB. In particular, it manages its FAB as a FIFO queue, choosing the FAB's frame it firstly detected as consistently exchanged. This reduces the delay with which the recipients deliver to the application the frames consistently received so far but that have not been confirmed yet. Moreover, as a consequence, the size of the buffers at the recipients and at the replicas is also reduced, as they must have enough capacity to accommodate all frames for which the ACCEPT frame has not been broadcast yet.

The format of the ACCEPT frame is different to the one proposed in TOTCAN [8]. In TOTCAN an ACCEPT frame included a CAN identifier (CAN-ID) that carries both control information and a value that identifies the original sender. This restricted the amount of CAN-IDs that are available for the application. Moreover, it made it hard to control the priority with which different ACCEPT frames should be transmitted. In contrast, in TOBE-CAN, all ACCEPT frames have the same CAN-ID, so that it only consumes one of the possible CAN-IDs. Also, by assigning the lowest CAN-ID to the ACCEPT frame, it is possible to prioritize it over the rest of CAN frames, which further reduces the delay of message delivery at the recipients.

Another format difference is that in C-TOTCAN the data field of an ACCEPT frame includes a first byte called *arbByte*, two bytes named *idBytes*, four bytes called *hashBytes*, and a byte named *abortByte*. Since all ACCEPT frames have the same CAN-ID, different replicas trying to transmit an ACCEPT frame at the 1st bit of bus idle will win the arbitration. Thus, to prevent the ACCEPT frames from colliding, all replicas also apply the CAN bit-wise arbitration during the arbByte. For this, each replica transmits an univocal value during this byte and backs off if it detects a dominant bit when transmitting a recessive one.

The idBytes and hashBytes identify the CAN frame the ACCEPT refers to, by respectively specifying the CAN-ID of that frame and a hash value calculated from the payload of the original frame. Note that by including the hash value in the ACCEPT frame it is unnecessary to use sequence numbers to distinguish between editions of the same CAN frame that carry different payloads.

The last byte of the ACCEPT frame, i.e. the abortByte, is used to force all replicas to agree on whether or not an ACCEPT frame must be transmitted. In this sense, note that different replicas may have different views of the channel, depending on the local errors each one of them encounters. This may lead to situations in which only a subset of the replicas consider that a CAN frame has been consistently received and, then, that the corresponding ACCEPT frame must be transmitted. An agreement on this decision could be reached by forcing every replica to accept as valid any ACCEPT frame it observes in the channel, even if it did not schedule that frame. This means that replicas could consider as false alarms all IMOs detected only by a subset of replicas (false IMOs may happen in CANsistant [6] due to local errors). However, to consider as valid any ACCEPT frame would be incorrect if that frame is scheduled by a replica that suffered from a fault that leaded it to encounter a quantity of errors greater than the maximum number of errors in whose presence CANsistant is able to detect all possible IMOs. Thus, to further tolerate these faults, all replicas must discard any ACCEPT frame that is not consistently scheduled by all of them.

For this purpose, a replica that sends an ACCEPT frame transmits the value FFh during the abortByte, which will be basically encoded as a sequence of 8 recessive bits with an in-between dominant stuff bit. Then, the replicas that do not agree on transmitting that ACCEPT abort it by sending a predefined pattern of dominant bits during the abortByte and, when this byte ends, by globalizing an error. In this way, if replicas observe the abort pattern within the abort-

Byte, then they discard the ACCEPT frame from their FAB and mark the corresponding frame of the FRB as *notConsRx*. This means that the management of the broadcast of the original CAN frame is resumed to the phase in which it must be retransmitted until it is consistently received by all recipients. Moreover, by globalizing an error just after the abort pattern, it is ensured that no node receives an ACCEPT frame for which all replicas have not reached a consensus. Also note that, to not discard an ACCEPT from the FAB unnecessarily, the replicas that do not observe the abort pattern, when an error is signaled during an ACCEPT, can assume that a channel error occurred and simply try to retransmit the ACCEPT. This last feature does not threat the agreement if errors during the abortByte lead a subset of replicas to not observe the abort pattern and, thus, to not discard the ACCEPT frame from their FAB. This is because the replicas that did detect the pattern will eventually abort the ACCEPT frames of those that did not.

Once this consensus has been reached, replicas still execute the following algorithm to agree on whether or not the ACCEPT frame has been consistently received by all recipients. Each replica tries to retransmit the ACCEPT in the 1st bit of bus idle as long as this frame suffers from a channel error or from an IMO (as already said). Conversely, when the replica correctly receives the ACCEPT and it does not detect an IMO, it does not transmit anything but checks if another replica starts retransmitting the ACCEPT. Thus, if no frame starts being broadcast at the 1st bit of bus idle, or if the frame that starts is not an ACCEPT frame, the replica knows that no other replica is trying to transmit. This means that they all agreed that the ACCEPT frame did not suffer from an IMO and, thus, that it was consistently received. Otherwise, if the replica detects that the ACCEPT is being retransmitted, it knows that at least one replica did not agree on the previous ACCEPT. Thus, it restarts the agreement algorithm considering the current ACCEPT, i.e. it checks if the new ACCEPT suffers from an IMO and, then, proceeds as just explained.

3.3.3 Recipient behavior

As concerns each recipient, as said before, its C-TOTCAN layer keeps the TOTCAN strategy for ordering the frames in its reception queue. The main difference with respect to TOTCAN is that in C-TOTCAN the recipients do not eagerly retransmit every ACCEPT frame to ensure that all recipients eventually receive it. The CANsistant replicas are the responsible for enforcing the reliable and consistent reception of every ACCEPT frame, which is more efficient, as an ACCEPT frame is retransmitted only when it suffers from a potential IMO. Moreover, the replicas simultaneously reach a consensus on whether an ACCEPT frame has been consistently received. This feature can be used by higher layer protocols for taking specific decisions, e.g. in a time-triggered protocol this could be used to determine if a frame is consistently received in a given cycle and, then, to schedule the appropriate frames in the next one.

Finally, note that in TOTCAN the recipient associates a given time out to each frame of the queue to detect when the sender crashes and, thus, to purge the frame to prevent it from blocking the head of the queue. This is not necessary in TOBE-CAN, as replicas take over when necessary to retransmit a given frame. Anyway, to provide graceful degradation, a single time out could be used to stop using TOBE-CAN when all replicas crash.

4. Conclusions and future work

We presented the initial design of TOBE-CAN, a solution intended to comprehensively overcome the limitations that make it impossible for CAN to enforce Total Order Broadcast (TOB). TOBE-CAN integrates and adapts a set of mechanisms previously proposed to face faults scenarios reported in the literature as potential impairments to TOB in CAN. Moreover, by extending these mechanisms it provides an holistic solution that further tolerates faults that, otherwise, would lead previous mechanisms to fail. We plan to formally verify and implement TOBE-CAN, as well as to enhance its capabilities to address potentially new scenarios such as one recently discovered, in which discrepancies in the DLC [1] value are the cause of inconsistency.

5. Acknowledgement

This work was supported by the Spanish Science and Innovation Ministry with grant DPI2008-02195, by the Spanish Economy and Competitiveness Ministry with grant DPI2011-22992, and by FEDER funding. We would also like to thank Fuyu Yang for the comments about a new potential inconsistency scenario in CAN.

References

[1] ISO11898-1. Controller Area Network (CAN) - part 1: Data link layer and physical signalling., 2003.

[2] V. Hadzilacos and S. Toueg. *Fault-tolerant broadcasts and related problems*. S. J. Mullender, editor, ACM-Press, 1993.

[3] M. Livani. SHARE: A transparent approach to fault-tolerant broadcast in CAN. In *Proc.of the 6th Int. CAN Conf.*, 1999.

[4] J. Pimentel, J. Proenza, L. Almeida, G. Rodríguez-Navas, M. Barranco, and J. Ferreira. *Dependable Automotive CAN Networks*. Handbook on Automotive Embedded Systems. CRC Press. Edited by N. Navet and F. Simonot-Lion, 2009.

[5] 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 Communication and Computations, Taipei, Taiwan*, 2000.

[6] 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 *IEEE Conf. on Emerging Technologies & Factory Automation*, 2009.

[7] G. Rodriguez-Navas, C. Winter, and J. Proenza. Injection of aggregated error flags as a means to guarantee consistent error detection in CAN. In *IEEE Conf. on Emerging Technologies & Factory Automation*, 2011.

[8] J. Rufino, P. Veríssimo, and A. Guillerme. Fault-Tolerant Broadcasts in CAN. In *28th Annual Int. Symposium on Fault-Tolerant Computing*, 1998.