Reliable Real-Time Communication in CAN Networks

Luís Miguel Pinho, *Member*, *IEEE*, and Francisco Vasques, *Member*, *IEEE*

Abstract—Controller Area Network (CAN) is a fieldbus network suitable for small-scale Distributed Computer Controlled Systems (DCCS), being appropriate for sending and receiving short real-time messages at speeds up to 1 Mbit/sec. Several studies are available on how to guarantee the real-time requirements of CAN messages, providing preruntime schedulability conditions to guarantee the real-time communication requirements of DCCS traffic. Usually, it is considered that CAN guarantees atomic multicast properties by means of its extensive error detection/signaling mechanisms. However, there are some error situations where messages can be delivered in duplicate or delivered only by a subset of the receivers, leading to inconsistencies in the supported applications. In order to prevent such inconsistencies, a middleware for reliable communication in CAN is proposed, taking advantage of CAN synchronous properties to minimize the runtime overhead. Such middleware comprises a set of atomic multicast and consolidation protocols, upon which the reliable communication properties are guaranteed. The related timing analysis demonstrates that, in spite of the extra stack of protocols, the real-time properties of CAN are preserved since the predictability of message transfer is guaranteed.

Index Terms—Communication protocols, controller area network, fault-tolerant systems, real-time systems.

1 Introduction

The Controller Area Network (CAN) protocol [1] was originally developed to be used within road vehicles to interconnect microprocessor-based components. More recently, the CAN protocol is also being considered for the automated manufacturing and distributed process control environments [2] and is already used as the communication interface in proprietary architectures such as DeviceNet [3].

The CAN protocol implements a priority-based bus, with a carrier sense multiple access with collision avoidance (CSMA/CA) MAC, being appropriate for sending and receiving short real-time messages at speeds up to 1 Mbit/sec. Several studies on how to guarantee the real-time requirements of messages in CAN networks are available (e.g., [2], [4], [5], [6], [7], [8]), providing the necessary preruntime schedulability conditions for its timing analysis.

The CAN protocol has extensive error detection/signaling mechanisms, imposing automatic message retransmission in case of a detected error. Nevertheless, it is known that CAN error recovery mechanisms may fail when an error is detected in the last but one bit of the frame [9]. This problem may cause messages to be delivered in duplicate to the application by some of the nodes (*inconsistent message duplicate*). Similarly, if the sender fails before retransmitting the message, it may cause the message to be delivered only by a subset of the nodes (*inconsistent message omission*).

This misbehavior may be disastrous if the CAN network is used to support replicated applications, as these applications require replicated components providing the same results when they are correct. The consistency of the delivered messages must be guaranteed by atomic multicast protocols, which guarantee both that messages are delivered by all (or none) of the component replicas and that messages are delivered only once. Furthermore, there is the need to agree in the delivery order of multicasts and to consolidate values from replicated inputs. Hence, it is necessary to provide a middleware guaranteeing these properties in spite of the underlying CAN inconsistencies. Moreover, it is also necessary to guarantee the real-time behavior of the overall approach (allowing the offline analysis of the messages' response time).

This paper proposes a middleware layer intended to guarantee reliable real-time communication in CAN networks in spite of the underlying inconsistency in message deliveries. The paper is structured as follows: After a brief description of the CAN protocol, Section 3 presents the requirements for reliable real-time communication in CAN networks. The proposed middleware layer is then described in Section 4, with a special focus on the proposed set of atomic multicast and consolidation protocols. Afterward, in Section 5, the related set of preruntime schedulability conditions is presented, enabling the timing analysis of the supported reliable real-time communication. For a better understanding of the proposed protocols, a numerical example is presented in Section 6. Finally, Section 7 presents a comparison with other relevant approaches and some conclusions are outlined in Section 8.

2 CAN Protocol

The CAN protocol implements a priority-based bus, with a carrier sense multiple access with collision avoidance (CSMA/CA) MAC, where bus signals can take two different states: *recessive bits* (idle bus) and *dominant bits* (which always overwrite recessive bits). The collision resolution mechanism works as follows: When the bus

- *L.M. Pinho is with the Department of Computer Engineering, ISEP, Polytechnic Institute of Porto, Rua Dr. Ant. Bernardino de Almeida, 431, 4200-072 Porto, Portugal. E-mail: lmp@isep.ipp.pt.*
- *F. Vasques is with the Department of Mechanical Engineering, FEUP, University of Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal. E-mail: vasques@fe.up.pt.*

Manuscript received 3 Apr. 2001; revised 15 Feb. 2002; accepted 3 Jan. 2003. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 113923.

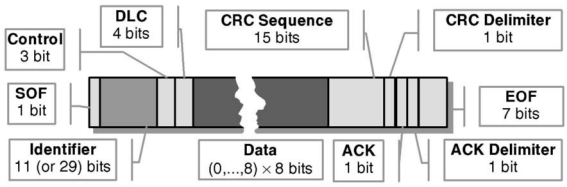


Fig. 1. Structure of a Data Frame.

becomes idle, every node with pending messages will start to transmit. If a node transmitting a recessive bit reads a dominant one, it means that there was a collision with a higher-priority message and, consequently, the transmission is aborted. The highest-priority message (the one with most leading dominant bits) being transmitted will proceed without perceiving any collision and, thus, will be successfully transmitted. Nodes that lose the arbitration phase will automatically retry the transmission of requested messages.

There are four types of frames that can be transferred in a CAN network. Two are used during the normal operation of the CAN network: the Data Frame, which is used to send local data, and the Remote Frame, which is used to request remote data. Besides these two frames, there are also the Error Frame, which signals the detection of error states in the CAN network, and the Overload Frame, which is used by nodes requiring extra delays before the transmission of Data Frames.

Fig. 1 shows the structure of a Data Frame (specific fields: SOF, Identifier, Control, DLC, CRC, ACK, and EOF are described in [1]). A Remote Frame has the same identifier and structure (without data field) as the remotely requested Data Frame. The structure of both the Error and Overload Frames will be presented in Section 2.1.

At the physical layer, frames are transmitted using the NRZ (Non-Returning to Zero) coding technique, with the insertion of stuff bits. That is, whenever there are more than five equal consecutive bits (up to the end of the CRC Field), there is the insertion of an opposite bit in the frame. This opposite bit will be detected and removed by the physical layer at the receiving side. This bit stuffing technique ensures that, in the normal behavior, there will never be more than five consecutive equal bits on the bus.

2.1 Error Detection and Recovery Mechanisms

In the CAN protocol, every node continuously monitors the bus to detect any transmission error. A node detecting an error will transmit an Error Frame, violating the bit-stuffing rule. As a consequence, all receiving nodes know that the frame being transmitted has an error. An Error Frame has the following structure:

- Six to twelve consecutive dominant bits (Error Flag). The node that first detects the error transmits the 6-bit Error Flag. If any other node only recognizes the bit stuffing error induced by the Error Flag, it will transmit a new Error Frame, thus the Error Flag will be up to 12 bits long;
- Eight consecutive recessive bits (Error Delimiter) signaling the end of the Error Frame.

Nodes that are still not ready to receive another frame may also transmit one or two consecutive Overload Frames (with the same structure of the Error Frame) in order to slow down network transmission.

Sending Error Frames is a very interesting mechanism to ensure that every node acquires the same global state of the network (state coherence). However, a node failure may induce the transmission of consecutive Error Frames, blocking all the ongoing communications. To solve this problem, CAN controllers have two error counters (for, respectively, transmitting and receiving errors) to isolate erratic nodes. The value of these counters, which determine the operating state of the node, is increased or decreased (at different rates) as a function of the detected errors. Thus, these counters act as self-surveillance mechanisms, disconnecting faulty nodes (fault-confinement techniques) when their values reach a specified threshold. In the case of multiple errors, the node goes first into the *Error-Passive* state (where it does not interfere with the transmissions of other nodes) and then into the *Bus-Off* state (where the node is disconnected from the network) [1].

2.2 Response Time Analysis of CAN Networks

In order to guarantee the real-time requirements of messages transferred by CAN networks, it is necessary to evaluate the worst-case response time of messages. In [4], the authors address in detail the response time analysis of CAN networks. Existing schedulability analysis is adapted to the case of scheduling messages in a CAN network. The analysis assumes a network with n message streams defined as:

$$S_m = (C_m, T_m, D_m), \tag{1}$$

where S_m defines a message stream characterized by a unique identifier. A message stream is a temporal sequence of messages concerning, for instance, the remote reading of a specific process variable. C_m is the longest message duration of stream S_m and T_m is the periodicity of its requests. In order to have the response time analysis independent of the upper layers model, it is assumed that this periodicity is the minimum time interval between the arrival of two consecutive requests to the outgoing queue. Finally, D_m is the relative deadline of a message, that is, the maximum time interval between the instant when the message is placed in the outgoing queue and the instant when the message must be completely transmitted.

The analysis assumes fixed priorities for message streams (as the medium access is based on fixed identifiers), relative deadlines not greater than periods, and a non-preemptive scheduling model (as lower priority messages being transmitted cannot be preempted by requested higher priority messages).

Therefore, the worst-case response time of a queued message, measured from the arrival of the message request to its complete transmission, is:

$$R_m = I_m + C_m. \tag{2}$$

The schedulability of the message stream set is guaranteed if every message has a response time smaller than its deadline. The term I_m represents the worst-case queuing delay (longest time interval between the arrival of the message request and the start of its transmission):

$$I_m = B_m + \sum_{\forall_{j \in hp(m)}} \left(\left\lceil \frac{I_m + \tau_{bit}}{T_j} \right\rceil \times C_j \right) \tag{3}$$

B_m is the worst-case blocking, which is the duration of the longest lower priority message:

$$B_m = \max_{\forall k \in lp(m)} \{0, C_k\}. \quad (4)$$

$lp(m)$ and $hp(m)$ are, respectively, the set of message streams with lower-priority and higher-priority than S_m. τ_{bit} is the duration of a bit transmission, taking into account the difference in the arbitration start time at different nodes (due to propagation delays).

Equation (3) embodies a mutual dependency since I_m appears in both sides of the equation. The easiest way to solve such an equation is to form a recurrence relation [10].

The computation of the network load is a single measurement based on the characteristics of the message streams. Such a network load can be evaluated as follows:

$$U = \sum_{m=1}^{n} \frac{C_m}{T_m}. \quad (5)$$

2.3 Inaccessibility Analysis of CAN Networks

The use of CAN networks to support dependable real-time applications requires not only time-bounded transmission services, but also a minimum level of confidence on the continuity of service. Such continuity of service is not fully guaranteed in CAN networks since it may be disturbed by temporary periods of network inaccessibility (periods during which nodes cannot communicate with each other, due to the existence of on-going error recovery mechanisms).

Considering the existent error recovery mechanisms, it follows that the longest network inaccessibility [11] results from a Form Error (incorrect structure of the frame) detected at the end of the EOF delimiter. Such network inaccessibility is:

$$t_{ina} = C_{MAX} + C_{error} + C_{IFS}, \quad (6)$$

where C_{error} and C_{IFS} are the duration of an Error Frame and the Inter-Frame Spacing (two consecutive frames must be separated by at least 3 recessive bits), respectively, and C_{MAX} is the longest duration of a CAN message.

In the presence of multiple bus errors, two different scenarios can be considered [11]:

- A burst of successive bit errors, where only the first one corresponds to a bit corruption in the Data Frame. The others will just disturb Error Frames being transmitted in response to the first error.
- A longer network inaccessibility can be induced by errors that are sufficiently apart to interfere with n consecutive Data Frames being transmitted in the bus, resulting in n failed attempts to transmit a Data Frame (also producing n Error Frames).

The network inaccessibility resulting from this second scenario is:

$$t_{n\_ina} = n \times (C_{MAX} + C_{error} + C_{IFS}). \quad (7)$$

In order to integrate this inaccessibility analysis in the response time analysis of CAN networks, the maximum inaccessibility time $Ina(I_m)$ must be added to (3) [4], [7]:

$$I_m = B_m + \sum_{\forall j \in hp(m)} \left(\left\lceil \frac{I_m + \tau_{bit}}{T_j} \right\rceil \times C_j \right) + Ina(I_m). \quad (8)$$

The maximum inaccessibility time interfering with the transmission of a message of stream S_m can be evaluated considering that the maximum number of errors (n_{errors}) which can interfere with the transmission of message m (considering n errors in a period T) is:

$$n_{errors} = n \times \left\lceil \frac{I_m + C_m}{T} \right\rceil. \quad (9)$$

Hence, the inaccessibility time due to bus errors is:

$$Ina(I_m) = n \times \left\lceil \frac{I_m + C_m}{T} \right\rceil \times t_{ina}. \quad (10)$$

The network load considering periods of temporary network inaccessibility is:

$$U_{ina} = \frac{n \times t_{ina}}{T}. \quad (11)$$

Consequently, the overall network load is:

$$U = \left(\sum_{\forall m} \frac{C_m}{T_m} \right) + U_{ina}. \quad (12)$$

2.4 Inconsistencies in the Transfer of Messages

In spite of the extensive error detection and recovery mechanisms in CAN, there are some known reliability problems [9] that can lead to an inconsistent state of the supported applications. This misbehavior is a consequence of different error detection mechanisms at the transmitter and receiver sides. A message is valid for the transmitter if there is no error until the end of the transmitted frame. If the message is corrupted, a retransmission is triggered according to its priority. For the receiver, a message is valid if there is no error until the last but one bit of the received frame, the value of the last bit being treated as "do not care." Thus, a dominant value in the last bit does not lead to an error state, in spite of violating the CAN rule stating that the last 7 bits of a frame are all recessive.

In Fig. 2, the Sender node transmits a frame to Receivers A and B. Receiver B detects a bit error in the last but one bit of the frame. Therefore, it rejects the frame and sends an Error Frame (requesting the frame retransmission) starting in the following bit (last bit of the frame). As for a receiver, the last bit of a frame is a "do not care" bit, Receiver A will not detect this error, and will accept the frame. However, as the transmitter schedules the frame for retransmission, Receiver A will have an *inconsistent message duplicate*. The use of sequence numbers in messages can easily solve this problem, but it does not prevent messages from being received in different orders, thus not guaranteeing total order. On the other hand, if the Sender fails before being able to successfully retransmit the frame, then Receiver B will never receive the frame, although Receiver A has delivered it. This causes an *inconsistent message omission*, which is a problem with a more difficult solution.

In [9], the probability of message omissions and/or duplicates is evaluated, in a reference period of one hour, for a 32-node CAN network, with a network load of approximately 90 percent. Bit error rates were used ranging

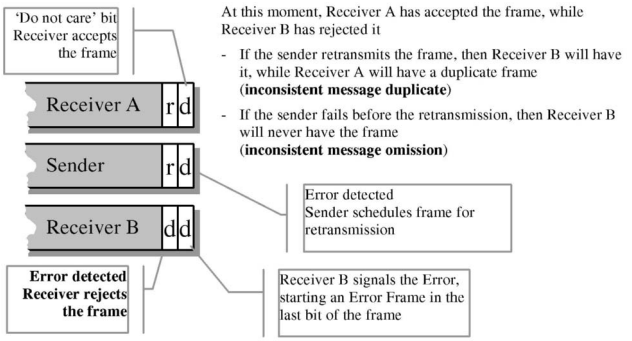


Fig. 2. Inconsistency in CAN.

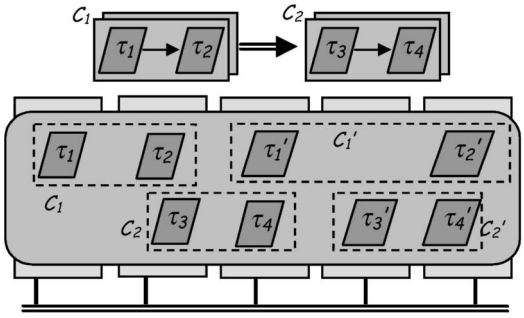


Fig. 3. Replicated hard real-time application.

from 10^{-4} to 10^{-6} and node failures per hour of 10^{-3} and 10^{-4}. For inconsistent message duplicates, the results obtained ranged from 2.87×10^{1} to 2.84×10^{3} duplicates per hour, while, for inconsistent message omissions, the results ranged from 3.98×10^{-9} to 2.94×10^{-6} omissions per hour.

These values demonstrate that, for reliable real-time communications, CAN built-in error recovery mechanisms are not sufficient. The use of CAN networks to support DCCS applications requires not only a time-bounded transmission service, but also the guarantee of delivery consistency for the supported applications.

Particularly when a CAN network is used as the communication infrastructure for a fault-tolerant distributed system, it must support the communication mechanisms required by the replicated applications. In such applications, replicas must behave as single fault-free components, that is, they must provide the same results when they are correct. Thus, all replicas must work with the same input values in the same order, meaning that the consistency of message deliveries must be guaranteed.

3 Communication Requirements

3.1 System Model

A distributed hard real-time application constituted by several tasks (processing units) is considered. As the target is reliability through replication, the notion of "component" is introduced as the replication unit. Applications are divided into components, each one including tasks and resources from several nodes or located in just one node.

Fig. 3 illustrates a real-time application with four tasks (τ_1, τ_2, τ_3, and τ_4). The application is divided into two different components (C_1 and C_2), which are replicated (its replicas being C_1' and C_2'). Since replication is at the component level, it is not necessary to agree on the output of component internal tasks (for instance, the output of τ_1 to τ_2 does not need to be agreed upon with the output of τ_1' to τ_2'). Results must only be consolidated when they are made available to other components.

As this model considers the existence of active replication, there is the need to guarantee that replicas execute deterministically, that is, replicated tasks execute with the same data and timing-related decisions are the same in each replica [12]. In order to enforce the deterministic execution of the replicated components, it must be guaranteed that all messages sent by correct components are delivered to all its recipients. For the case of a message sent by an incorrect component, there must be an all-or-none guarantee: Either all correct components deliver that message or none of them deliver it. Furthermore, there is the need to agree upon the order by which messages are delivered to its recipients and to consolidate messages from replicated components' outputs.

Concerning timing-related determinism, it is necessary to guarantee that timing-related decisions are the same in each replica. The use of the timed messages [13] concept allows a restricted model of multitasking to be used and eliminates the need for agreement between the internal tasks of each component. With timed messages, agreement is only needed to guarantee that all replicated components work with the same input values and that they all vote on the final output. The use of timed messages implies the use of appropriate clock synchronization protocols (such as the one proposed in [14]) since clock deviations must be bounded.

The presented model is the underlying model of the Hard Real-Time Subsystem of the DEAR-COTS architecture [15]. From this model, it is clear that four types of message exchanges must be supported: *1-to-1*, *1-to-many*, *many-to-1*, and *many-to-many*.

For *1-to-1 communication* (communication from a nonreplicated component to another nonreplicated component or communication internal to a component), there is only the need for a reliable multicast service since, as there is no replication, order issues are not relevant. However, when a result is to be disseminated to a group of replicated components (*1-to-many communication*), atomic multicast protocols [16] must be used to guarantee that replicated receivers get the same information in the same order.

When a group of replicated components receives a message from another group of replicated components (*many-to-many communication*), it must agree on the value to use. Thus, an interactive consistency protocol must be used. If an underlying atomic multicast mechanism is used to disseminate each value, then it is guaranteed that every receiver will have the same input values and by the same order. The agreement decision can then be performed by a simple *Consolidate* protocol, which decides on one of the received values.

The case of communication from a group of replicated components to a single *component* (*many-to-1 communication*) is a simplified version of the previous one. The receiving component only has to decide from the set of received inputs. The same *Consolidate* protocol can be used combined with a reliable data transfer from the replicated transmitters to the receiver (as there are no order requirements).

3.2 Reliable Communication Properties

The reliability of the supported hard real-time applications is highly dependent on the reliability of the underlying communication mechanisms: atomic multicast and consolidation protocols. Based on [16] and defining a correct node as a node that does not fail while a multicast is in progress, an atomic multicast has the following properties:

- Validity: If a correct node multicasts a message m, then all correct nodes deliver m.
- Agreement: If a correct node delivers a message m, then all correct nodes deliver m.
- Integrity: For any message m, every correct node delivers m at most once and only if m was previously broadcast by $sender(m)$;
- Total Order: If correct node p and q both deliver message m and m', then p delivers m before m' if and only if q delivers m before m'.

Note that these properties are related to the message delivery and not to its reception. Nodes may receive incorrect messages (concerning these properties), but the proposed protocols will guarantee a correct delivery of messages to the supported applications.

CAN error detection and recovery mechanisms ensure the Validity property since, when the sender is correct, all nodes will receive (and deliver) the message. Note that the network can be referred as a fail-consistent bus [17] since there is no possibility for different nodes to receive the same message with different values. CAN error detection and recovery mechanisms are not, however, sufficient to guarantee the Agreement and Integrity properties [9]. In fact, it is possible for a correct node to receive a message not received by some other correct node (inconsistent message omission) and it is also possible that some node receives the same message more than once (inconsistent message duplicate). Total Order is also not guaranteed since new messages can be interleaved with retransmissions of failed messages, inducing nodes to receive messages in different orders.

The consolidation needs just to guarantee that the *decide* function is correctly applied to the full set of proposed values (as an underlying atomic multicast protocol is already used to disseminate the proposed values). Therefore, the *Consolidate* protocol requires the following properties (based on the consensus agreement properties [16]):

- Validity: If all components proposing a value, propose v, then all correct components will decide v;
- Agreement: If a correct component decides v, then all correct components decide v.

The Integrity property [16] is not considered as it precludes decisions on values different from those proposed, like average or median functions. For the case where a single component receives replicated inputs (*many-to-1 communication*), these properties can be reduced to Validity as the Agreement property is only relevant for replicated receivers.

3.3 Failure Assumptions

A synchronous distributed system is considered, where a fixed number of components exchange information through a synchronous communication channel. The use of a real-time network (CAN), together with state-of-the-art schedulability analysis techniques [4], allows the system to be considered synchronous, even in the presence of temporary periods of inaccessibility [6], [7], [8].

In the assumed network model, temporary failures are a consequence of either bus errors or network interface (transceiver) errors. Such network failures have the following semantics:

- Bus error bursts never affect more than n transmissions during the interval of analysis T. This means that, even for the case of multiple sources of errors, the time interval during which the network is inaccessible is upper-bounded.
- Transceivers either behave correctly or crash after a given number of failures, during the interval of analysis T. This behavior is guaranteed by the CAN protocol since, in the case of multiple errors, the node goes first to the *Error-Passive* state and then to the *Bus-Off* state [1].
- Multicasts may fail either by inconsistent message omissions or inconsistent message duplicates. All other errors are detected (and failed messages retransmitted) by the CAN built-in error detection and recovery mechanisms, with a sufficiently high probability [1].
- A single message can be disturbed by at most k_{dup} duplicates. As the probability of an inconsistent message duplicate is approximately 10^{-4} (the transmission of 2.87×10^7 messages per hour results in, at most, 2.84×10^3 duplicate messages [9]), the necessity of k_{dup} being considered greater than 2 is not foreseen.
- During a time interval T, greater than the worst-case delivery time of any message, at most one single inconsistent message omission occurs in the network. Considering the existence of 3.98×10^{-9} to 2.94×10^{-6} inconsistent message omissions per hour [9], the occurrence of a second omission error in a period T of, at most, several seconds has an extremely low probability.
- There are no permanent medium faults, such as the partitioning of the network. This type of faults must be masked by appropriate network redundancy schemes.

As the target of the proposed middleware is to tolerate network-related faults, nodes are assumed to be fail-silent in what concerns communications, that is, it is assumed that all communication requests performed by any node are correct. It is also considered that protocol software does not fail by producing incorrect messages (with either value or timing faults). These assumptions may be quite restrictive when considering faults at the software or processor level. Nevertheless, it is considered that more stringent assumptions can only be covered through the use of network redundancy schemes (such as those proposed in [18], [19]), with the provision of special-purpose hardware and/or software with memory protection schemes, in order to provide a fail-silent behavior to the node.

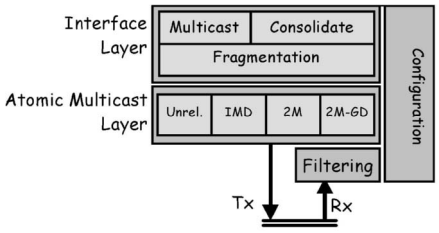
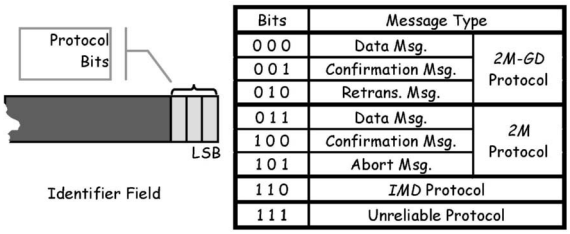


Fig. 4. Middleware structure.



Bits	Message Type	
000	Data Msg.	2M-GD Protocol
001	Confirmation Msg.	
010	Retrans. Msg.	
011	Data Msg.	2M Protocol
100	Confirmation Msg.	
101	Abort Msg.	
110	IMD Protocol	
111	Unreliable Protocol	

Fig. 5. Identifier field and protocol information.

4 Middleware for Reliable Real-Time Communication in CAN

The structure of the proposed middleware is presented in Fig. 4. The Filtering module enforces that nodes not registered to receive a particular message stream will not be able to process messages related to that stream, decreasing the number of messages in error situations. The Atomic Multicast Layer provides a set of multicast protocols, ranging from an unreliable one to a protocol that guarantees an ordered delivery in the presence of inconsistent message omissions.

The Interface Layer provides a group abstraction to the upper layers in order to abstract them from both implementation details and membership and location issues. Consolidation of replicated components and fragmentation and concatenation of messages are also provided by this layer. The Fragmentation module allocates a range of CAN identifiers for each message stream requiring fragmentation. The resulting fragments are treated as independent messages by the atomic multicast protocols.

The Configuration module is responsible for storing all the information required for the correct functioning of the protocols. This information concerns which identifiers a node is registered to receive, which atomic multicast protocol to use when a multicast request is received, the fragment information, and the information required to consolidate replicated messages.

As the target of the presented work is to support reliable hard real-time applications, the proposed middleware considers the existence of a fixed set of processing units and, thus, the full set of message streams and their characteristics (periodicity, size, and replication) are previously known. This offline knowledge is required to allow the use of state-of-the-art response time analysis techniques, to provide the desired real-time guarantees to the supported applications.

A group communication approach is not considered for the communication layer level in order to avoid burdening the CAN identifier field with source and destination information, especially as CAN multicast capabilities already provide location transparency. Nevertheless, a group communication support is provided to the upper levels, enabling upper layers to be loosely dependent on the used communication infrastructure.

As the focus of this paper is on the provided reliable real-time communication mechanisms, only the Atomic Multicast and Consolidation protocols will be presented. The Atomic Multicast module provides a set of protocols, with different failure assumptions and different behaviors in the case of errors. The IMD (Inconsistent Message Duplicate) protocol provides an atomic multicast that just addresses the inconsistent message duplicate problem. The 2M (Two Messages) protocol provides an atomic multicast addressing both inconsistent message duplicates and omissions, where messages are not delivered in error situations. Finally, the 2M-GD (Guaranteed Delivery) protocol is an improvement of the 2M protocol, which guarantees the message delivery if at least one node has correctly received it. The Unreliable protocol is a simple multicast protocol that does not provide any guarantees.

These atomic multicast protocols provide the system designer with the possibility of trading efficiency by reliability since they can be simultaneously used in the same system. The IMD protocol uses less bandwidth, but it does not cover the inconsistent omission failure assumption. On the other side, the use of protocols with higher assumption coverage (e.g., the 2M protocol) introduces higher overheads. Hence, streams with higher criticality may use protocols with higher assumption coverage, while streams with smaller criticality may use lighter protocols.

The less significant bits of the frame identifier are used to carry protocol information (Fig. 5), identifying the message type without interfering with the message criticality (defined by the most significant bits of the frame identifier).

Knowing that CAN frames are simultaneously received in every node, the atomic multicast properties are guaranteed by delaying the delivery of a received frame during a specific (bounded) time. The proposed approach is similar to the Δ-protocols [20], where delivery order is guaranteed by delaying the message delivery during a specific time interval (Δ). The difference is that delivery delays are now evaluated on a stream by stream basis, increasing the system throughput, as messages are delayed according to their specific worst-case response times. It is assumed that clocks are approximately synchronized by the use of an appropriate protocol (such as [14]), guaranteeing the correct evaluation of the delivery delays.

4.1 IMD Protocol

The IMD protocol provides an atomic multicast addressing just the inconsistent message duplicate problem. As presented in Section 2.4, the transmitter automatically retransmits a message if at least one node has signaled an error, even if some nodes have correctly received the message. In order to correctly manage these duplicates, every node, when receiving a message marks it as unstable, tagging it with a $t_{deliver}$ stamp (current time plus a $\delta_{deliver}$ delay).

This delay guarantees that if the message is retransmitted, the retransmission will be received before delivering the message to the application. If a duplicate is received before $t_{deliver}$ (Fig. 6), the duplicate is discarded and $t_{deliver}$ is updated. This updating is required since other nodes may have not received the original message, thus their $t_{deliver}$

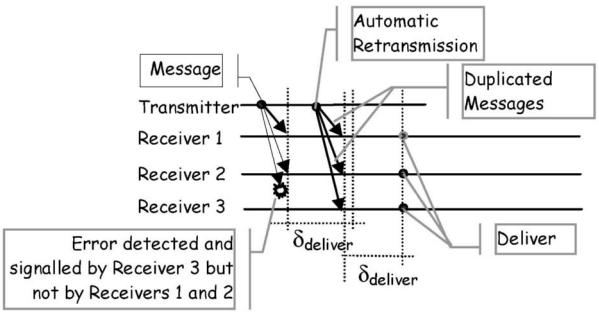


Fig. 6. IMD protocol in the presence of an inconsistent duplicate.

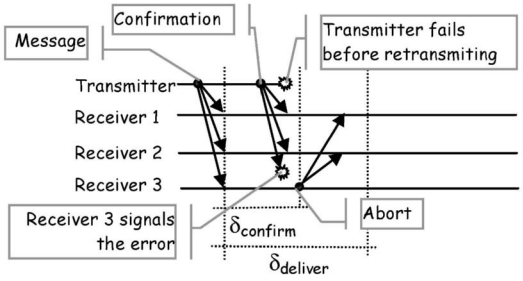


Fig. 7. 2M protocol in the presence of an inconsistent omission while sending the confirmation.

refers to the duplicate. If a node does not receive any duplicate before $t_{deliver}$, it delivers the message to the application as it knows that all other nodes have correctly received the message (no retransmission was required).

The transmitter (if it also delivers the message) can itself deliver the message after $\delta_{deliver}$. As the CAN controller will only acknowledge the transmission when every node has correctly received it (no more retransmissions), there will be no duplicates.

4.2 2M Protocol

The 2M protocol addresses both inconsistent message duplicates and inconsistent message omissions, guaranteeing that either all or none of the receivers will deliver the message. For the latter, not delivering a message is equivalent to a transmitter crash before sending the message.

In the 2M protocol, a node wanting to send an atomic multicast transmits the data message, followed by a confirmation message, which carries no data. When a message is received, the node marks it as unstable, tagging it with $t_{confirm}$ and $t_{deliver}$ stamps. A node receiving a duplicate message discards it, but updates both $t_{confirm}$ and $t_{deliver}$. As the data message has higher priority than the related confirmation (Fig. 5), then all message duplicates will be received before the confirmation. Duplicate confirmation messages will just confirm an already confirmed message.

A receiving node must receive both the message and its confirmation before delivering the message. If it does not receive the confirmation before $t_{confirm}$ (Fig. 7 presents an example of an inconsistent confirmation message), it multicasts the related abort frame. This implies that several aborts can be simultaneously sent (at most one from each consumer node). A message is delivered after $t_{deliver}$ if the node did not receive any related abort frame (a node receiving a message but not the confirmation does not know if the transmitter has failed while sending the message or while sending the confirmation).

The advantage of the 2M protocol is that, in a fault-free execution behavior, there is only one extra frame (without data) transferred in the bus per multicast. The number of protocol-related messages will be increased only in an error case (low probability). Note that the transmission of an abort only occurs in the case of a previous transmitter failure. Therefore, from the failure assumptions presented in Section 3.3 (there is no second inconsistent message omission in the same period T), this abort will be free of inconsistent message omissions.

The transmitter can automatically confirm the message since, if it does not fail, every node will correctly deliver the message and the confirmation. The situation is the same as for the IMD protocol since, if the transmitter remains correct and delivers the message, then it will retransmit any failed message.

4.3 2M-GD Protocol

The 2M protocol can be modified to guarantee the delivery of a transmitted message if at least one node has correctly received it. In the 2M-GD protocol, nodes receiving the message but not the confirmation (Fig. 8) retransmit the message (instead of an abort). This protocol is, however, less efficient than the 2M protocol (in error situations) since messages are retransmitted with the data field. To guarantee the delivery order, it is necessary to use a $t_{deliver\_after\_error}$ stamp to solve inconsistent retransmission duplicates. When a protocol retransmission is received, the node tags it with $t_{deliver\_after\_error}$ to delay the delivery of the message, until it can guarantee that no more retransmission duplicates will be received.

4.4 Replica Consolidation

The Consolidate module is built on top of the atomic multicast protocols. By using atomic multicasts, it is guaranteed that every replica of a replicated component receives the same set of messages in the same order. The Consolidate protocol delays the *decide* phase until it knows that it has received the full set of messages (Fig. 9 top) or until a specific time (δ_{decide}) has elapsed (Fig. 9 bottom).

Note that this protocol can be used to implement the *many-to-1* and *many-to-many* communication exchanges. In the particular case of *many-to-1* communication, there is no

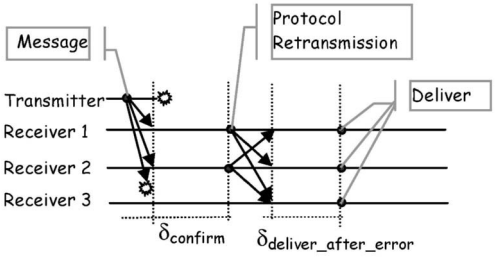


Fig. 8. 2M-GD protocol, in the presence of an inconsistent omission while sending the message.

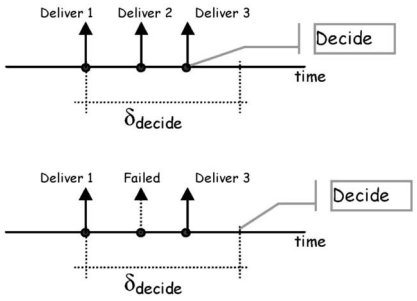


Fig. 9. Consolidate in error-free (top) and error (bottom) situations.

need to solve the inconsistent message omission problem since just one node will deliver the message. However, it is still necessary to address the inconsistent message duplicate problem as the receiving node may receive duplicate messages. Thus, it is sufficient to use the IMD protocol.

4.5 Guaranteeing Reliable Communication Properties

Defining a correct node as a node that does not fail while a multicast is in progress, the atomic multicast properties (defined in Section 3.2) are guaranteed by the proposed multicast protocols since:

- Validity: As the CAN built-in mechanisms guarantee that any message will be automatically retransmitted in the case of either a network or a receiving node failure [1], then the Validity property is guaranteed.
- Agreement: For the case of a correct transmitter, the Agreement property is guaranteed by the Validity property since the transmitter will retransmit any failed message until all the nodes have correctly received it.

  For the case of an incorrect transmitter (leading to omissions), only the 2M and 2M-GD protocols need to be examined (note that the IMD protocol is not relevant as it does not prevent inconsistent message omissions):

  1. For the 2M protocol, a correct node will only deliver a message after receiving the confirmation and being sure that it will not receive any abort. Thus, it knows that all other correct nodes have also received both the message and the confirmation and will deliver the message. Any node that does not receive both the message and the confirmation will send an abort message that, according to the failure assumptions (no second message omission during the interval of analysis), will prevent other nodes from delivering the message.
  2. For the 2M-GD protocol, the behavior is equivalent, except in error situations, where any node not receiving the confirmation will retransmit the message. All nodes will receive this retransmission, once again because there is no second message omission.
- Integrity: As all the correct nodes delay the delivery of the message, duplicates are discarded and the Integrity property is guaranteed. On the other side, the CAN built-in mechanisms guarantee that a message is from the actual sender since a bit error in the identifier field is detected with a sufficiently high probability [1].
- Total Order: Both the cases of message duplicates and omissions must be examined. Considering the case of message duplicates, Total Order is guaranteed since any correct node will discard any previous message when receiving a duplicate. This protocol behavior implies that the delivery of any message will be referred to the last received duplicate, which is the one that is simultaneously received by all nodes (if not, there would be a retransmission).

  In the case of inconsistent message omissions, as for the Agreement property, only the 2M and 2M-GD protocols need to be examined:

  1. For the 2M protocol, the Agreement property guarantees that no node will deliver the message. Thus, order is preserved.
  2. For the 2M-GD protocol, in an inconsistent omission situation, nodes receiving the message will retransmit it. Since the occurrence of more than one retransmission is treated as a duplicate, the delivery of any message will be referred to the last received duplicate. Thus, order is also preserved.

Finally, The Consolidate protocol must guarantee the Agreement property of consolidation. However, this property is only necessary for the case *of many-to-many communication*. In this case, by using an atomic multicast protocol to disseminate the proposed values, it is guaranteed that all replicated receiving components will have the same set of proposed values and by the same order. Therefore, they will all decide on the same value. As for the Validity property, it is related to the *decide* function being used, not to the Consolidate protocol itself.

5 Response Time Analysis

In order to guarantee the timing requirements of the supported applications, it is necessary to previously analyze the response time of the proposed protocols. Such response time analysis is constrained by the imposed delays to the delivery and consolidation phases, which must therefore be carefully evaluated.

Also, some (or all) of these message streams may involve the exchange of extra messages in the network, either from errors (duplicate messages) or from protocol-related messages (confirmation, abort, and retransmission messages), which must also be integrated in the response time analysis. Extra messages related to a message stream S_m are respectively referred to as S_m^{dup}, S_m^{conf}, S_m^{ab}, and $S_m^{retrans}$.

This section provides the preruntime schedulability conditions necessary to analyze the responsiveness of the proposed protocols, considering both the related response and delivery times. The *Response Time* is comprised of the time interval between requesting a message transfer up to its complete reception at the receiver side. The *Delivery Time* is comprised of the time interval between requesting a message transfer until the message is delivered to the upper layers of the receiver side. If multicast protocols are not

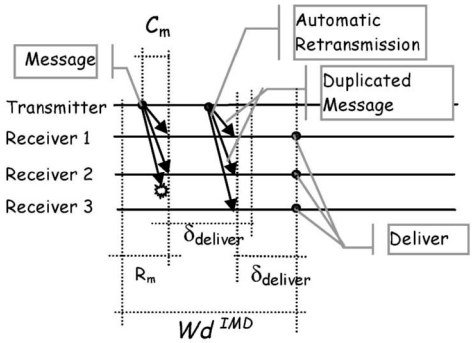


Fig. 10. IMD protocol with one duplicate message.

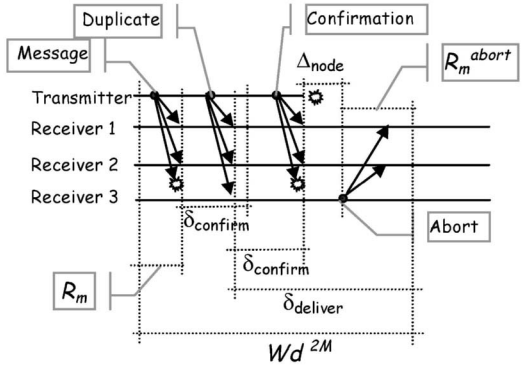


Fig. 11. 2M protocol with message duplicate followed by confirmation omission.

used, these times are equivalent as messages are delivered when they are correctly received.

This analysis does not consider execution delays caused by the protocol execution in each node. However, these can be easily integrated since they can be bounded through the use of the same response time analysis [10] as for the application software.

5.1 Response Time Analysis of the IMD Protocol

The IMD protocol delay ($\delta_{deliver}$) guarantees that a message is only delivered when it is known that there will be no more duplicates. As the receiving node must evaluate such delay based on local information, it must take the arrival instant as its time reference. It then delays the message delivery during the time interval it takes to completely retransmit a failed message. In the presence of a duplicate message (Fig. 10), $\delta_{deliver}$ is reset.

Thus, $\delta_{deliver}$ must be greater than or equal to the worst-case response time of the duplicate message. This response time is equivalent to the worst-case response time of the original message (as it has the same priority), considering that the retransmission does not suffer any blocking (as the transmitter immediately tries to retransmit the failed frame). Thus, R_m^{dup} is evaluated considering that $B_m^{dup} = 0$ and then:

$$\delta_{deliver} = R_m^{dup}. \tag{13}$$

The worst-case delivery time (Wd_m^{IMD}) for a message stream S_m must consider the delay introduced by the possible existence of k_{dup} duplicates:

$$Wd_m^{IMD} = R_m + (k_{dup} + 1) * \delta_{deliver}. \tag{14}$$

The best-case delivery time (Bd_m^{IMD}) considers that the message is transmitted with its best-case response time, that is, there is no interference or blocking and no duplicates are transmitted:

$$Bd_m^{IMD} = C_m + \delta_{deliver}. \tag{15}$$

5.2 Response Time Analysis of the 2M Protocol

For the 2M protocol, it is considered that both the message and the confirmation are put in the transmission queue atomically. As the message has higher priority than the confirmation (Fig. 5), it will be scheduled ahead. Thus, the confirmation will not suffer any blocking from lower priority messages and, as the arrival instant of the message is taken as the time reference, the confirmation will not suffer any interference from the related message. Thus, $\delta_{confirm}$ must be set to, at least:

$$\delta_{confirm} = R_m^{conf} - C_m, \tag{16}$$

where R_m^{conf} is evaluated with $B_m^{conf} = 0$ (no blocking from lower priority messages).

The $\delta_{deliver}$ interval must be determined considering that every receiver waits until it is known that it will not receive any abort message (from any node not receiving the confirmation message before $\delta_{confirm}$). Therefore, the $\delta_{deliver}$ interval must also consider the response time of the node (Δ_{node}) to generate an abort message request:

$$\delta_{deliver} = \delta_{confirm} + \Delta_{node} + R_m^{abort}. \tag{17}$$

Note that several abort messages may be transmitted in the network, related to the same omission error. To determine the $\delta_{deliver}$ interval, it is just necessary to consider the case of the latest abort message (the one sent by the node with the largest Δ_{node}). If no abort message has been received during such $\delta_{deliver}$ interval, then the message is stable and can be delivered. The possible existence of several aborts in the network must be properly considered for the response-time evaluation of lower priority messages.

The worst-case delivery time (Wd_m^{2M}) of a message stream S_m considers that a message is transmitted both with its worst-case response time and with k_{dup} duplicates, thus resetting both $\delta_{confirm}$ and $\delta_{deliver}$ (Fig. 11). Therefore, the worst-case delivery time considers an extra $\delta_{confirm}$ for each duplicate message:

$$Wd_m^{2M} = R_m + k_{dup} * \delta_{confirm} + \delta_{deliver}. \tag{18}$$

The best-case delivery time (Bd_m^{2M}) considers that the message is transmitted with its best-case response time and that there are no duplicates or omissions:

$$Bd_m^{2M} = C_m + \delta_{deliver}. \tag{19}$$

5.3 Response Time Analysis of the 2M-GD Protocol

The 2M-GD protocol has a similar behavior to the 2M protocol. For $\delta_{confirm}$ and $\delta_{deliver}$, it is only necessary to replace the worst-case response time of the abort message (R_m^{abort}) in (17) with the worst-case response time of the

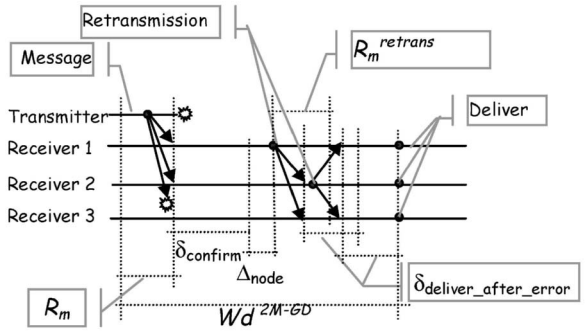


Fig. 12. 2M-GD Protocol with retransmissions.

retransmitted message (which is equal to the worst-case response time of the original message). Multiple retransmissions need also to be considered for the response time evaluation of lower priority messages.

However, an extra delay $\delta_{deliver\_after\_error}$ must also be determined (Fig. 12) since a receiving node cannot guarantee that the other nodes have correctly received the retransmitted message (due to the inconsistent retransmission duplicates). Thus, a similar approach to the IMD protocol is followed, delaying the delivery until it is guaranteed that all duplicates have been correctly received. Hence, $\delta_{deliver\_after\_error}$ is equal to the worst-case response time of a duplicated retransmitted message:

$$\delta_{deliver\_after\_error} = R_m^{retrans}, \tag{20}$$

where $R_m^{retrans}$ is evaluated with $B_m^{retrans} = 0$ (no blocking from lower priority messages).

The worst-case delivery time ($Wd_m^{2M\text{-}GD}$) of a message stream S_m considers that both an inconsistent message omission and duplicate retransmissions occur (once again, the response time of duplicate retransmissions is determined without blocking since duplicates are immediately rescheduled). The existence of multiple retransmissions must also be considered (Fig. 12), where a node receiving a second retransmission will consider it as a duplicate retransmission and will reset $\delta_{deliver\_after\_error}$. Therefore, the worst-case delivery time considers the maximum number of retransmissions (where n_m^{rec} is the number of receivers of stream S_m):

$$\begin{aligned} Wd_m^{2M-GD} = {} & R_m + k_{dup} * \delta_{confirm} + \delta_{deliver} \\ & + (n_m^{rec} + k_{dup}) * \delta_{deliver\_after\_error}. \end{aligned} \tag{21}$$

The best-case delivery time ($Bd_m^{2M\text{-}GD}$) of a message stream S_m is similar to the case of the 2M protocol:

$$Bd_m^{2M-GD} = C_m + \delta_{deliver}. \tag{22}$$

5.4 Response Time Analysis of the Consolidate Protocol

The Consolidate protocol has to delay the *decide* phase (δ_{decide}) until it knows that it will not receive any further messages (Fig. 9). This delay depends on the worst-case delivery time of the related messages, referring to an initial time reference common to all sending nodes. This common time reference must be the release time of the sending tasks. Therefore, the worst-case delivery time of the messages must consider the worst-case response time of the replicated sending tasks.

If the replicated sending tasks are periodic, then their release time is common in all the nodes (with a small jitter as clocks are just approximately synchronized). If these tasks are sporadically released by external events, then their release time must also be agreed upon between the different replicas (to guarantee a deterministic execution). Thus, this agreed time instant must be considered as the time reference for the worst-case response time of the related messages. If the replicated tasks are sporadically released by other tasks, then their release time may vary between replicated components. In this case, it is necessary to consider the release time of the initial replicated tasks with common release time and determine the worst-case response time of the replicated sending tasks related to that initial time reference.

Knowing the worst-case delivery time for each related message, an upper bound for δ_{decide} may be determined assuming the best-case delivery time for the first message to arrive and the worst-case delivery time for the last message to arrive. Therefore:

$$\delta_{decide} = \max_{\forall i \in rep(m)} \{W_i^{com}\} - \min_{\forall i \in rep(m)} \{B_i^{com}\} + \varepsilon, \tag{23}$$

where W_i^{com} is the worst-case delivery time of message stream i and B_i^{com} is the best-case delivery time of message stream i, both considering a common time reference, and $rep(m)$ is the set of replicated message streams. ε is the maximum clock deviation.

The best-case delivery time (Bd_m^{decide}) of the consolidated message can be determined, considering that all messages are received with their best-case delivery time. Therefore:

$$Bd_m^{decide} = \max_{\forall i \in rep(m)} \{B_i^{com}\} + \varepsilon. \tag{24}$$

The worst-case delivery time (Wd_m^{decide}) depends on the assumed set of failure assumptions. If some of the replicated messages are not delivered (due to inconsistent message omissions or failed sending tasks), the protocol must wait during a δ_{decide} interval from the arrival of the first message (Fig. 9 bottom). Therefore, considering that f related messages are not delivered, the worst-case delivery time of the consolidated message is:

$$Wd_m^{decide} = \min_{\forall i \in rep'(m)} \{W_i^{com}\} + \delta_{decide}, \tag{25}$$

where $rep'(m)$ is the set of replicated message streams, excluding the f streams with smaller worst-case delivery time.

5.5 Integrating Communication Overheads in the Response Time Analysis

The response time analysis of CAN networks must now be updated to integrate the overheads concerning confirmation messages and possible aborts or retransmissions. Therefore, the worst-case queuing delay (8) must be updated to consider periods of interference from higher priority message streams using atomic multicast protocols:

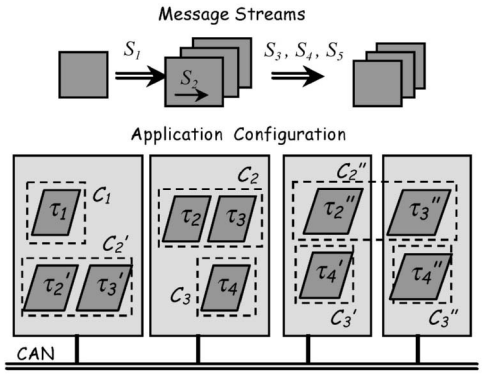


Fig. 13. Application example.

$$I_m = B_m + \sum_{\forall j \in hp(m)} \left(\left\lceil \frac{I_m + \tau_{bit}}{T_j} \right\rceil \times (C_j + C_j^{extra}) \right) + Ina(I_m) + \max_{\forall j \in hp(m)} \{extra\_msg_j\}, \quad (26)$$

where C_j^{extra} is the interference caused by the confirmation message, which is:

$$C_j^{extra} = \begin{cases} C_j^{conf} & 2M \text{ or } 2M\text{-}GD \text{ protocol} \\ 0 & otherwise. \end{cases} \quad (27)$$

Additionally, $max\{extra\_msg_j\}$ accounts for the aborts or retransmissions in the network due to inconsistent message omissions. As a single inconsistent message omission is assumed during the period of analysis T (greater than the largest worst-case delivery time), each receiver of message stream S_j will transmit, at most, one abort/retransmission due to inconsistent message omissions, that is:

$$extra\_msg_j = \begin{cases} n_j^{rec} * C_j^{abort} & 2M \text{ protocol} \\ n_j^{rec} * C_j^{retrans} & 2M\text{-}GD \text{ protocol} \\ 0 & otherwise, \end{cases} \quad (28)$$

where n_j^{rec} is the number of receivers for the message stream S_j.

The $Ina(I_m)$ term (26) integrates the periods of network inaccessibility caused by errors in frame transmission, therefore it already includes the retransmissions of inconsistently failed messages (that is, duplicates).

Considering the network utilization, (12) must also be updated. For each message stream transmitted with the 2M or 2M-GD protocol, an extra confirmation message must be considered (C_m^{extra}, (27)). Also, the maximum number of extra messages (related to inconsistent message omissions) per period of analysis T must be considered:

$$U = \left(\sum_{\forall m} \frac{C_m + C_m^{extra}}{T_m} \right) + U_{ina} + \frac{\max_{\forall m}\{extra\_msg_m\}}{T}. \quad (29)$$

6 Numerical Example

In order to clarify the use of the proposed protocols, consider a system where a distributed hard real-time application executes (Fig. 13). The system is constituted by four nodes, connected by a CAN network at a rate of 1 Mbit/sec.

TABLE 1
Tasks' Characteristics

Task	Type	WCET	Period	Comp.	Nodes
τ_1	Periodic	2	5	C_1	1
τ_2	Periodic	2	10	C_2	1,2,3
τ_3	Sporadic	3	10	C_2	1,2,4
τ_4	Periodic	4	15	C_3	2,3,4

The application is constituted by four tasks $(\tau_1 \ldots \tau_4)$, which are spread over the nodes. As component replication is also used, then some of these tasks are also replicated. In this simple application, each task outputs its results to the following task. The application is divided into three components: component C_1 encompasses task τ_1, component C_2 encompasses τ_2 and τ_3, and, finally, component C_3 is just τ_4. Components C_2 and C_3 are replicated in three replicas, while component C_1 is not replicated (Fig. 13).

Table 1 presents each task's characteristics, while Table 2 presents the characteristics of the necessary message streams (all values are in milliseconds).

Note that messages from τ_2 to τ_3 and τ_2' to τ_3' are internal to the node since both tasks are in the same node. As message stream S_1 is a *1-to-many communication*, the 2M-GD protocol is used in order to guarantee that every replica of task τ_2 delivers the message. Therefore, there will be an extra confirmation message with the same period of message stream S_1, but without data. Since it is considered that an inconsistent message omission may occur, then it is also necessary to account for three possible retransmissions (one from each receiving node).

Message stream S_2 is internal to a component (although the component being spread between nodes 3 and 4) and it is a *1-to-1 communication*. Therefore, it is sufficient to use the IMD protocol since only duplicates are of concern. Message streams S_3 to S_5 are messages from replicated τ_3 to replicated τ_4, therefore they need consolidation in every replica of τ_4. As this consolidation will mask node failures of the senders, then it is sufficient to use the 2M protocol for the transmission of messages. Therefore, there will be an extra confirmation message for each message sent (and possible abort messages).

The following assumptions are considered:

- a maximum of two message faults in each 10 ms time interval, resulting from a bit error rate of approximately 10^{-4}, which is an expectable range for bit error rates in aggressive environments [9];

TABLE 2
Messages Streams' Characteristics

Stream	Bytes	Period	From	To	Prot.
S_1	4	5	τ_1	τ_2,τ_2',τ_2''	*2M-GD*
S_2	8	10	τ_2''	τ_3''	*IMD*
S_3	6	10	τ_3	τ_4,τ_4',τ_4''	*2M*
S_4	6	10	τ_3'	τ_4,τ_4',τ_4''	*2M*
S_5	6	10	τ_3''	τ_4,τ_4',τ_4''	*2M*

TABLE 3
Meesage Streams' Response Time with Protocols

Stream	T_m (ms)	C_m (ms)	R_m^{NP} (ms)
S_1	5	0.089	0.519
S_2	10	0.127	0.630
S_3	10	0.108	0.741
S_4	10	0.108	0.852
S_5	10	0.108	0.852
Ina	10	0.300	-
U		9.29 %	

- one inconsistent message omission during the period of analysis;
- one duplicate in the transmission of a message ($k_{dup} = 1$);
- a Δ_{node} equal to 100 μs and a maximum deviation between clocks (ε) of 100 μs.

Table 3 presents the response time for each message stream and the network load when multicast protocols are not used (the Unreliable protocol is used instead of the IMD/2M/2M-GD protocols). R_m^{NP} represents the worst-case response time (NP: no protocols), P is the periodicity, and C_m is the actual time taken to transmit a message. An inaccessibility time of 0.3 ms ($2 * t_{ina}$, where t_{ina} is evaluated using (6)) during a time interval of 10 ms is a consequence of the assumption of two faulty messages in the network. The network utilization (U) is determined using (12), thus accounting for both the network utilization of the message streams and the inaccessibility periods.

As can be seen, the worst-case response time of messages is considerably greater than its actual transmission time. Although the interference from higher priority messages is one of the factors leading to such a difference, the main factor is the bit error rate. For instance, a message of stream S_1 in an error-free environment would have a worst-case response time of 0.219 ms (instead of 0.519 ms as in Table 3). The possible existence of errors in the network more than duplicates its worst-case response time [7], even not using multicast protocols.

Tables 4 and 5 present the messages' delays and delivery times, considering the use of the proposed multicast protocols. R_m^{MP} represents the worst-case response time of a message stream when multicast protocols (MP) are considered. Wd_m and Bd_m are, respectively, the worst and best-case delivery time for message stream S_m.

As can be seen in Table 5, the worst-case delivery time is greater than the related worst-case response time because, apart from the multicast-related delays, it is assumed that each message may be disturbed by, at most, one duplicate. For instance, the worst-case delivery time for message stream S_5 is not only given by the message stream response time plus its $\delta_{deliver}$, but also by summing an extra $\delta_{confirm}$ due to a message duplicate.

The last column of Table 5 presents the ratio worst-case delivery time/worst-case response time when considering the use of multicast protocols. It is obvious that the IMD protocol is the one that introduces smaller delays (message stream S_2), while the 2M-GD protocol is the one with the higher delays (message stream S_1). Therefore, the system designer can use this reasoning to balance reliability versus efficiency in the system.

TABLE 4
Protocol-Related Delays

Stream	Prot.	$\delta_{confirm}$	$\delta_{deliver}$	$\delta_{del\_aft\_er}$
S_1	2M-GD	0.350	0.969	0.389
S_2	IMD	-	0.848	-
S_3	2M	0.901	2.013	-
S_4	2M	1.065	2.341	-
S_5	2M	1.229	2.558	-

Also presented in Table 5 is the network utilization considering the use of the proposed protocols (29). As can be seen, these protocols increase network utilization less than 30 percent since multicast-related retransmissions only occur in inconsistent message omission situations. Although this network load increase is still large, it is much smaller than in other software-based approaches and it is strictly necessary to cope with inconsistent message omission using a software-based approach.

Since messages from replicated tasks τ_3 to replicated tasks τ_4 need to be consolidated, it is still necessary to determine the δ_{decide} parameter of the Consolidate protocol. As stated, it is necessary to find the worst-case and best-case delivery times for each one of the message streams (S_3 to S_5). However, these delivery times must refer to a common time base. Thus, it is necessary to determine the best-case and worst-case response time of replicated tasks τ_3. This task is a sporadic task released by τ_2. Hence, its response time is dependent of the response time of τ_2. These response times can be easily determined using the analysis presented in [10]. However, replicated component C_2'' is spread over nodes 3 and 4. Thus, to determine the worst and best-case response times of τ_3'', it is necessary to consider the delivery time of message stream S_2.

Table 6 presents the best-case and worst-case response time of replicated tasks τ_3 and the associated worst-case and best-case delivery time of messages streams S_3 to S_5 (all referring to the common release time of task τ_2). Therefore, (from (23)):

$$\delta_{decide} = 13.64 - 7.121 + 0.1 = 6.619\ ms. \tag{30}$$

The worst-case delivery time of the consolidation (referring to the time instant where the first message is scheduled for transmission) is determined assuming that all messages but one are delivered at their worst-case delivery time and the other is not delivered. In this case (assuming that the message not delivered is the one with the lower worst-case response

TABLE 5
Message Streams' Delivery Time Considering Protocols

Stream	Prot.	R_m^{MP}	Wd_m	Bd_m	Wd_m/R_m^{MP}
S_1	2M-GD	0.519	3.394	1.058	6.54
S_2	IMD	0.959	2.655	0.975	2.77
S_3	2M	1.070	3.984	2.121	3.72
S_4	2M	1.234	4.640	2.449	3.76
S_5	2M	1.287	5.074	2.666	3.94
U			11.79%		

TABLE 6
Consolidation

Task	WCRT	BCRT	Stream	Wd^{com}	Bd^{com}
τ_3	5	5	S_3	8.984	7.121
τ_3'	9	7	S_4	13.64	9.449
τ_3''	7.655	5.975	S_5	12.729	8.641

time (S_3)), the message from stream S_4 will arrive with its worst-case delivery time of 4.640, which, summed to the δ_{decide}, gives a worst-case consolidation time of:

$$Wd^{decide} = Wd_{M4} + \delta_{decide} = 11.259\ ms. \quad (31)$$

Although delays are induced by this consolidation, the advantage is that no extra overhead is introduced in the network, preserving the predictability of the system. As one of the main targets of the proposed multicast protocols is to provide reliable CAN communication, preserving CAN real-time characteristics (allowing the offline analysis of messages' response times), such a target is achieved as the predictability of message transfers is guaranteed.

7 Comparison with Other Approaches

The problem of inconsistent message delivery in CAN networks has been given some research in the last years. In [9], a set of fault-tolerant broadcast protocols is proposed which solve the message omission and duplicate problems. The RELCAN protocol is similar to the 2M-GD protocol, being based in the transmission of a second data-free message (CONFIRM message) to signal that the sender is still correct. If this confirm message does not arrive before a specific timeout, the message is retransmitted. This retransmission is performed using a lower layer protocol (EDCAN), which is based on the retransmission of messages by every node in the system (that has correctly received the message).

In the RELCAN protocol, the transmission request of the CONFIRM message is made after receiving information from the CAN controller that the data message has already been sent. This two-phase approach is necessary to guarantee that there is no order inversion between protocol messages, that is, the CONFIRM message is only sent after the related data message. In the 2M (and 2M-GD) protocol, this noninversion guarantee is provided by giving a lower priority to the confirmation message than to its related data message. Therefore, the request for transmission of both the data and confirmation messages can be atomically performed, drastically reducing the worst-case response time of the related message stream, especially for lower priority streams.

As the RELCAN protocol delivers a message as soon as it is received, it is not able to provide the total order guarantee as the 2M-GD protocol (thus, it cannot be used for atomic multicasts). In [9], total order is addressed by the TOTCAN protocol. This protocol is also based on a two-phase approach, where transmission of an ACCEPT message (similar to the CONFIRM message) is performed using the EDCAN protocol. Therefore, multiple retransmissions will occur in normal operation, even if no error occurs. In case of sender failure, the protocol does not deliver the message (it just guarantees that the message is delivered by all or none of the recipients, as in the case of the 2M protocol).

The protocols presented in [9] are targeted to tolerate a generic number j of inconsistent message omissions. Therefore, the lower-layer EDCAN protocol performs at least $j+1$ retransmissions in order to guarantee that every node receives at least one copy of the message, meaning that there will be a significant number of retransmissions in the bus in normal behavior. The possibility of several identical retransmissions being clustered (all transmitted at the same time) allows the protocol overhead to be reduced, particularly for larger values of j (increasing the probability of messages being clustered [9]). However, for a smaller j (and particularly for j equal to 1), this possibility cannot be assumed, particularly for worst-case scenarios. It is expected that some of these messages will not be simultaneously transmitted since sender nodes have distinct processing delays.

Hence, in order to provide total order, the TOTCAN protocol will incur in higher overheads. For instance, when transmitting a fault-free message in a network with four nodes, in addition to the data message there will be the ACCEPT message plus the possible existence of three ACCEPT retransmissions. For a data message with 8 bytes, the overhead is approximately 150 percent, compared to the 40 percent of the 2M protocol. If it is assumed that, in the lower-layer EDCAN protocol, all retransmissions are clustered, there will be still an 80 percent overhead (the message, the ACCEPT message, and only one clustered retransmission).

Another approach presented in the literature is to use a hardware-based solution [21] to prevent message inconsistencies. This approach is based on a hardware error detector which automatically retransmits messages that could potentially be omitted in some nodes. This detector (SHARE) detects the bit pattern that occurs in an inconsistent message failure and automatically retransmits the received frame, even if the transmitter handles this failure.

Although solving the inconsistent message omission problem of CAN, this hardware-based approach does not provide a solution to the total order problem as duplicates may occur (furthermore, inconsistent message omissions are transformed in inconsistent message duplicates). In order to achieve total order, it is necessary to complement this mechanism with an offline analysis approach [5], where hard real-time messages are offline adjusted to never compete for the bus (using fixed time slots). This approach induces smaller delivery times for hard real-time messages, at the cost of an increased burden in the system analysis.

The approaches presented in [9] and [21] always consider the existence of both inconsistent message omissions and duplicates. The work presented in this paper also provides an atomic multicast protocol (the IMD protocol), which, addressing only inconsistent message duplicates, does not introduce any extra overhead. This type of inconsistent message is the most probable one to appear, thus the IMD protocol provides a significant advantage for message streams with less critical requirements.

8 Conclusion

The use of CAN networks to support Distributed Computer Controlled Systems requires not only time-bounded

transmission services, but also the guarantee of consistency for the supported applications. In spite of its built-in error detection/signaling mechanisms, the CAN protocol may cause inconsistencies in the supported applications as messages can be delivered in duplicate or delivered only by a subset of the receivers.

This paper proposes a set of atomic multicast and consolidation protocols upon which the CAN reliable real-time communication is guaranteed. In the proposed approach, atomic multicasts are guaranteed through the transmission of just one extra message (without data) per transferred message while in error-free situations. More protocol-related retransmissions will only be necessary in the case of inconsistent message omissions, which are low probability situations. Inconsistent message duplicates can be solved by a simpler protocol that guarantees total order without requiring any extra message transfer. Moreover, atomic multicast properties are achieved without more overhead than strictly needed for a reliable multicast. Consolidation of replicated inputs is also provided through the use of a consolidate protocol, built on top of the multicast protocols.

These protocols explore the CAN synchronous properties to minimize their runtime overhead and, thus, provide a reliable and timely service to the supported applications. In this paper, the model and assumptions for the evaluation of the message streams' response time of these protocols are also presented, demonstrating that the real-time capabilities of CAN are preserved since predictability of message transfers is guaranteed.

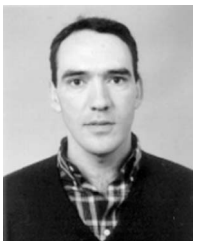
ACKNOWLEDGMENTS

This work was partially supported by FCT (project DEAR-COTS 14187/98) and IDMEC. A preliminary version of this work is included in the *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, June 2001.

REFERENCES

[1] ISO 11898, "Road Vehicle—Interchange of Digital Information—Controller Area Network (CAN) for High-Speed Communication," ISO, 1993.
[2] K. Zuberi and K. Shin, "Scheduling Messages on Controller Area Network for Real-Time CIM Applications," *IEEE Trans. Robotics and Automation,* vol. 13, no. 2, pp. 310-314, 1997.
[3] Rockwell Automation, "DeviceNet Product Overview," Publication DN-2.5, Rockwell, 1997.
[4] K. Tindell, A. Burns, and A. Wellings, "Calculating Controller Area Network (CAN) Message Response Time," *Control Eng. Practice,* vol. 3, no. 8, pp. 1163-1169, 1995.
[5] M. Livani and J. Kaiser, "Evaluation of a Hybrid Real-Time Bus Scheduling Mechanism for CAN," *Proc. Seventh Int'l Workshop Parallel and Distributed Real-Time Systems (WPDRTS '99),* pp. 425-429, Apr. 1999.
[6] S. Punnekkat, H. Hansson, and C. Norstrom, "Response Time Analysis under Errors for CAN," *Proc. 2000 IEEE Real-Time Technology and Applications Symp.,* pp. 258-275, June 2000.
[7] L.M. Pinho, F. Vasques, and E. Tovar, "Integrating Inaccessibility in Response Time Analysis of CAN Networks," *Proc. Third IEEE Int'l Workshop Factory Comm. Systems,* pp. 77-84, Sept. 2000.
[8] N. Navet, Y.-Q. Song, and F. Simonot, "Worst-Case Deadline Failure Probability in Real-Time Applications Distributed over CAN (Controller Area Network)," *J. Systems Architecture,* vol. 46, no. 7, pp. 607-617, 2000.
[9] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues, "Fault-Tolerant Broadcasts in CAN," *Proc. 28th Symp. Fault-Tolerant Computing,* pp. 150-159, June 1998.
[10] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying New Scheduling Theory to Static Priority Pre-Emptive Scheduling," *Software Eng. J.,* vol. 8, no. 5, pp. 285-292, 1993.
[11] J. Rufino and P. Veríssimo, "A Study on the Inaccessibility Characteristics of the Controller Area Network," *Proc. Second CAN Conf.,* pp. 7.12-7.21, Oct. 1995.
[12] "Delta-4—A Generic Architecture for Dependable Distributed Computing," ESPRIT Research Reports, D. Powell, ed., Springer Verlag, Nov. 1991.
[13] S. Poledna, A. Burns, A. Wellings, and P. Barret, "Replica Determinism and Flexible Scheduling in Hard Real-Time Dependable Systems," *IEEE Trans. Computers,* vol. 49, no. 2, pp. 100-111, Feb. 2000.
[14] L. Rodrigues, M. Guimarães, and J. Rufino, "Fault-Tolerant Clock Synchronization on CAN," *Proc. 19th IEEE Real-Time Systems Symp.,* pp. 420-429, Dec. 1998.
[15] P. Veríssimo, A. Casimiro, L.M. Pinho, F. Vasques, L. Rodrigues, and E. Tovar, "Distributed Computer-Controlled Systems: The DEAR-COTS Approach," *Proc. 16th IFAC Workshop Distributed Computer Control Systems,* pp. 128-135, Dec. 2000.
[16] V. Hadzilacos and S. Toueg, "Fault-Tolerant Broadcasts and Related Problems," *Distributed Systems,* S. Mullender, ed., second ed., chapter 5, pp. 97-145, Addison-Wesley, 1993.
[17] D. Powell, "Failure Mode Assumptions and Assumption Coverage," *Proc. 22nd Symp. Fault-Tolerant Computing,* pp. 386-395, July 1992.
[18] J. Rufino, P. Veríssimo, and G. Arroz, "Design of Bus Media Redundancy in CAN," *Proc. Fieldbus Conf. (FeT '99),* pp. 375-380, 1999.
[19] H. Hilmer, H.-D. Kochs, and E. Dittmar, "A CAN-Based Architecture for Highly Reliable Communication Systems," *Proc. Fifth CAN Conf.,* pp. 6.10-6.16, 1998.
[20] F. Cristian, H. Aghili, R. Strong, and D. Dolev, "Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement," *Information and Control,* vol. 118, no. 1, pp. 158-179, 1995.
[21] J. Kaiser and M. Livani, "Achieving Fault-Tolerant Ordered Broadcasts in CAN," *Proc. Third European Dependable Computing Conf.,* pp. 351-363, Sept. 1999.

Luís Miguel Pinho received the BSc, MSc, and PhD degrees in electrical and computer engineering from the University of Porto, Portugal, in 1994, 1997, and 2001, respectively. Since 1996, he has been a teaching assistant in the Department of Computer Engineering, School of Engineering of the Polytechnic Institute of Porto. His main research interests include fault-tolerant real-time systems, real-time system architectures, and real-time programming languages. He is particularly interested in the analysis, design, and implementation of mechanisms for the implementation of fault-tolerant real-time distributed systems. He is a member of the IEEE.



Francisco Vasques received the BSc degree in electrical engineering from the University of Porto, Portugal, in 1987 and both the MSc and PhD degrees in computer science from LAAS-CNRS, Toulouse, France, in 1992 and 1996. Currently, he is an assistant professor in the Department of Mechanical Engineering at the University of Porto. He is also responsible for the real-time systems course within the Electrical and Computer Engineering MSc degree. His research interests include real-time communication systems, real-time factory communications, fault-tolerant systems, and real-time system architectures. Since 1991, he has authored or coauthored more than 40 technical papers in the area of real-time systems. He is a member of the IEEE.