Atomic Multicast Protocols for Reliable CAN Communication

Luís Miguel Pinho
Department of Computer Engineering,
ISEP, Polytechnic Institute of Porto
Rua São Tomé, 4200-072 Porto, Portugal
E-mail: lpinho@dei.isep.ipp.pt

Francisco Vasques
Department of Mechanical Engineering
FEUP, University of Porto
R. Dr. Bernardino de Almeida, 4200-465 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. This misbehaviour may be disastrous if the CAN network is used to support replicated 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.

Keywords: Distributed Systems: Algorithms; Real-Time; Fault-Tolerance.

Resumo

É geralmente considerado que as propriedades de difusão atómica em redes CAN (Controller Area Network) são garantidas através dos mecanismos existentes de detecção e sinalização de erros. No entanto, existem situações para as quais se verifica que uma mensagem pode ser entregue em duplicado em alguns dos seus consumidores ou ser entregue unicamente num subconjunto de consumidores. Estas irregularidades do protocolo CAN podem ter consequências catastróficas, caso este seja utilizado para suportar aplicações replicadas.

Este artigo propõe um conjunto de protocolos para serem utilizados em redes CAN, que garantem as propriedades de difusão atómica de mensagens. A análise temporal dos protocolos propostos evidencia o seu correcto funcionamento temporal, nomeadamente em termos de garantia de tempo de resposta limitado superiormente.

Palavras Chave: Sistemas Distribuídos: Algoritmos; Tempo-Real e Tolerância a Falhas;

1. Introduction

Controller Area Network (CAN) [1] is a fieldbus network suitable for small-scale Distributed Computer Controlled Systems (DCCS), being appropriate for transferring short real-time messages. The CAN protocol implements a priority-based bus, with a carrier sense multiple access with collision avoidance (CSMA/CA) MAC. In this protocol, any node can access the bus when it becomes idle. However, contrarily to Ethernet-like networks, the collision resolution is non-destructive, in the sense that one of the messages being transmitted will succeed.

This priority-based medium access control enables the use of CAN as the communication support for real-time distributed systems. 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/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 (*inconsistent message duplicate*), or, if the sender fails before re-transmitting the message, to the message being delivered only by a subset of the receivers (*inconsistent message omission*), leading to inconsistencies in the supported applications.

This misbehaviour 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 also in the order by which broadcasts are delivered, and to consolidate values from replicated inputs. Thus, it is necessary to provide protocols that guarantee these properties in spite of CAN inconsistencies, while at the same time preserving CAN real-time characteristics (thus allowing the offline analysis of the messages' response time).

In this paper, the timing characteristics of the provided reliable protocols are analysed, demonstrating that the real-time characteristics of CAN are preserved. The paper is organised as follows. The following Section presents some work related to reliable communication in CAN. Section 3 presents the protocols used to provide reliable real-time communication in CAN. The timing analysis of these protocols is then described in Section 4, developing the necessary pre-run-time schedulability conditions for the timing analysis of reliable real-time communication in CAN networks. Finally, a numerical example is presented in Section 5 and some conclusions are outlined in Section 6.

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.

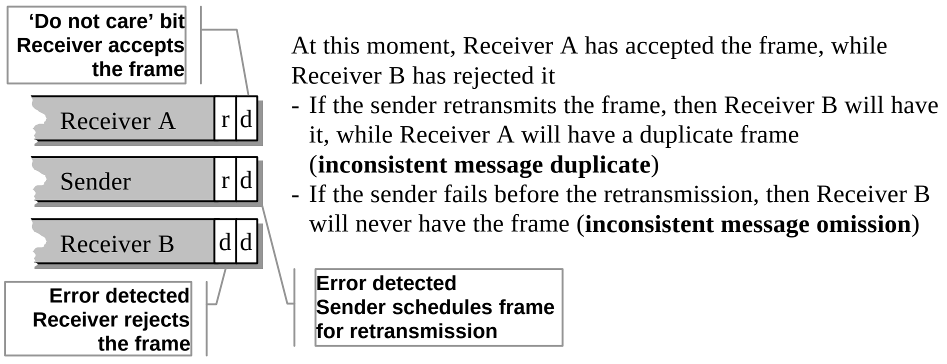
'Do not care' bit
Receiver accepts the frame
Receiver A
r
d
Sender
r
d
Receiver B
d
d
Error detected
Receiver rejects the frame
At this moment, Receiver A has accepted the frame, while Receiver B has rejected it
- If the sender retransmits the frame, then Receiver B will have it, while Receiver A will have a duplicate frame (inconsistent message duplicate)
- If the sender fails before the retransmission, then Receiver B will never have the frame (inconsistent message omission)
Error detected
Sender schedules frame for retransmission

Figure 1. Inconsistency in CAN.

In Figure 1, 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 receivers the last bit of a frame is a ‘do not care’ bit, Receiver A will not detect the error and will accept the frame. However, the transmitter re-schedules the frame, as there was an error. As a consequence, 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 of atomic multicasts. 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 situation causes an *inconsistent message omission*. This is a more difficult problem to solve, than in the case of inconsistent message duplicates.

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 [5]. 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. Reliable CAN Communication

Relying on CAN frames being simultaneously received in every node, the proposed protocols are based in delaying the deliver of a received frame for a specific (bounded) time. The approach is similar to the Δ-protocols [6], where, in order to obtain order, message delivery is delayed for a specific time (Δ). In the proposed 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 by an appropriated algorithm [7], to guarantee both deterministic execution of replicated components [8] and the correct evaluation of the delivery delays.

3.1 Failure Assumptions

In the proposed 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 in the network, at most one single inconsistent message omission occurs. 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 appropriate network redundancy schemes.

3.2 Atomic Multicasts

The proposed architecture provides several atomic multicast protocols with different failure assumptions and different behaviours in the case of errors. The *IMD* protocol provides an atomic multicast that just addresses the inconsistent message duplicate problem. The *2M* protocol provides an atomic multicast addressing both inconsistent message duplicates and inconsistent message omissions, where messages are not delivered at any node in an error situation.

The *IMD* protocol (Figures 2 and 3) provides an atomic multicast that just addresses the inconsistent message duplicate problem. In order 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 $d_{deliver}$). 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).

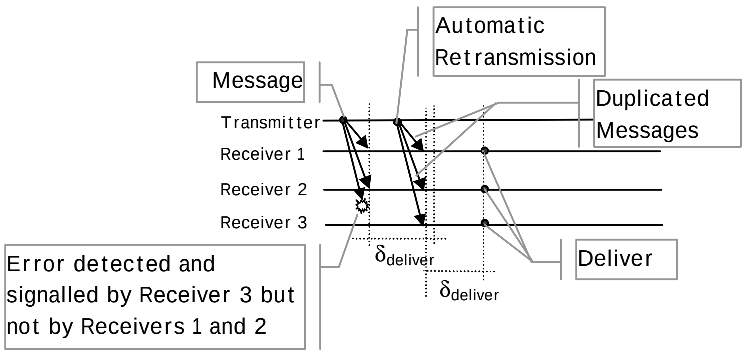

Fig. 2. Inconsistent message duplicate (*IMD* protocol).

```
Transmitter

1:  atomic_multicast (id, data):
2:     send (id, data)

3:  when sent_confirmed (id, data): -- if it is registered for this msg
4:     receivedMsgSet := receivedMsgSet ∪ msg(id,data)
5:     t_deliver(id) := clock + d_deliver(id)

6:  deliver:
7:     for all id in receivedMsgSet loop
8:        if t_deliver(id) < clock then
9:           state(id) := delivered
10:       end if
11:    end loop


Receiver

1:  when receive (id, data):
2:     if id ∉ receivedMsgSet then
3:        receivedMsgSet := receivedMsgSet ∪ msg(id,data)
4:        state(id) := unstable
5:     end if
6:     t_deliver(id) := clock + d_deliver(id)

7:  deliver:
8:     for all id in receivedMsgSet loop
9:        if state(id) = unstable and t_deliver(id) < clock then
10:          state(id) := delivered
11:       end if
12:    end loop
```

Fig. 3. ***IMD*** **Protocol Specification**

The *2M* protocol (Figures 4 and 5) addresses both the inconsistent message duplicates and inconsistent message omissions where for the case of inconsistent message omissions, it guarantees that either all or none of the receivers deliver the message. For the latter, not delivering a message is equivalent to a transmitting node crash before sending the message.

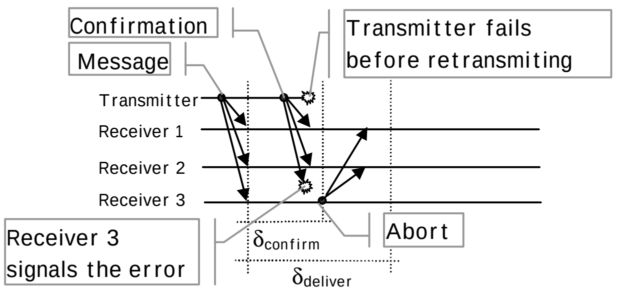

Fig. 4. Inconsistent message omission while sending the confirmation (*2M* protocol).

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 $t_{confirm}$, it multicasts the corresponding abort frame. 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 $t_{deliver}$ (as a node receiving the message but not receiving the confirmation does not know if the transmitter has failed while sending the message or while sending the confirmation).

```
Transmitter

atomic_multicast (id, data):
   send (id, message, data)
   send (id, confirmation)

when sent_confirmed (id, message, data): -- if interested in the msg
   receivedMsgSet := receivedMsgSet È msg(id,data)
   state(id) := confirmed
   $t_{deliver}(id)$ := clock + $d_{deliver}(id)$

deliver:
   for all id in receivedMsgSet loop
      if state(id) = confirmed and $t_{deliver}(id)$ < clock then
         state(id) := delivered
      end if
   end loop


Receiver

when receive (id, type, data):
   if type = message then
      if id $\notin$ receivedMsgSet then
         receivedMsgSet := receivedMsgSet $\cup$ msg(id,data)
         state(id) := unstable
      end if
      $t_{deliver}(id)$ := clock + $d_{deliver}(id)$ -- duplicate update
      $t_{confirm}(id)$ := clock + $d_{confirm}(id)$
   elsif type = confirmation then
      state(id) := confirmed
   elsif type = abort then
      if id $\in$ receivedMsgSet then
         receivedMsgSet := receivedMsgSet – msg(id)
      end if
   end if

deliver:
   for all id in receivedMsgSet loop
      if state(id) = confirmed and $t_{deliver}(id)$ < clock then
         state(id) := delivered
      elsif state(id) = unstable and $t_{confirm}(id)$ < clock then
         send (id, abort)
         receivedMsgSet := receivedMsgSet – msg(id)
      end if
   end loop
```

Fig. 5. *2M* Protocol Specification

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.

4. Response Time Analysis of Reliable Communication

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:

$$S_i = (C_i, T_i, D_i) \tag{1}$$

where S_i defines a message stream *i* characterised by a unique identifier. C_i is the longest message duration of stream S_i. T_i is the periodicity of stream S_i requests. In order to have a timing analysis independent from the model of the tasks at the application process level, it is assumed that this periodicity is the minimum time interval between two consecutive arrivals of S_i requests to the outgoing queue. Finally, D_i is the relative deadline of a message; that is, the maximum admissible 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 previously addressed in [2], considering fixed priorities for message streams (since the network access is based on the identifier's priority) and a non-preemptive scheduling model (since lower priority messages being transmitted cannot be preempted by pending higher priority messages). Considering such scheduling model, the existing schedulability analysis [9] is adapted to the case of scheduling messages on a CAN network. 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. The term 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:

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

where B_m is the worst-case blocking factor, which is equal to the longest duration of a lower priority message, t_{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 network inaccessibility caused by errors in frame transmission [3], including the time necessary to retransmit failed messages. As a duplicate message is a consequence of the retransmission of a 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_i are referred respectively has S_i^{dup}, S_i^{conf}, S_i^{ab} and $S_i^{retrans}$.

4.1 *IMD* Protocol

The *IMD* protocol delay ($d_{deliver}$) is used to guarantee that a message is only delivered when it is known that there will be no more duplicates duplicates. A duplicate message appears when there is an error in the last but one bit of a frame and some of the nodes do not detect it. Thus, in this case, 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. Thus, it must delay the delivery until the time it takes to completely retransmit a failed message (Fig. 6).

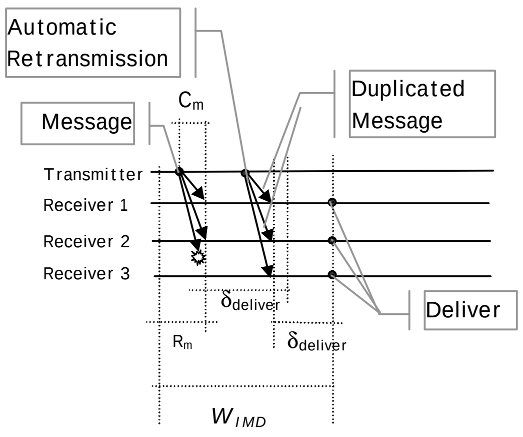

Fig. 6. *IMD* Protocol with one duplicate message.

Thus, $d_{deliver}$ 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:

$$d_{deliver} = R_m^{dup}, B_m^{dup} = 0 \qquad (4)$$

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 ($d_{deliver}$ is reset when a new duplicate arrives):

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

Considering the *IMD* protocol, the best-case delivery time of message stream S_m is when a message is transmitted with its best-case response time (actual transmission time) and no duplicate is transmitted:

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

4.2 *2M* Protocol

For the *2M* protocol, two different delays must be considered: $d_{confirm}$ and $d_{deliver}$. For the former, 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.

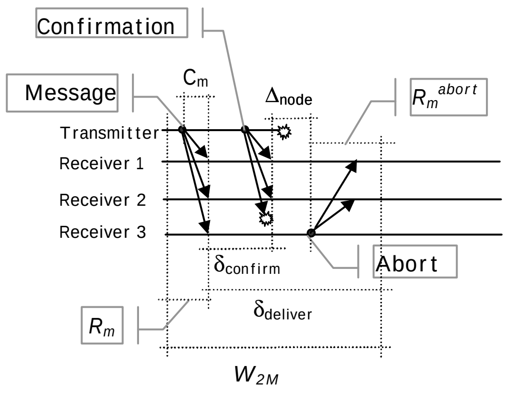

Fig. 7. *2M* Protocol with confirm omission.

Thus, the evaluation of $d_{confirm}$ considers that the confirmation message will not suffer any blocking:

$$d_{confirm} = R_m^{conf}, B_m^{conf} = 0 \tag{7}$$

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.

The $d_{deliver}$ 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 $d_{confirm}$ by the nodes that does not receive the confirmation message. However, it must also be taken into account the response time of the node itself (D_{node}), between detecting a missed confirmation until it places the abort message in the outgoing queue:

$$d_{deliver} = d_{confirm} + \Delta_{node} + R_m^{abort} \tag{8}$$

Note that several abort messages may be transmitted in the network, in relation to the same omission error. However, to determine the $d_{deliver}$ 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.

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 (thus resetting both $d_{confirm}$ and $d_{deliver}$). Therefore, the worst-case delivery time must consider an extra $d_{confirm}$ for each assumed duplicate message. 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:

$$W_m^{2M} = R_m + K_{dup} * d_{confirm} + d_{deliver} \tag{9}$$

$$B_m^{2M} = C_m + d_{deliver} \tag{10}$$

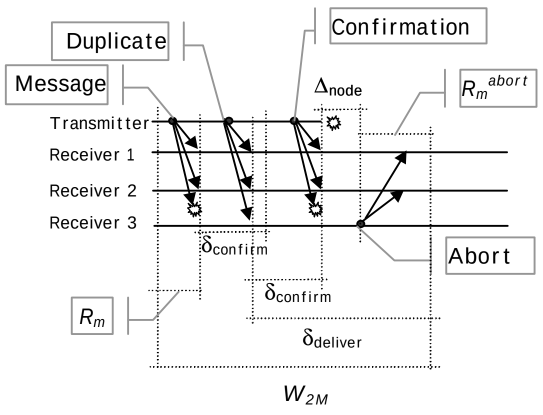

Fig. 8. ***2M*** **Protocol with message duplicate followed by confirm omission.**

4.3 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* protocol. Equation 3 must be updated to account for these new interferences.

$$I_m = B_m + \sum_{\forall j \in hp(m)} \left(\left\lceil \frac{I_m + t_{bit}}{T_j} \right\rceil \times (C_j + C_j^{extra}) \right) + Ina(I_m) + \max_{\forall j \in hp(m)} \{extra\_msg_j\} \tag{11}$$

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

$$c_j^{extra} = \begin{cases} C_j^{conf} & j \text{ is transmitted with the } 2M \text{ protocol} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Additionally, *max{extra_msg$_j$ }* accounts for the possible 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:

$$extra\_msg_j = \begin{cases} n_j^{rec} * C_j^{abort} & j \text{ is transmitted with the } 2M \text{ protocol} \\ 0 & \text{otherwise} \end{cases} \qquad (13)$$

The *Ina(I$_m$)* term of Equation 11 integrates the temporary periods of network inaccessibility caused by errors in frame transmission, therefore it includes the retransmissions of inconsistently failed messages (that is, duplicates).

4.4 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]:

$$U = \left(\sum_{\forall m} \frac{C_m}{T_m} \right) + U_{ina} \qquad (14)$$

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 14 must be updated to account for the extra messages in the network. For each message stream transmitted with the *2M* protocol, an extra confirmation message must be added (C_m^{extra}, Equation 12) to the first term of Equation 14. Furthermore, a third factor is included in Equation 15 to account for network utilisation related to inconsistent message omissions (one for each period *T*).

$$U = \left(\sum_{\forall m} \frac{C_m + C_m^{extra}}{T_m} \right) + U_{ina} + \frac{\max_{\forall m}\{extra\_msg_m\}}{T} \qquad (15)$$

5. Numerical Example

In order to clarify the use of the presented model, a simple example constituted by four nodes, connected by a CAN network at a rate of 1 Mbit/sec, is considered (Fig. 9). A distributed hard real-time application, constituted by four tasks ($\tau_1..\tau_4$), is spread over the nodes. As component replication is also used, then some of these tasks are also replicated.

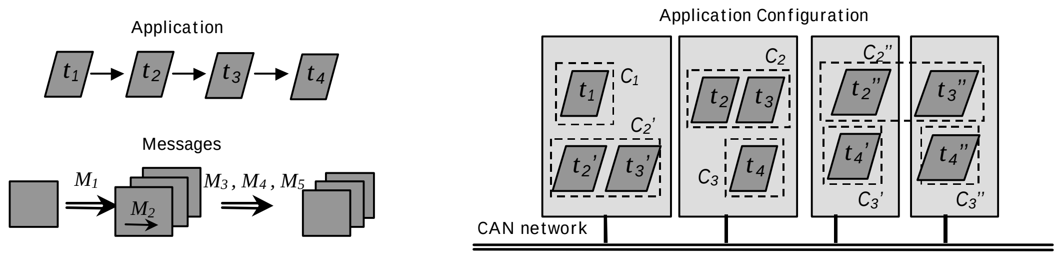

Fig. 9. Application Example

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

Table 1. Tasks' characteristics

Task	Type	WCET	Period	Component	Nodes
τ_1	Periodic	2	5	C1	1
τ_2	Periodic	2	10	C2	1,2,3
τ_3	Sporadic	3	10	C2	1,2,4
τ_4	Periodic	4	15	C3	2,3,4

Table 2. Messages Streams' Characteristics

Msg	Bytes	Period	From	To	Protocol
M_1	4	5	τ_1	τ_2,τ_2',τ_2''	2M
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* protocol is used. 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 the *2M* protocol is used for the transmission of messages. Thus 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 (e) 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 message is delivered to the upper layers. If multicast protocols are not used, response and delivery times are equivalent, since messages are delivered as soon as they are correctly received.

Table 3. Messages' Response Time without considering 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. 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 considered 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 is responsible for more than duplicating its worst-case response time, even when multicast protocols are not used.

Table 4 presents the messages' delivery times considering the use of 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. Messages' Delivery Time considering protocols

Msg	Protocol	R_m^{MP}	$\delta_{confirm}$	$\delta_{deliver}$	$\delta_{deliver\_ae}$	W_m	B_m	W_m/R_m^{MP}
M_1	*2M*	0.519	0.350	0.969	0.389	3.394	1.058	6.54
M_2	*IMD*	0.959	-	0.848	-	2.655	0.975	2.77
M_3	*2M*	1.070	0.901	2.013	-	3.984	2.121	3.72
M_4	*2M*	1.234	1.065	2.341	-	4.640	2.449	3.76
M_5	*2M*	1.287	1.229	2.558	-	5.074	2.666	3.94
U				9.09 %				

As it can be seen in Table 4, 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 $d_{deliver}$, but also by summing an extra $d_{confirm}$ due to the possible existence of a message duplicate.
The last column of Table 4, 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, while the *2M* protocol is the one with the higher delays (relative to its response time). 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.
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.

6. 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 has been 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.
These protocols explore the CAN synchronous properties to minimise its run-time overhead, and thus to 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 is also presented, demonstrating that the real-time capabilities of CAN are preserved, since predictability of message transfers is guaranteed.

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*, Vol. 3, No. 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*, pages 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] 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

[6] 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.

[7] Rodrigues, L., Guimarães, M. and Rufino, J. Fault-Tolerant Clock Synchronisation on CAN, In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[8] S. Poledna (1998). Deterministic Operation of Dissimilar Replicated Task Sets in Fault-Tolerant Distributed Real-Time Systems. In *Dependable Computing for Critical Applications 6*, pp. 103-119, IEEE Computer Society Press, 1998.

[9] A. N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. (1993). Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):285-292.

[10] J. Rufino and P. Veríssimo (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.