A TOTAL ORDERING SCHEME FOR REAL-TIME MULTICASTS IN CAN

Mohammad Ali Livani * **Jörg Kaiser** *

* *University of Ulm, Department of Computer Structures, 89069 Ulm, Germany*

Abstract: The Controller Area Network (CAN) is a broadcast medium providing advanced features, which make it suitable for many real-time applications. Common application layer protocols designed for CAN (e.g. CAL, SDS, DeviceNet) exploit these features in order to provide reliable real-time communication. However, they do not provide a consistent global message ordering in certain fault situations, nor do they consider temporal properties of the application.
This paper presents a multicast protocol which supports timely delivery of messages and guarantees atomic order, under anticipated fault conditions. In order to support causal ordering of events, the paper provides a discussion on establishing Lamport's precedence relation between events by appropriate usage of real-time multicasts.
In a CAN-based distributed system, the restricted communication bandwidth constitutes a serious bottle-neck. Therefore an important feature of this multicast protocol is to achieve optimal protocol termination time while requiring minimum communication overhead.

Keywords: Real-time communication; fault-tolerance; multicasting; total order; CAN

1. INTRODUCTION

Future computer systems will, to a large extent, monitor and control real-world processes. This results in an inevitable demand for timeliness and reliability. Distributed systems which inherently provide extensibility and immunity against single failures, are an adequate architecture to cope with spatially distributed real world applications. Moreover, the availability of inexpensive, powerful micro-controllers promotes distributed solutions.

Group communication is the basic mechanism to coordinate distributed activities. In order to control the competition and cooperation among distributed processes, and to allow for object replication, atomic delivery of multicast messages is inevitable. In this paper, the following definition of atomic multicast delivery is used.

Definition 1. Atomic multicast delivery – Let G be a multicast group, i.e. a set of objects which must receive a multicast message. Let $del_i(m)$ denote the delivery of a message m to an object i and '$\longrightarrow$' define the precedence relation. Then two messages m and m' are delivered atomically to the group G, if and only if the following holds:

$$\begin{aligned} \forall i,j:\ & i,j \in G \ \wedge\ i,j \text{ non-faulty} \\ & \Rightarrow (\ del_i(m) \longrightarrow del_i(m') \\ & \quad \Leftrightarrow (del_j(m) \longrightarrow del_j(m')) \end{aligned}$$

In other words, atomic delivery of multicast messages implies two properties:

- *Consistent Delivery*: if a multicast message is received by a non-faulty group member, then it is received by all non-faulty group members.

- *Consistent Ordering*: messages received by different non-faulty group members, are received in the same order.

Achieving the consistent delivery requires either a two-phased commit protocol with a considerable acknowledgment overhead (Babaoglu and Drummond, 1985; Birman and Joseph, 1987; Chang and Maxemchuk, 1984), or a mechanism to retransmit each message so many times that the probability of losing all copies is negligible (Cristian, 1990; Livani, 1998; Rufino *et al.*, 1998).

Consensus on the ordering of the messages comes free with the two-phased commit protocol. But if the multiple transmission approach is applied, an additional mechanism for establishing a unique order of the messages must be used. (Schneider, 1990) has described an approach to achieve consensus on the message ordering among a group of receivers, using a time-dependent unique identifier of messages. An approach which exploits knowledge about the delivery deadlines of the messages, has been introduced by (Cristian *et al.*, 1985). A similar mechanism has been proposed for embedded applications by (Zuberi and Shin, 1996). These approaches, however, rely on the timely and reliable delivery of all messages.

The ordering scheme presented in this paper considers also the late transmission of soft real-time messages, which is a timing failure of the message transmission, but does not lead to a system failure. As already shown in (Livani *et al.*, 1998) it is possible to schedule the real-time communication in a CAN-bus system so that hard real-time messages are always transmitted timely while soft transmission deadlines are missed in overload situations. In such a system it is not possible to guarantee a total delivery order among all hard and soft real-time messages.

To see the reason of this restriction consider a hard real-time message H, a soft real-time message S, and two nodes N_1 and N_2. Since no latest delivery time can be guaranteed for S, its consistent delivery must depend on the reception of a signal σ from the bus (e.g. S itself, a commit packet, or another packet depending on the protocol). Assume that N_1 receives σ at time t_1, and N_2 does not because of a single communication error. If N_1 and N_2 receive H after t_1 and N_2 does not receive σ until d_H (i.e. the deadline of H) due to bus overload, then at d_H, N_1 must deliver S before H, and N_2 must deliver H before S.

Due to the restriction mentioned above, this paper proposes an ordering algorithm, which establishes a total order for the messages of each class (i.e. hard and soft real-time) separately.

The paper is organized as follows: section 2 introduces some properties of the CAN bus which are essential to understand the approach. Section 3 presents the deadline-based total ordering scheme. Section 4 discusses how to reflect Lamport's precedence relation using the deadline-based total order. A summary concludes the paper.

2. SOME PROPERTIES OF CAN

The CAN-bus (BOSCH, 1991) is a priority bus targeted to operate in a noisy environment with speeds of up to 1 Mbit/s, exchanging small real-time control messages. The priority-based arbitration mechanism of the CAN bus can be exploited to guarantee the timely transmission of hard real-time messages under anticipated load and fault conditions (Tindell and Burns, 1994). In order to provide best effort scheduling of soft real-time communication in a CAN bus while guaranteeing hard deadlines, different approaches may be applied (Davis, 1994; Livani *et al.*, 1998) which are based on multiple priority classes.

The CAN protocol provides efficient hardware-implemented error handling, which is based on various error detection mechanisms with a very high total coverage, and an approach to immediately signaling the error condition. These features of the CAN protocol ensure atomic broadcast delivery in most fault situations. However, if an error occurs within the transmission time of the two last bits of a data frame, some receivers might accept and other receivers reject the frame. Although the sender (or in case of immediate sender crash an alternative mechanism like Eager Diffusion (Rufino *et al.*, 1998) or Shadow Retransmitter (Livani, 1998)) will retransmit the frame, the consistent ordering of the incoming messages in different sites is not trivial due to the possible transmission of other (high-priority) messages between the first and the second transmission of a frame. Another problem caused by such errors is the duplicate frame reception by some receivers.

In such situations, commercially available application level protocols for CAN, like CAL (CiA, 1993), SDS (Crovella, 1994), and DeviceNet (Noonen *et al.*, 1994) manage to discard frame duplicates, but they fail to provide consistent message ordering among a group of receivers.

Relying on a feasible and flexible real-time scheduling policy (Davis, 1994; Livani *et al.*, 1998) combined with a reliable broadcast delivery mechanism (Livani, 1998; Rufino *et al.*, 1998), following assumptions can be made about the communication system in a CAN bus:

(A1) Hard real-time messages are transmitted timely (i.e. until their transmission deadline)

under anticipated fault scenarios, even in overload situations.

(A2) Soft real-time messages are scheduled by static or dynamic priorities, but their deadlines may be missed in overload situations.

(A3) If a non-faulty receiver receives a message, then eventually all non-faulty receivers receive the message.

Given these assumptions, atomic multicasting can be achieved by establishing a consistent global delivery order for multicast messages.

3. THE DEADLINE-BASED TOTAL ORDERING SCHEME

This section introduces a scheme to achieve globally consistent ordering of multicast messages in a CAN-based system. The algorithm presented here relies on the assumptions A1 through A3 in order to achieve atomic multicast with minimum communication overhead, based on the knowledge about the message transmission deadlines. The transmission deadline of a message denotes the time, at which the message must be successfully transmitted to all non-faulty destination sites. Since the transmission deadline is tightly related to the time, where the sender expects the message delivery to all receiving application objects, this ordering mechanism allows an application specific ordering, which is related to temporal requirements. Although this approach is similar to some other ones known from much previous work (Cristian *et al.*, 1985; Schneider, 1990), it considers the late transmission of soft real-time messages in overload situations. Another advantage of this scheme is that it needs no additional communication to establish a globally consistent ordering decision.

Let m and m' be two real-time messages with transmission deadlines d_m and $d_{m'}$. Then the deadline-based ordering algorithm implements the following rule:

(O1) $d_m < d_{m'} \Rightarrow del_j(m) \longrightarrow del_j(m')$

This means that if the deadline of the message m is before the deadline of the message m', then m must be delivered to destination objects before m'. If the underlying protocol layer adjusts the transmission deadlines of hard real-time messages to reserved time-slots (Livani *et al.*, 1998), then the delivery order of hard real-time messages can be always established by this rule.

However, when the deadlines of different messages are equal, the following rule must be used to ensure the globally consistent decision on delivery order of messages.

(O2) $d_m = d_{m'} \Rightarrow (del_j(m) \longrightarrow del_j(m') \Leftrightarrow header_m < header_{m'})$

Where $header_m$ consists of a tuple (d_m, $sender_m$, $subject_m$) and identifies a message uniquely. This rule requires that different messages sent by the same sender with the same subject have different transmission deadlines, otherwise an additional sequence number (or "toggle-bit") must be provided in the header. Note that this requirement is also necessary for distinguishing duplicate frames from successive frames.

3.1 *Protocol Termination and End-to-End Delivery*

An atomic multicast protocol may terminate and deliver a message to destination objects, if and only if a) the message is received and accepted by all non-faulty destination sites, and b) the message can be consistently ordered, which means that no other message will arrive later, which must precede the current message according to the ordering criteria.

a) Consistent Message Reception in CAN

In order to minimize the communication overhead, the nodes must not exchange explicit information about the status of their message queues. Thus they have to conclude this information from implicit knowledge. Since the underlying message transfer protocol guarantees the timely transmission of hard real-time messages under anticipated fault conditions, the protocol can assume that a hard real-time message is consistently transmitted to all receivers at its deadline. Hence a receiver can deliver the message to the destination objects at the transmission deadline.

For soft real-time messages, however, the transmission deadline may be missed because of bus overload, and hence, late transmission of the message is still possible. In these situations, the deadline cannot be used to specify the time when the message may be delivered. However, in such situations the following considerations lead to a decision criterion:

Claim 2. If a node has received a message m, and then another message with lower priority is observed on the CAN bus, or the bus is idle, then the sender of m will not retransmit it in future.

PROOF. Assume that a message m is transmitted at least once on the CAN bus. Further, assume that the sending CAN controller still attempts to retransmit m due to the inconsistent transmission. According to the CAN specification (BOSCH, 1991), the sender will try to retransmit m "immediately", thus no bus idle period will be

observed before the retransmission of m. Furthermore, no lower-priority message will be transmitted before the retransmission of m, because m will win the arbitration process against any lower-priority message. □

As a consequence, the receivers of a soft real-time message m can be sure that either the sender is crashed, or the transmission was successful, as soon as they detect either a bus idle period or a lower-priority message after receiving m.

If a frame is accepted by a subset of the receivers, and rejected by other receivers, the immediate sender crash may lead to inconsistent message reception. This failure must be tolerated by having some nodes retransmit the frame. The Eager Diffusion Protocol (Rufino *et al.*, 1998) retransmits every frame at least by one receiver. However, due to processing delay the Eager Diffusion Protocol cannot guarantee the immediate retransmission. The ordering scheme presented in this paper relies on dedicated Shadow Retransmitters (Livani, 1998), which guarantee immediate retransmission whenever an inconsistent message reception is possible. The Shadow retransmitters also transmit an ultra-low priority frame (called trailer) at the end of each non-broken sequence of CAN frames. So an idle bus can be detected by observing a trailer on the bus.

From the previous discussion, following properties are derived:

(P1) Any receiver of a hard real-time message m with deadline d_m, can assume that m will be received by all sites until d_m.

(P2) Any receiver of a soft real-time message m can assume that the message has been received by all sites, if it observes a frame with a lower priority on the bus after receiving m.

b) Achieving a Consistent Message Order in CAN

In case of hard real-time messages, the knowledge of time is sufficient for stabilizing the ordering decision: after a deadline d_m, no hard real-time message with an earlier deadline will arrive. But this statement is not true in case of soft real-time messages. Here, the following assumption constitutes the constraint necessary to find a stable ordering decision criterion:

(A4) Every real-time message m is ready to transmit before $d_m - \Delta T_{\max}$, with $\Delta T_{\max}$ being the maximum time required for a single transmission of an arbitrary frame.

The assumption (A4) concludes the following:

Claim 3. Assume a deadline-based message priority scheme e.g. (Livani *et al.*, 1998), where a soft real-time message with a higher priority has an earlier deadline than a soft real-time message with a lower priority. If after the transmission deadline d_m of a soft real-time message m another message with lower priority is transmitted on the CAN bus, or the bus is idle, then no other soft real-time message m' with a deadline $d_{m'} \leq d_m$, will be transmitted later.

PROOF. Assume that a message m' with deadline $d_{m'} \leq d_m$ is pending for transmission at the time $t > d_m$. Because of the deadline-based priority assignment the priority of m' is not lower than the priority of m. Due to (A4), m' has been pending for transmission at least since $d_{m'} - \Delta T_{\max}$, hence at least since $d_m - \Delta T_{\max}$. Hence the sender of m' must have been trying to transmit m' at the beginning of every bus-idle period since $d_m - \Delta T_{\max}$. Therefore no idle bus can be observed between d_m and the successful transmission of m'. Also, no lower-priority message can be transmitted on the bus before m', because m' wins the arbitration process against any lower-priority message. Thus, if after d_m a message with a lower priority than m is transmitted on the CAN bus, or the bus is idle, then no message m' with $d_{m'} \leq d_m$ will be transmitted later. □

From the previous discussion, following property is derived:

(P3) For any soft real-time message m, no preceding soft real-time message will arrive later, if after the transmission deadline d_m a lower-priority message is observed on the bus.

3.2 *Atomic Multicast Delivery Algorithm*

The proposed atomic multicast delivery algorithm is based on the following rules:

Order: Messages are ordered by their deadlines. In case of equal deadlines, the value of the message header is used for the order decision.

Time1: every received hard real-time message is delivered at its deadline. (This realizes the deadline-based order implicitly)

Time2: Every received soft real-time message must be delivered as soon as either another message with later deadline, or a bus idle time is observed after its transmission deadline.

Due to the rule (Time2), the delivery of a soft real-time message m may be delayed until $d_m + \Delta T_{\max}$. This must be considered when calculating the transmission deadline of any soft real-time message.

In the following, the atomic multicast delivery algorithm is presented. The algorithm assumes

the availability of a real-time clock, which is synchronized globally with a bounded inaccuracy.

```
Atomic_multicast_delivery
initialization:
  SRT_q ← empty_q;
  HRT_q ← empty_q;
  next_HRT_delivery ← eternity;
receive(m,t): /* m is sent or received at t */
  if m.dest_group ∈ my_groups
       and m.category = HRT then
    HRT_q.insert (m);
    if m.dl < next_HRT_delivery then
      next_HRT_delivery ← m.dl;
      set_wakeup (next_HRT_delivery);
  if m.dest_group ∈ my_groups
       and m.category = SRT then
    SRT_q.insert (m);
    if m.dl ≤ t then
    /* overload! Deliver preceding SRTM */
      while SRT_q.head.dl < m.dl
            or SRT_q.head < m do
        mx ← SRT_q.gethead;
        SRT_deliver (mx, mx.dest_group);
    else
    /* Deliver SRTM with passed DL */
      while SRT_q.head.dl < t do
        mx ← SRT_q.gethead;
        SRT_deliver (mx, mx.dest_group);
end receive;
wakeup(): /* invoked when the wake–up
                time is reached */
  mx ← HRT_q.gethead;
  HRT_deliver (mx, mx.dest_group);
  if HRT_q ≠ empty_q then
    next_HRT_delivery ← HRT_q.head.dl;
    set_wakeup (next_HRT_delivery);
  else
    next_HRT_delivery ← eternity;
end wakeup;
bus_idle(t): /* invoked if at time t a
                trailer frame is received */
  while SRT_q.head.dl < t do
  /* Deliver all SRTM with passed DL */
    mx ← SRT_q.gethead;
    SRT_deliver (mx, mx.dest_group);
end bus_idle;
end Atomic_multicast_delivery
```

4. GLOBAL ORDERING OF EVENTS

The deadline-based ordering of multicast messages enables a consistent global order between different events observed in the distributed system. In this paper an event is called a global event, if the object which observes it, sends an atomic multicast to enforce a global observation of the event by all concerning objects. As seen by application objects, the order of global events is the same as the globally consistent delivery order of their notification messages. This globally consistent delivery order of messages is established by the atomic multicast protocol described in section 3.2. Note that if a set of events has to be globally ordered, then all multicast messages propagating those events must belong to the same class (hard or soft real-time).

4.1 *Achieving Causal Ordering of Events*

If global events have to be ordered according to their causality, following cases can be observed:

A: Let e and e' be two global events locally observed by an object i, with a causal precedence relation $e \longrightarrow e'$, and let m and m' be messages, which propagate the events e and e' in the system. Let d_m denote the deadline of m. In order to globally agree on the order $e \longrightarrow e'$, for every object j, the relation $del_j(m) \longrightarrow del_j(m')$ is sufficient. Hence, due to the rule (O1), the relation $d_m < d_{m'}$ is sufficient. This means, that in order to reflect a precedence order $e \longrightarrow e'$ globally, i must assign deadlines $d_m < d_{m'}$ to the messages propagating e and e' in the system.

B: Let e and e' be two events observed by two different objects i and j, and $e \longrightarrow e'$, then (according to Lamport's definition of precedence relation (Lamport, 1978), and the discussion in section 3.1) following conditions hold:

(1) Event e is observed by i at local time $T^i(e)$.

(2) Event e' is observed by j at local time $T^j(e')$.

(3) There is a message m sent by i at the local time $T^i(send_i(m))$, which propagates the event e, where $T^i(e) \leq T^i(send_i(m)) < d_m$.

(4) There is a message m' sent by j at the local time $T^j(send_j(m'))$, which propagates the event e', where $T^j(e') \leq T^j(send_j(m')) < d_{m'}$.

(5) If j receives m, then it receives m at the local time $T^j(del_j(m))$, where $d_m \leq T^j(del_j(m))$, and $T^j(del_j(m)) \leq T^j(e')$. Thus due to (4) it follows that $d_m < d_{m'}$.

(6) If j does not receive m, then there is a set of messages $m_1, m_2, m_3, \cdots, m_n$, and a set of objects $k_1, k_2, k_3, \cdots, k_n$, where firstly, k_1 receives m and sends m_1, k_2 receives m_1 and sends $m_2, \cdots, k_n$ receives m_{n-1} and sends m_n, and j receives m_n. Secondly, $del_{k_1}(m) \longrightarrow send_{k_1}(m_1)$, $del_{k_2}(m_1) \longrightarrow send_{k_2}(m_2)$, $\cdots$, and $del_{k_n}(m_{n-1}) \longrightarrow send_{k_n}(m_n)$. And thirdly, $T^j(del_j(m_n)) \leq T^j(e')$. In this case, $d_m \leq T^{k_1}(del_{k_1}(m)) < T^{k_1}(send_{k_1}(m_1)) < d_{m_1} \leq T^{k_2}(del_{k_2}(m_1)) < T^{k_2}(send_{k_2}(m_2)) < d_{m_2} \cdots < d_{m_n} \leq T^j(del_j(m_n)) \leq T^j(e')$. Again, due to (4) it follows that $d_m < d_{m'}$.

In either case (i.e. A; B–5; B–6), every object l in the system which receives m and m', will establish the delivery order $del_l(m) \longrightarrow del_l(m')$ due to the relation $d_m < d_{m'}$. Consequently, l will establish the precedence order $e \longrightarrow e'$. Thus, if e and e' are two global events observed in the distributed system, and $e \longrightarrow e'$ according to Lamport's definition of precedence relation, then all non-faulty objects in the system – which are addressed by messages propagating e and e' – will be notified about e before e', and hence they will establish consistently the precedence order $e \longrightarrow e'$.

5. CONCLUSION

The paper introduced a real-time multicast ordering scheme, which achieves consistent delivery ordering of multicast messages using the message transmission deadlines.

The protocol termination time depends on the message class. The protocol relies on a medium access protocol, which guarantees timely and reliable transmission of hard real-time messages to all destination nodes (Livani, 1998; Livani *et al.*, 1998). Hence, for a hard real-time message the protocol terminates at the transmission deadline and delivers the message to destination objects timely. Under 'normal' load conditions, the protocol delivers soft real-time messages up to one frame transmission time later than their transmission deadlines (cf. rule Time2 in section 3.2). However, in overload situations, the protocol may delay the delivery of soft real-time messages until the end of the overload period plus one frame transmission time.

The mechanism ensures consistent multicast ordering in presence of communication failures leading to inconsistent global view of the CAN bus status among non-faulty sites. Common high-level communication protocols designed for CAN, do not provide consistent multicast delivery order among all application objects in these failure situations.

It was shown, that causal order between events can be globally established by using the deadline-based ordering approach.

An important benefit of this scheme is its efficiency: the algorithm requires no additional communication, like acknowledgements etc. In the Controller Area Network, communication bandwidth is a serious bottle-neck. While the computing power of hardware nodes is increasing rapidly, the bandwidth of CAN will remain limited to a maximum of 1 Mbits/sec. This was the main motivation to develop a multicast scheme which minimizes communication amount for the price of some computational overhead.

6. REFERENCES

Babaoglu, Ö. and R. Drummond (1985). Streets of byzantium: Network architectures for fast reliable broadcasts. *IEEE Tr. Software Eng.* **11**(6), 546–554.

Birman, K.P. and T.A. Joseph (1987). Reliable communication in the presence of failures. *ACM Tr. on Computer Systems.*

BOSCH (1991). CAN specification version 2.0. *Published by Robert BOSCH GmbH.*

Chang, J.M. and N.F. Maxemchuk (1984). Reliable broadcast protocols. *ACM Tr. on Computer Systems* **2**(3), 251–273.

CiA (1993). CAN application layer (CAL) for industrial applications. *CiA Draft Standards 201..207.*

Cristian, F. (1990). Synchronous atomic broadcast for redundant broadcast channels. *The Journal of Real-Time Systems* **2**, 195–212.

Cristian, F. et al. (1985). Atomic broadcast: From simple message diffusion to byzantine agreement. *15th Int. Symposium on Fault Tolerant Computing.*

Crovella, R.M. (1994). SDS: A CAN protocol for plant floor control. *1st Int. CAN Conference.*

Davis, R. (1994). Dual priority scheduling: A means of providing flexibility in hard real-time systems. *Technical Report YCS230, University of York.*

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of ACM.*

Livani, M.A. (1998). SHARE: A transparent mechanism for reliable broadcast delivery in CAN. *Informatik Bericht 98-14, University of Ulm.*

Livani, M.A., J. Kaiser and W.J. Jia (1998). Scheduling hard and soft real-time communication in the controller area network (CAN). *23rd IFAC/IFIP Workshop on Real Time Programming.*

Noonen, D., S. Siegel and P. Maloney (1994). Devicenet application protocol. *1st Int. CAN Conference.*

Rufino, J., P. Verissimo, G. Arroz, C. Almeida and L. Rodrigues (1998). Fault-tolerant broadcasts in CAN. *28th Int. Symposium on Fault Tolerant Computing.*

Schneider, F. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys.*

Tindell, K. and A. Burns (1994). Guaranteeing message latencies on controller area network (CAN). *1st Int. CAN Conference.*

Zuberi, K.M. and K.G. Shin (1996). A causal message ordering scheme for distributed embedded real-time systems. *Proc. Symp. on Reliable and Distributed Systems.*