Timing Analysis of Reliable Real-Time Communication in CAN Networks

Luís Miguel Pinho
Department of Computer Engineering
ISEP, Polytechnic Institute of Porto
Porto, Portugal
E-mail: lpinho@dei.isep.ipp.pt

Francisco Vasques
Department of Mechanical Engineering
FEUP, University of Porto
Porto, Portugal
E-mail: vasques@fe.up.pt

Abstract

The Controller Area Network (CAN) is a fieldbus network with real-time capabilities. It is generally considered that CAN guarantees atomic multicast properties, through its extensive error detection/signalling mechanisms. However, there are error situations where messages can be delivered in duplicate by some receivers or delivered only by a subset of the receivers, leading to inconsistencies in the supported applications.

In order to prevent such inconsistencies, a set of atomic multicast protocols is proposed, taking advantage of CAN synchronous properties to minimise its run-time overhead. This paper presents such set of protocols, focusing on the timing analysis of the supported reliable real-time communication. It demonstrates that, in spite of the extra stack of protocols, the real-time capabilities of CAN are preserved, since the predictability of message transfer is guaranteed.

1. Introduction

Controller Area Network (CAN) [1] is a fieldbus network suitable for small-scale Distributed Computer Controlled Systems (DCCS). Several studies on how to guarantee the real-time requirements of messages in CAN networks are available (e.g. [2]), providing the necessary pre-run-time schedulability conditions for the timing analysis of the supported traffic, even for the case of networks disturbed by temporary errors [3].

CAN networks also have extensive error detection and signalling mechanisms, which impose the retransmission of the message when an error is detected. However, it is known that these mechanisms may fail when an error is detected in the last but one bit of the frame [4]. This problem may cause messages to be delivered in duplicate by some receivers or to be delivered only by a subset of the receivers.

This may be disastrous if the CAN network is used to support replicated applications, since these applications require that replicated components provide the same results, when they are correct. Thus, the consistency of the delivered messages must be guaranteed by atomic multicast mechanisms, which guarantee that messages are delivered by all (or none) of the component replicas, and that they are delivered only once. Furthermore, there is the need to agree in the delivery order of multicasts, and to consolidate values from replicated inputs.

The DEAR-COTS (Distributed Embedded ARchitecture using Commercial Off-The-Shelf components) architecture [5] provides a generic framework for distributed hard real-time applications, guaranteeing their timeliness and reliability requirements. In this architecture, CAN is used as the communication infrastructure for reliable real-time communication. A set of protocols is provided in order to guarantee atomic multicasts and consolidation of replicated components' outputs. In this paper, the timing characteristics of the protocols are analysed, demonstrating that the real-time characteristics of CAN are preserved.

The paper is organised as follows. Section 2 presents work related to reliable communication in CAN. Then, Section 3 presents the requirements posed to the communication infrastructure. Section 4 presents the proposed protocols for reliable real-time communication in CAN. Their timing analysis is then described in Section 5, developing the necessary pre-run-time schedulability conditions. Finally, a numerical example is presented in Section 6 and some conclusions are outlined in Section 7.

2. Related work

The use of CAN networks to support DCCS applications requires not only time-bounded transmission services, but also the guarantee of consistency for the supported applications. In spite of the extensive CAN built-in mechanisms for error detection and recovery [1],

there are some known reliability problems [4], which can lead to an inconsistent state of the supported applications.

Such misbehaviour 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, being the value of the last bit treated as 'do not care'. Thus, a dominant value in the last bit does not lead to an error, in spite of violating the CAN rule stating that the last 7 bits of a frame are all recessive.

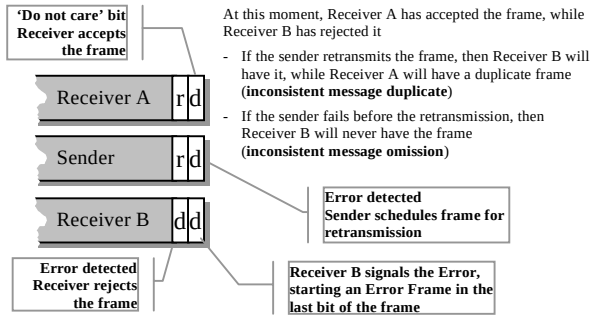

Figure 1. Inconsistency in CAN.

Receivers detecting a bit error in the last but one bit of the frame (Figure 1) reject the frame and send an Error Frame starting in the following bit (last bit of the frame). As for receivers the last bit of a frame is a 'do not care' bit, other receivers may not detect the error and will accept the frame. However, the transmitter re-transmits the frame, as there was an error, thus some receivers will have an *inconsistent message duplicate*. The use of sequence numbers can solve this problem, but it does not prevent messages from being received in different orders. Additionally, if the transmitter fails before being able to retransmit the frame, some receivers will never receive the frame, which causes an *inconsistent message omission.*

In [4], the probability of message omission 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%. Bit error rates were used ranging from 10^{-4} to 10^{-6}, and node failures per hour of 10^{-3} and 10^{-4}. For inconsistent message duplicates the results obtained were 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 mechanisms for error recovery and detection are not sufficient.

Thus, the use of CAN to support reliable real-time communications must be carefully evaluated and appropriate mechanisms must be devised. In [4], a set of fault-tolerant broadcast protocols is proposed, which solve the message omission and duplicate problems. However, such protocols do not take full advantage of the CAN synchronous properties, therefore producing a greater run-time overhead under normal operation. For instance, in the best-case (data message with 8 bytes), the overhead of the total order protocol (TOTCAN) is approximately 150%. The problem is that, in order to achieve ordered multicasts, each receiver must re-transmit an ACCEPT message, even if there is no error. Other protocols in the set do not guarantee total order. Therefore, the overheads introduced by these protocols make them very inefficient.

Another approach would be to use hardware-based solutions, such as the one described in [6]. This approach is based in a hardware error detector, which automatically retransmits messages that could potentially be omitted in some nodes. It solves the inconsistent message omission problem, but, in order to achieve order, it is necessary to restrict hard real-time messages to never compete for the bus, in a time-slotted approach.

3. Communication requirements

In the DEAR-COTS architecture, a hard real-time application is constituted by several tasks (processing units) distributed over the nodes of the system. To guarantee the reliability requirements of applications, parts of it must be replicated, in order to tolerate individual faults. Therefore, the notion of "component" is introduced, which is the replication unit. Each component can include tasks and resources from several nodes, or it can be located in just one node. As an example, Figure 2 shows a real-time application with 4 tasks (τ_1, τ_2, τ_3 and τ_4), and two different components (C_1 and C_2).

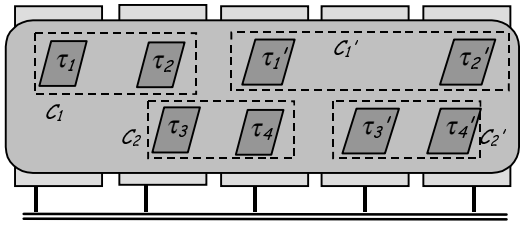

Figure 2. Hard real-time application.

The communication infrastructure must guarantee that all messages sent by correct nodes are orderly delivered to all its recipients. However, there must also be an all-or-none guarantee for the case of a message sent by an incorrect node: either all correct nodes deliver that message, or none of them deliver it. There is also the need to agree in the order by which broadcasts are delivered, and to consolidate values from replicated inputs.

In the DEAR-COTS architecture, there is the need for the following four types of message exchanges: *1-to-1*, *1-to-many*, *many-to-1* and *many-to-many*.

For *1-to-1 communication* (communication from a non-replicated component to another non-replicated component, or communication internal to a component) there is only the need for a reliable multicast, since there is no replication, thus 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 [7] must be used to guarantee that replicated receivers get the same information, in the same order.

When a group of replicated components receives input from another group of replicated components (*many-to-many communication)*, it must agree on the value to use. Thus, an interactive consistency algorithm must be used. Each of the receiving components must receive all the values proposed by the replicated transmitters, and agree on the value to be delivered. 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 agree decision can then be performed by a simple *Consolidate* protocol, that decides on one of the received values (or on some value computed from them).

The case of *many-to-1 communication* is a simplified version of the previous one. The receiving component has only to decide from the set of received inputs. The same *Consolidate* protocol is used, but using only a reliable data transfer from the replicated transmitters to the receiver.

4. Reliable CAN communication

In the DEAR-COTS architecture, the Communication Manager Layer (Figure 3) is the responsible for the reliable and timely communication. The Filtering layer allows that only nodes registered to receive a particular message stream will process messages related to that stream, decreasing the number of messages in error situations.

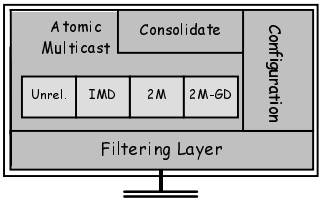

Figure 3. Communication Manager structure.

Relying on CAN frames being simultaneously received in every node, the protocols are based in delaying the deliver of a received frame for a specific (bounded) time. The approach is similar to the Δ-protocols [8], where, in order to obtain order, message delivery is delayed for a specific time (Δ). In the DEAR-COTS approach, delivery delays are evaluated on a stream by stream basis, where messages are delayed accordingly to their worst-case response times, considering the case of a network disturbed by inaccessibility periods [2] [3]. It is also assumed that clocks are approximately synchronised, to guarantee the deterministic behaviour of components [9] and the correct evaluation of the delivery delays.

4.1 Failure assumptions

In the DEAR-COTS architecture it is assumed that:

- 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 [4]), it is not foreseen the necessity of a k_{dup} greater than 2.
- During a time 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 [4], 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 network redundancy schemes.

4.2 Atomic multicasts

The Atomic Multicast module provides several protocols with different failure assumptions and different behaviours in the case of errors.

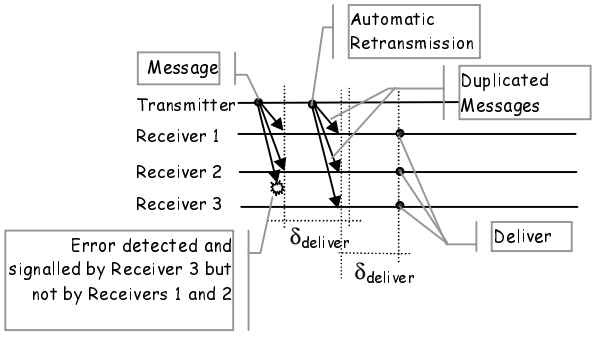

Figure 4. *IMD* protocol.

The *IMD* (Inconsistent Message Duplicate) protocol (Figure 4) provides an atomic multicast that just addresses the inconsistent message duplicate problem. To guarantee that the duplicates are correctly managed, every node, when receiving a message marks it as unstable, tagging it with a $t_{deliver}$ (current time plus $\delta_{deliver}$). If a duplicate is received before $t_{deliver}$, such duplicate is discarded and $t_{deliver}$ is updated (since in a node not receiving the original message, $t_{deliver}$ refers to the duplicate).

The *2M* (Two Messages) protocol addresses both the inconsistent message duplicates and inconsistent message omissions. For the case of inconsistent message omissions it guarantees that either all or none of the receivers deliver the message. In this case, not delivering a message is equivalent to a transmitting node crash before sending the message.

The *2M* protocol is based on the transmission of a confirmation for every multicast sent in the bus, and, if needed, the transmission of related aborts. A node wanting to send an atomic multicast transmits the data message, followed by a confirmation message, which carries no data. A receiving node before delivering the message, must receive both the message and the confirmation. If it does not receive the confirmation before a specific $t_{confirm}$ (Figure 5 presents an example of an inconsistent confirmation message), it multicasts the corresponding abort frame.

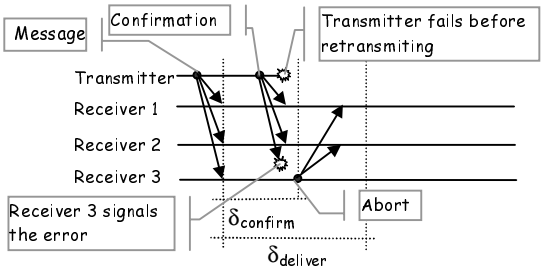

Figure 5. Inconsistent message omission while sending the confirmation (*2M* protocol).

This implies that several aborts can be simultaneously sent (at most one from each receiving node that is interested in that particular message stream). A message is only delivered if the node does not receive any related abort frame until after a specific $t_{deliver}$ (since a node that receives the message but not the confirmation does not know if the transmitter has failed while sending the message or while sending the confirmation).

When a message is received, the node saves it in the set of received messages, and marks it as unstable, tagging it with the $t_{confirm}$ and $t_{deliver}$. A node receiving a duplicate message discards it, but updates both $t_{confirm}$ and $t_{deliver}$. As the data message has a higher priority than the related confirmation, then all duplicates will be received before the confirmation. Duplicate confirmation messages will always be sent before any abort (confirmation messages have higher priority than related abort messages), thus they will confirm an already confirmed message. No update is performed in this case to $t_{confirm}$ and $t_{deliver}$ since they are related to the time of reception of the message, not the confirmation.

The *2M* protocol can be modified to guarantee the delivery of a transmitted message to all the receiving nodes, if it is correctly received by at least one node. In the *2M-GD* (Guaranteed Delivery) protocol, nodes receiving the message but not the confirmation, shall 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 an orderly deliver, it is necessary to use a $t_{deliver\_after\_error}$ to solve inconsistent message duplicates.

4.3 Consolidation of replicated inputs

The Consolidate module is built on top of the atomic multicast protocols, in order to guarantee that every replicated component receives the same set of messages, in the same order. The Consolidate protocol performs the *decide* phase after a specific delay (δ_{decide}), or when it knows that it will not receive further messages (Figure 6).

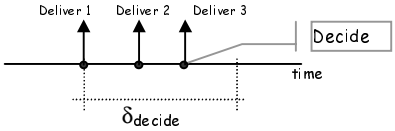

Figure 6. Consolidate in error free situation.

For the case of *many-to-1* communication, there is no need to use underlying protocols that 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 have duplicate messages. Thus, it is sufficient to use the *IMD* protocol.

5. Response time analysis

In order to guarantee the timeliness requirements of real-time applications it is necessary to previously analyse the response time of the proposed protocols. As these protocols are based on delaying of the delivery and consolidation phases, the response time analysis is constrained by the evaluation of these delays.

In the following analysis, it is considered a CAN network with *n* message streams defined as:

$$S_m = (C_m, T_m, D_m) \quad (1)$$

where S_m defines a message stream *m* characterised by a unique identifier. C_m is the longest message duration of stream S_m and T_m is the periodicity of stream S_m requests. In order to have a timing analysis independent from the model of the tasks, it is assumed that this periodicity is the minimum time interval between two consecutive arrivals of S_m 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 request is

placed in the outgoing queue and the instant when the message is delivered.

The response time analysis of CAN networks has been addressed in [2], considering fixed priorities for message streams and a non-preemptive scheduling model. In [3], this response time analysis is extended to integrate temporary periods of network inaccessibility (introduced in [10]). In such analysis, 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. C_m represents the actual transmission time of the message.

Considering the deadline monotonic (DM) priority assignment, the worst-case queuing delay of a message of message stream S_m is:

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

where B_m is the worst-case blocking factor, which is equal to the longest duration of a lower priority message, τ_{bit} is the duration of a bit transmission and $hp(m)$ is the set of message streams with higher-priority than S_m. $Ina(I_m)$ integrates the temporary periods of inaccessibility caused by errors in frame transmission [3], including the time necessary to re-transmit failed messages. As a duplicate message is a consequence of the retransmission of an inconsistently failed message, the duration of its transmission is also included in the $Ina(I_m)$ term.

Some (or all) of these message streams may use the atomic multicast protocols presented in the previous Section. Therefore, they 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). Extra messages related to a message stream S_m are referred respectively has S_m^{dup}, S_m^{conf}, S_m^{ab} and $S_m^{retrans}$.

5.1 *IMD* protocol

The *IMD* protocol delay ($\delta_{deliver}$) is used to guarantee that a message is only delivered when it is known that there will be no more duplicates. A duplicate message appears when there is an error in the last but one bit of a frame and some nodes do not detect it. Thus, the sender will automatically retransmit the failed message. As the receiving node must evaluate such delay based in local information, it must take the arrival instant as its time reference. It must delay the delivery until the time it takes to completely retransmit a failed message. In the presence of a duplicate message (Figure 7), $\delta_{deliver}$ is reset.

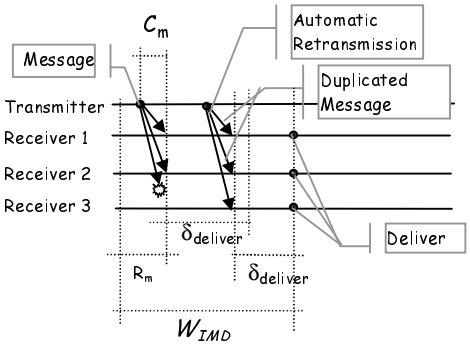

Figure 7. *IMD* protocol with one duplicate message.

Thus, $\delta_{deliver}$ must be greater 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. However, as the transmitter automatically tries to retransmit the failed frame, this retransmitted frame will not be blocked by any lower priority message:

$$\delta_{deliver} = R_m^{dup} \quad \wedge \quad B_m^{dup} = 0 \tag{4}$$

Considering the *IMD* protocol, the worst-case delivery time for message stream S_m is the sum of the message worst-case response time plus the delay introduced by each one of its duplicates:

$$W_m^{IMD} = R_m + (k_{dup} + 1) * \delta_{deliver} \tag{5}$$

The best-case delivery time is when a message is transmitted with its best-case response time (actual transmission time) and no duplicate is transmitted:

$$B_m^{IMD} = C_m + \delta_{deliver} \tag{6}$$

5.2 *2M* protocol

For the *2M* protocol, two different delays must be considered: $\delta_{confirm}$ and $\delta_{deliver}$. For $\delta_{confirm}$, it is considered that the message and the confirmation are both put in the transmission queue atomically, and that any delays needed to handle the transmission of the confirmation message by the sender node are inferior to the transmission time of the message. Thus, the evaluation of $\delta_{confirm}$ considers that the confirmation message will not suffer any blocking:

$$\delta_{confirm} = R_m^{conf} \quad \wedge \quad B_m^{conf} = 0 \tag{7}$$

Although network disturbances may lead to the duplication of confirmation messages, the $Ina(I_m)$ term of

equation (3) already integrates these duplicates in the evaluation of the response time.

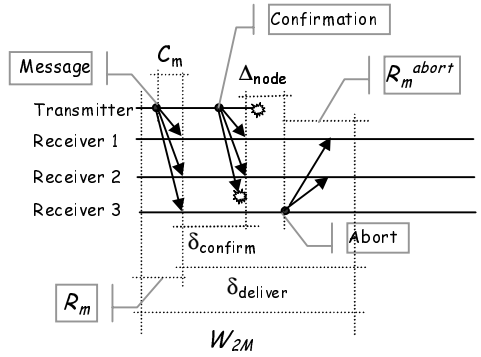

Figure 8. *2M* protocol with confirm omission.

The $\delta_{deliver}$ bound must be determined considering that every receiver must wait until it is known that it will not receive any abort message. These abort messages will be sent after $\delta_{confirm}$ by the nodes that do not receive the confirmation message (Figure 8). However, it must also be taken into account the response time of the node itself (Δ_{node}), between detecting a missed confirmation until it places the abort message in the outgoing queue:

$$\delta_{deliver} = \delta_{confirm} + \Delta_{node} + R_m^{abort} \quad (8)$$

Note that several abort messages may be transmitted in the network, in relation to the same omission error. However, to determine the $\delta_{deliver}$ bound it is only necessary to consider the first one to be transmitted, thus to consider the smaller Δ_{node} of all receiving nodes. The possible existence of several aborts in the network in case of error must be properly considered for the response-time evaluation of less priority messages.

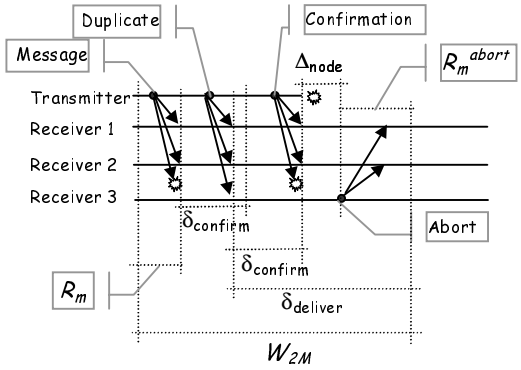

Figure 9. *2M* protocol with message duplicate followed by confirm omission.

The worst-case delivery time of message stream S_m, considering the *2M* protocol, is when a message is transmitted with its worst-case response time with possible duplicates (Figure 9), thus resetting both $\delta_{confirm}$ and $\delta_{deliver}$. Therefore, the worst-case delivery time must consider an extra $\delta_{confirm}$ for each assumed duplicate message.

$$W_m^{2M} = R_m + k_{dup} * \delta_{confirm} + \delta_{deliver} \quad (9)$$

The best-case delivery time is obtained when the message has its best-case response time (actual message transmission time) and there are no duplicates or omissions:

$$B_m^{2M} = C_m + \delta_{deliver} \quad (10)$$

5.3 *2M-GD* protocol

The *2M-GD* protocol has the same behaviour as 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 equation (8) with the worst-case response time of the retransmission 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.

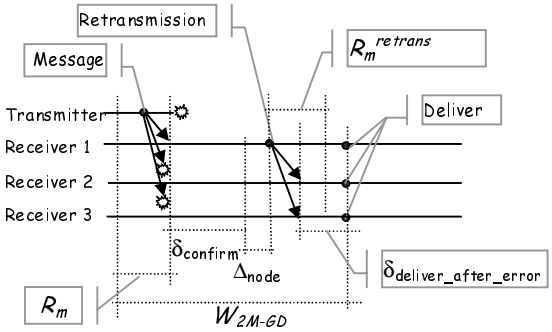

Figure 10. *2M-GD* protocol with message omission followed by transmitter failure.

An extra delay $\delta_{deliver\_after\_error}$ must be determined (Figure 10). This delay must be imposed within each node when a retransmission is received, since it can not guarantee that other nodes have already received it (due to inconsistent retransmission duplicates). Thus, it is necessary to follow a similar approach to the *IMD* protocol, and delay the delivery until all duplicates are correctly received. Hence, $\delta_{deliver\_after\_error}$ is equal to the worst-case response time of a duplicated retransmission message:

$$\delta_{deliver\_after\_error} = R_m^{retrans} \quad \wedge \quad B_m^{retrans} = 0 \quad (11)$$

The worst-case and best-case delivery time of the 2M-GD protocol are then given by considering that the worst-

case is when an inconsistent message omission occurs, and retransmissions are needed. Duplicate retransmission messages must be taken into account, since it must be guaranteed that every node delivers the message at the same time (once again, this response time is determined without blocking, since duplicates are immediately re-scheduled).

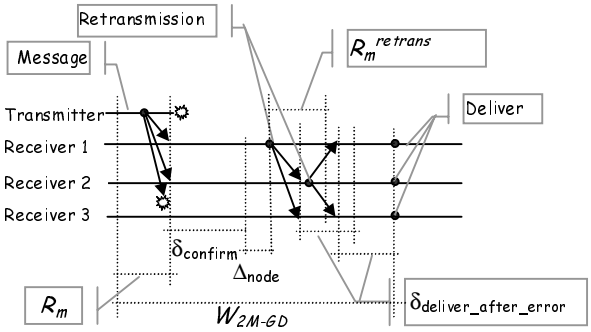

Figure 11. *2M-GD* Protocol with retransmissions.

However, the possible existence of multiple retransmissions must also be considered (Figure 11). A node receiving a second retransmission will consider it as a duplicate retransmission, as the messages are equivalent, and it will update $\delta_{deliver\_after\_error}$ accordingly. Therefore, the worst-case response time must also consider the maximum number of retransmissions it will receive (n_m^{rec} is the number of receivers of message stream S_m):

$$W_m^{2M-GD} = R_m + k_{dup} * \delta_{confirm} + \delta_{deliver} + \\ + (n_m^{rec} + k_{dup}) * \delta_{deliver\_after\_error} \quad (12)$$

$$B_m^{2M-GD} = C_m + \delta_{deliver} \quad (13)$$

5.4 *Consolidate* protocol

The *Consolidate* protocol has to delay the *decide* phase for a specific time (δ_{decide}), until it knows that it will not receive any more messages. This time is dependent on the worst-case delivery for each of the replicated messages. However, this worst-case time must be referred to an initial time reference, common to all nodes sending these messages. This common time reference must be the release time of the task that sends the message. Therefore, the worst-case delivery time of the messages must also take into account the worst-case response time of the tasks that send the message.

Knowing the worst-case delivery time for each of the replicated messages, δ_{decide} may be determined assuming that the first message to arrive has been transmitted with its best-case delivery time, and that the last message to arrive will be the one with the greater worst-case delivery time. Therefore:

$$\delta_{decide} = \max_{\forall i \in replicas} \{W_i\} - \min_{\forall i \in replicas} \{B_i\} + \varepsilon \quad (14)$$

where W_i is the worst-case delivery time of message *i* and B_i is the best-case delivery time of message *i*, considering a common time reference. ε is the maximum deviation between nodes' synchronised clocks.

5.5 Response time of message streams

In order to determine the response time of each message stream in the network, it is also necessary to consider the interference of confirmation messages and possible aborts or retransmissions of higher priority message streams that use the *2M* or *2M-GD* protocols. Equation (3) must be updated to account for these new periods of interference:

$$I_m = B_m + \sum_{\forall j \in hp(m)} \left(\left\lceil \frac{I_m + \tau_{bit}}{T_j} \right\rceil \times (C_j + C_j^{extra}) \right) + \\ + Ina(I_m) + \max_{\forall j \in hp(m)} \{extra\_msg_j\} \quad (15)$$

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

$$C_j^{extra} = \begin{cases} C_j^{conf} & 2M \text{ or } 2M\text{-}GD \text{ protocol} \\ 0 & \text{otherwise} \end{cases} \quad (16)$$

Additionally, $max\{extra\_msg_j\}$ accounts for the aborts or retransmissions in the network, due to inconsistent message omissions. As it is assumed the existence of a single inconsistent message omission during a period *T* (greater than the largest worst-case delivery time), each message stream needs only to consider the effect of one abort/retransmission due to inconsistent message omission per receiver of message *j*, that is:

$$extra\_msg_j = \begin{cases} n_j^{rec} * C_j^{abort} & 2M \text{ protocol} \\ n_j^{rec} * C_j^{retrans} & 2M\text{-}GD \text{ protocol} \\ 0 & \text{otherwise} \end{cases} \quad (17)$$

The $Ina(I_m)$ term in equation (15) integrates the periods of network inaccessibility caused by errors in frame transmission, therefore it includes the retransmissions of inconsistently failed messages (that is, duplicates).

5.6 Network utilisation

The network utilisation is given by the sum of the ratio transmission delay versus period of all message streams. Additionally, periods of temporary network inaccessibility

(due to on-going error detection and recovery mechanisms) must also be considered [3]:

$$U = \left(\sum_{\forall m} \frac{C_m}{T_m} \right) + U_{ina} \quad (18)$$

The U_{ina} term accounts for the network utilisation due to errors in frame transmission, therefore, as already referred, it includes the network utilisation related to duplicate messages.

Considering the proposed atomic multicast protocols, equation (18) must be updated to account for the extra messages in the network. For each message stream transmitted with the *2M* or *2M-GD* protocol, an extra confirmation message must be added (C_m^{extra}) to the first term of equation (18). Furthermore, a third factor is included in equation (19) to account for network utilisation related to inconsistent message omissions (one for each period *T*).

$$U = \left(\sum_{\forall m} \frac{C_m + C_m^{extra}}{T_m} \right) + U_{ina} + \frac{\max_{\forall m}\{extra\_msg_m\}}{T} \quad (19)$$

6. Numerical example

In order to clarify the use of the presented model, a simple example is used. In this example (Figure 12), a system where a distributed hard real-time application executes is considered. The system is constituted by four nodes, connected by a CAN network at a rate of 1 Mbit/sec.

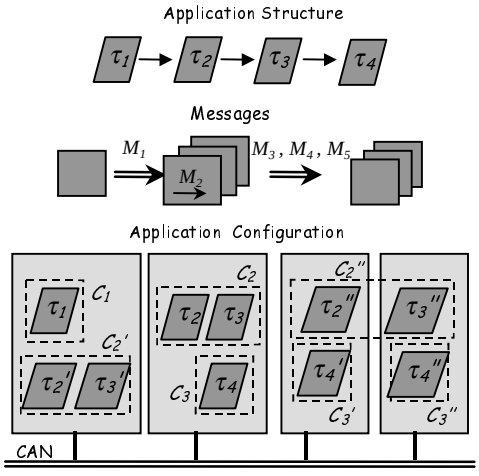

Figure 12. Application example.

The application is constituted by four tasks ($\tau_1..\tau_4$), 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 in three components: component C_1 encompasses tasks τ_1, component C_2 encompasses τ_2 and τ_3, and finally component C_3 is just τ_4. Component C_2 and C_3 are replicated in three replicas, while component C_1 is not replicated. Node 1 encompasses component C_1 (τ_1) and C_2' (τ_2', τ_3'), node 2 component C_2 (τ_2, τ_3) and C_3 (τ_4), node 3 component C_3' (τ_4') and C_2'' (but just τ_2'') and node 4 component C_3''(τ_4'') and C_2'' (but just τ_3'').

Table 1. Tasks' characteristics.

Task	Type	WCET	Period	Comp.	Nodes
τ_1	Per.	2	5	C1	1
τ_2	Per.	2	10	C2	1,2,3
τ_3	Spo.	3	10	C2	1,2,4
τ_4	Per.	4	15	C3	2,3,4

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

Table 2. Messages streams' characteristics.

Msg	Bytes	Period	From	To	Prot.
M_1	4	5	τ_1	τ_2,τ_2',τ_2''	2M-GD
M_2	8	10	τ_2''	τ_3''	IMD
M_3	6	10	τ_3	τ_4,τ_4',τ_4''	2M
M_4	6	10	τ_3'	τ_4,τ_4',τ_4''	2M
M_5	6	10	τ_3''	τ_4,τ_4',τ_4''	2M

Note that messages from τ_2 to τ_3 and τ_2' to τ_3' are internal to the node, since they are intra-component, and both tasks are in the same node. Since message M_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 M_1, but without data bytes. Since it is considered that an inconsistent message omission may occur, then it is also necessary to account for the possible 3 retransmission messages (one from each receiving node).

Message M_2 is internal to a component (although the component is 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 to concern. Messages M_3 to M_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 to *2M* protocol for the transmission of messages. Therefore there will be an extra confirmation message for each message sent (and possible abort messages).

In this analysis, the model of [3] is used, with the following error assumptions:

- a maximum of 2 errors 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;
- possible existence of an inconsistent message omission during the period of analysis;
- possible existence of 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.

The target of this example is to analyse the responsiveness of the proposed protocols, for both the response time and the delivery time of messages. *Response time* is considered as the time interval between requesting a message transfer until the message is fully received at the receiver side. *Delivery time* is considered as the time interval between requesting a message transfer until the Communication Manager delivers the message to the upper layers. If multicast protocols are not used, these times are equivalent, as it can be assumed that messages are delivered when they are correctly received.

Table 3. Messages' response time without protocols.

Msg	P	C_m	R_m^{NP}
M_1	5	0.089	0.519
M_2	10	0.127	0.630
M_3	10	0.108	0.741
M_4	10	0.108	0.852
M_5	10	0.108	0.852
U		6.590 %	

Table 3 presents the response time for each message stream and the network load when multicast protocols are not used, that is, the *Unreliable* protocol is used instead of *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. U is the network utilisation.

As it can be seen, the worst-case response time of messages is considerably greater than its actual transmission time. Although interference from higher priority messages is one of the factors leading to such difference, the main factor is the network bit error rate. For instance, a message of stream M_1 in an error free environment would have a worst-case response time of 0.219 ms. The possible existence of errors in the network more than duplicates its worst-case response time, even when multicast protocols are not used.

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. W_m and B_m are, respectively, the worst- and best-case delivery time for message stream M_m.

Table 4. Protocol-related delays.

Msg	Prot.	$\delta_{confirm}$	$\delta_{deliver}$	$\delta_{del\_aft\_er}$
M_1	*2M-GD*	0.350	0.969	0.389
M_2	*IMD*	-	0.848	-
M_3	*2M*	0.901	2.013	-
M_4	*2M*	1.065	2.341	-
M_5	*2M*	1.229	2.558	-

Table 5. Messages' delivery time considering protocols.

Msg	R_m^{MP}	W_m	B_m	W_m/R_m^{MP}
M_1	0.519	3.394	1.058	6.54
M_2	0.959	2.655	0.975	2.77
M_3	1.070	3.984	2.121	3.72
M_4	1.234	4.640	2.449	3.76
M_5	1.287	5.074	2.666	3.94
U		9.09 %		

As it 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 introduced delays, it is assumed that each message may be disturbed by one duplicate. For instance, the worst-case delivery time for message stream M_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 the possible existence of 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 M_2), while the *2M-GD* protocol is the one with the higher delays (Message M_1). Therefore, the system's engineer can use this reasoning to better balance reliability and efficiency in the system. Moreover, the multicast protocols increase network utilisation less than 50%, 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 approaches, and it is the strictly necessary to cope with inconsistent message omission using a software-based approach.

Since messages from replicated task τ_3 to replicated task τ_4 need to be consolidated, it is 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 time of each one of the messages (M_3 to M_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.

Table 6. Consolidation.

Task	WCRT	BCRT	Msg	W	B
τ_3	5	5	M_3	8.984	7.121
τ_3'	9	7	M_4	13.64	9.449
τ_3''	7.655	5.975	M_5	12.729	8.641

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 M_3 to M_5 (all referring to the common release time of task τ_2). Therefore, using equation (14):

$$\delta_{decide} = 13.64 - 7.121 + 0.1 = 6.619 \text{ ms} \qquad (20)$$

The worst-case response time of the consolidation (from the time that the first message is scheduled for transmission) is determined assuming that messages are delivered at their worst-case delivery time, but there is a message that is lost. In this case (assuming that the lost message is the one with the lower worst-case response time (M_3)), message M_4 will arrive with its worst-case response time of 4.640, which summed to the δ_{decide} gives a worst-case consolidation time of:

$$W_{Consolidate} = R_{M4} + \delta_{decide} = 11.259 \text{ ms} \qquad (21)$$

Although, this type of consolidation introduces delays in the system, its advantage is that no extra overhead is introduced in the network, preserving at the same time the predictability of the system. One of the main targets of the proposed multicast protocols is to introduce reliability in CAN communication, while at the same time preserving CAN real-time characteristics (thus allowing the offline analysis of messages' response times). Such target is achieved with the proposed multicast protocols, since the predictability of message transfers is guaranteed.

7. Conclusions

In spite of its built-in error detection/signalling mechanisms, CAN networks may cause inconsistencies in the supported applications, as messages can be delivered in duplicate by some receivers or delivered only by a subset of the receivers. In order to preclude such incorrect behaviour, a set of atomic multicast protocols is proposed. Total order is guaranteed through the transmission of just an extra message (without data) for each message that must tolerate inconsistent message omissions. Only in case of an inconsistent message omission (low probability) there will be more protocol-related retransmissions. Consolidation of replicated inputs is considered through the use of a consolidate protocol, built on top of the multicast protocols.

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

Acknowledgements

The authors would like to thank the anonymous referees for their helpful comments. This work was partially supported by FCT (project DEAR-COTS 14187/98) and IDMEC.

References

[1] ISO 11898. (1993). Road Vehicle - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication. ISO.

[2] Tindell, K., Burns, A. and Wellings, A. (1995). Calculating Controller Area Network (CAN) Message Response Time. In *Control Engineering Practice*, 3(8), pp. 1163-1169.

[3] Pinho, L., Vasques, F. and Tovar, E. (2000). Integrating inaccessibility in response time analysis of CAN networks. In *Proceedings of the 3rd IEEE International Workshop on Factory Communication Systems*, pp. 77–84, Porto, Portugal, September 2000.

[4] Rufino, J., Veríssimo, P., Arroz, G., Almeida, C. and Rodrigues, L. (1998). Fault-Tolerant Broadcasts in CAN. In *Proc. of the 28th Symposium on Fault-Tolerant Computing*, Munich, Germany, June 1998.

[5] Veríssimo, P., Casimiro, A., Pinho, L. M., Vasques, F., Rodrigues, L. and Tovar, E. (2000). Distributed Computer-Controlled Systems: the DEAR-COTS approach. In *Proc. of the 16th IFAC Workshop on Distributed Computer Control Systems*, Sydney, Australia, November 2000.

[6] Kaiser, J. and Livani, M. (1999). Achieving Fault-Tolerant Ordered Broadcasts in CAN. In *Proc. of the 3rd European Dependable Computing Conference*, Prague, , Czech Republic, September 1999, pp. 351-363

[7] Hadzilacos, V. and Toueg, S. (1993). Fault-Tolerant Broadcasts and Related Problems. In Mullender, S. (Ed.), *Distributed Systems*, 2nd Ed., Addison-Wesley, 1993.

[8] Cristian, F., Aghili, H., Strong, R. and Dolev, D. Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement. In *Information and Control*, 118:1, 1995.

[9] Poledna, S., Burns, A., Wellings, A. and Barret, P. (2000). Replica Determinism and Flexible Scheduling in Hard Real-Time Dependable Systems. IEEE Transactions on Computers, 49(2), pp. 100-111, 2000.

[10] Rufino, J. and Veríssimo, P. (1995). A Study on the Inaccessibility Characteristics of the Controller Area Network. In *Proc. of the 2nd International CAN Conference*, London, United Kingdom, October 1995.