SHARE: A Transparent Approach to Fault-tolerant Broadcast in CAN

Mohammad Ali Livani

The Controller Area Network (CAN) exhibits a highly reliable and predictable behavior, as it is required by embedded real-time control applications. Although CAN provides a consistent view of every transmitted frame among all peer CAN controllers in most fault scenarios, the frame-level consistency may be lost due to an error during the last two bits of a frame followed by an immediate sender crash, as indicated by Rufino et al [17].
Previous Approaches to solve this problem require modifications to the software-level communication protocol, which result in additional communication load. This would affect the performance parameters of the real-time application system.
In this paper, a hardware component (SHARE: Shadow Retransmitter) is suggested, which undertakes the retransmission of frames in such cases transparently, i.e. the actions of a SHARE are not visible to other CAN nodes, and do not influence the performance parameters of the existing application system.

Keywords: Real-time communication, fault-tolerance, reliable broadcast, CAN.

1 Introduction

Real-time control systems must provide timely and reliable computing service to the real-world environment. Distributed systems which inherently provide immunity against single failures, are an adequate architecture to achieve high availability of the computing system. Moreover, the availability of inexpensive, powerful micro controllers promotes distributed solutions. By replicating the processes on different sites the system is able to provide the service despite the failure of some processes or sites. The need for consistent replication of the process state information results in the demand for a communication service which provides reliable and consistent delivery of information to groups of processes. Such a communication service is termed atomic broadcast [5]. A real-time atomic broadcast protocol must terminate within a well-known worst-case message delivery delay. Some examples of real-time broadcast protocols are the Δ-protocol [5],[6] and the time-triggered protocol [9].
In the area of industrial automation and automotive applications, field busses are used to disseminate time critical messages. Field busses exhibit bounded message length, high reliability of data transfer, and efficiency of the protocol. Among field busses, the CAN bus [14] provides advanced features, which make it suitable for a wide range of real-time applications. Some of these features are high robustness against electro-magnetical interference, priority-based multiparty bus access control, variable but bounded message data length ($\leq$ 8 bytes), efficient implementation of acknowledgement and error indication, and automatic fail-silence enforcement.
There are application level protocols available for CAN which raise the network interface and shield the programmer from low level details. The most popular ones are CAL [4], CAN Kingdom [8], and Device Net [7]. These protocols rely on the consistent delivery of broadcast messages to all receivers by the CAN protocol. However, consistent delivery of broadcast messages is not provided by the CAN protocol, whenever receivers do not agree on the correctness of the message, and the sender crashes before a successful retransmission. This lack of consistency has been pointed out and analyzed by Rufino et al [17]. Although this situation is very rare, Rufino et al have shown that under assumption of a bit error rate of 10^{-5} and a node mean-time-

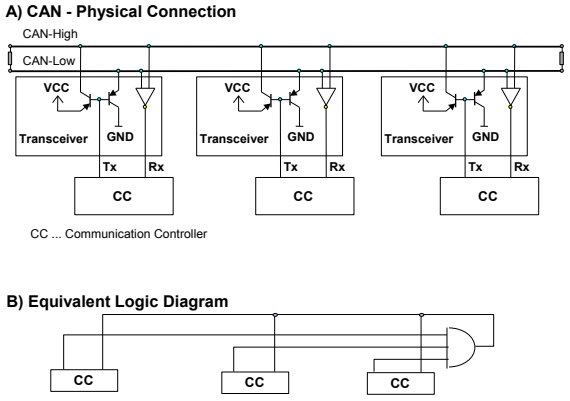

Fig. 1: CAN bus connection (wired-AND)

to-fail of 10^4 hours, the mean time to such an *inconsistent message omission* is 3.98 x 10^{-8}, which is low enough to justify its consideration in safety critical applications, where the system's mean-time-between-failures must be higher than 10^9 hours. They also have introduced a reliable broadcast protocol based on (sender-initiated) commit frames[1] and message retransmission by receivers upon timeout. This solution results in transmission of at least one additional frame for every broadcast message, and can lead to bandwidth shortage. It is not applicable to many existing systems, because of its impact on the bus schedulability. Considering the CAN bandwidth as a restricted resource, an optimal solution would detect this failure situation and retransmit the message only if necessary.

This paper is organized as following. Section 2 surveys the mechanisms of the CAN protocol to achieve consistent broadcast. It is shown that a consistent delivery of broadcast messages in presence of faults is not guaranteed by the CAN protocol itself. Section 3 reviews some approaches to achieve fault-tolerant broadcasts in CAN. Section 4 introduces a new approach to provide consistent delivery of broadcasts in CAN transparently by Shadow Retransmitters. A summary concludes the paper.

1 It is suggested in [17] to use RTR frames by the sender, to signalize the successful transmission. Since RTR frames are not received by hosts, the right choice would be an empty data frame.

2 Broadcast Consistency in the CAN Protocol

2.1 Mechanisms in the physical layer

The global consistency in the CAN bus is based on the fact that all nodes (including the sender) scan the value of every bit – at a 'safe' sample point – while it is transmitted. Hence all correct nodes have a consistent view of every bit.

Since the physical connection of the CAN bus (Fig. 1-A) is extremely robust against electro-magnetical noise, the signal level sensed at the sample point, reflects reliably the currently valid bit-value to be observed by all nodes.

The nodes of the CAN-Bus are connected logically via a wired-AND or, alternatively, a wired-OR function. In a wired-AND connection – as implemented in popular CAN controllers – a ‘1’ is a recessive bit and a ’0’ is a dominant bit (Fig. 1-B). Whenever different bit-values are put on the bus by different nodes simultaneously, the logical AND function of them is observed by all nodes. In the rest of the paper, the wired-AND connection is assumed.

2.2 Mechanisms in the data link layer

Error and Overload frames.

The Error Frame and the Overload Frame have identical structures. They both consist of six dominant bits without bit stuffing. Thus, they can be only distinguished by the time when they are transmitted. An error frame may be started only within a data or RTR frame, in order to interrupt the current transmission and signalize an error. An overload frame, however, may be started only immediately after the successful transmission of a data or RTR frame in order to indicate that a node is temporarily not able to accept further data frames due to an internal overload condition.

Error signaling.

Whenever a CAN node detects an error, it switches to the error signaling mode, and transmits an error frame. This violates either the bit stuffing rule or the frame format, and ensures that all other CAN nodes detect an error, too. After sending an error frame, a CAN node waits for all other nodes

Table 1. Effects of errors in CAN depending on their detection time and place
(EOF is the last bit of the frame, and EOF–1 is the last but one bit)

Error cause	Detection time and place	Effects of the error	Retransmission time	Frame level inconsistency
Any cause	Before EOF – 1	Error frame, all receivers drop the frame	Any time	No
Any cause	EOF – 1 by all nodes	Error frame, all receivers drop the frame	Any time	No
Faulty sender	EOF – 1 by sender	Error frame, all receivers accept the data	Any time < ∞	No (duplicate)
Faulty sender	EOF – 1 by sender	Error frame, all receivers accept the data	Never (crash)	No
Faulty sender	EOF – 1 by all receivers	Error frame, all receivers drop the frame	Any time	No
Faulty channel, faulty receiver	EOF – 1 by some but not all receivers	Error frame, some receivers drop the frame, some receivers accept it	Immediately after error frame	No (duplicate at some rec.)
Faulty channel, faulty receiver	EOF – 1 by some but not all receivers	Error frame, some receivers drop the frame, some receivers accept it	Interfered by other frames	Ambiguous ordering
Faulty channel, faulty receiver	EOF – 1 by some but not all receivers	Error frame, some receivers drop the frame, some receivers accept it	Never (due to sender crash)	Inconsistent delivery
Any cause	EOF; also by sender	Receivers accept data, sender retrans-mits	Any time < ∞	No (duplicate)
Any cause	EOF; also by sender	Receivers accept data, sender retrans-mits	Never (crash)	No
Any cause	EOF; not by sender	No effect, no retransmission	---	No

to finish their error frames (i.e. until it detects a recessive bit). A new data or RTR frame may start 11 bit-times after the bus level becomes recessive.

2.3 Inconsistency in CAN

Although CAN applies several techniques at the physical and data link layer to ensure reliable and consistent data transfer, inconsistencies among different CAN nodes are still possible.
The lowest level of inconsistency that is observable in a CAN bus, is the inconsistent view of a bit. Depending on when and why an inconsistency at the bit level happens, it is either repaired by error detection and error signaling mechanisms, or it grows to an inconsistency at the frame level, i.e. an inconsistent view of the frame status among different CAN nodes.
Table 1 shows the possible causes and effects of bit level errors in CAN. A detailed analysis is given in [10].

2.4 Anonymous error signaling constituting a weak-point

The data collected in Table 1 show clearly, that a frame-level inconsistency is caused whenever an error is detected at the end of a frame, and the CAN nodes are not able to achieve consensus on whether to discard the frame, or to accept it.
The reason of this lack of consensus is that it is not possible to indicate an error of the last bit of a frame – the error indication would seem to be an overload frame. Hence receivers are forced to ignore an error of the last bit of a frame.
However, if some but not all of the receivers detect an error in the last but one bit of a frame, they will discard the frame, and start an error signal immediately after the error detection, i.e. at the last bit of the frame. Every other receiver will then detect an error in the last bit. Those receivers will accept the frame, and inconsistency at the frame level occurs. In order to recover from this inconsistency, the CAN protocol specifies that a sender retransmits the frame upon the detection of an error in the last bit of a frame [14].
If the retransmission of a faulty frame could be guaranteed, the frame-level inconsistency would be repaired by detecting and discarding duplicate frames, and achieving consensus on the delivery order [12]. But if the sender crashes before a successful retransmission, then the frame is not retransmitted, resulting in inconsistent delivery.
In the following theorem it is shown that a frame-level consistency cannot be guaranteed by the CAN protocol itself.

Theorem 1. By using the error handling mechanisms of the CAN protocol, CAN

nodes cannot always achieve consensus on accepting or rejecting a frame.

Proof. The proof is based on the following facts:

i. For each frame, there must be a time where the frame is accepted, if it is not corrupted until then, and it is discarded, if an error was detected earlier. Let's call it the validation point. In CAN, the validation point is the beginning of the last bit of the frame.
ii. When a node decides to send an error frame, other nodes observe the error frame not before the following bit-time.

Due to these facts, if a subset of nodes detect an error at the last bit before the validation point, they must discard the frame. Although they immediately initiate an error frame, this error frame will be observed by other nodes after the validation point, thus they will accept the frame. Of course, those nodes will observe an *error frame* immediately after the validation point, but they may not discard the frame in this case. This is because those nodes cannot be sure that they are observing an *error frame* and not, for instance, a burst of dominant bits caused by physical faults.

If a node discards a frame due to the observation of an 'error frame' immediately after the validation point, and this 'error frame' is actually a burst error, it is possible that some other nodes in the system will observe this burst one bit later. Hence, those nodes would accept the frame and the result would be again a frame-level inconsistency. □

The inconsistency noticed above, is a fundamental problem whenever diffusing simple signals (e.g. 6 dominant bits) without authentication, to broadcast information (e.g. detection of an error). Since such signals are not authentic and can be produced by a faulty component (or a noisy channel), they cannot reliably lead to a consensus among all observers.

3 Consistent Broadcast Delivery by High Level Protocols

Consistent delivery of broadcast messages means that every message is either delivered correctly to all operational receivers at the protocol termination time, or to none of them at all. In order to decide on the delivery of a message at the protocol termination time, either all operational receivers must have received the message, or global agreement must be achieved, that at least one receiver has not received it. Reliable message transmission to all operational receivers can be ensured under anticipated fault conditions by appropriate retransmission mechanisms and resource adequacy [5] [6] [9] [13] [17]. A global agreement on the fact that at least one receiver failed to receive a message, however, needs at least a phase of individual acknowledgments and a phase of commitment, resulting in a two-phased commit protocol [1] [2] [3] [15] [20].

In the following, some approaches are described, which aim at achieving consistent broadcast delivery in the CAN bus.

3.1 The EDCAN protocol

The EDCAN (CAN Eager-Diffusion) protocol [17] is based on a multiple transmission policy similar to [6]. It has been developed to cope with the frame-level inconsistency of CAN and immediate sender crash. According to the EDCAN protocol, every node attached to the CAN bus keeps track of the bus traffic, and attempts to retransmit every frame that it receives. The protocol recognizes duplicates of the frames, and if a node observes a certain number of duplicates of a frame (Rufino et al named this number the *inconsistent omission degree*), it does not retransmit the frame any more.

The EDCAN protocol ensures the consistent delivery at the cost of a considerable portion of the bus bandwidth. E.g. in the simplest scenario with an "inconsistent omission degree" of 1, every receiver of a frame attempts to retransmit the frame until it receives a duplicate of it. Thus every frame is retransmitted at least by one receiver. However, after receiving the duplicate at a receiver, the protocol takes more than a few μ-seconds to decide on canceling the retransmission in the CAN controller. Since a transmission request can be cancelled only within 3 bit-times from the beginning of the bus-idle period, at high bus rates the frames are often retransmitted twice, resulting in an additional communication overhead of 200%.

3.2 The RELCAN protocol

Rufino et al. have also proposed a more efficient protocol called RELCAN [17], which uses the EDCAN protocol only when the receivers assume that the original sender has crashed. According to this protocol the sender first transmits the data frame, and immediately thereafter it transmits a short confirm frame without data. If a receiver receives the confirm frame within a given time period, it delivers the data to the application. Otherwise, it starts the eager diffusion of the data frame. Due to this "lazy diffusion" policy, the protocol needs less than 100% additional overhead in fault-free situations.

A problem related to the above approaches, is that the frames are retransmitted by ordinary nodes running application software. Thus a faulty application software on a node may manipulate the contents of received messages, before they are retransmitted by that node. This results in the Byzantine agreement problem. In order to obtain consistent data at the receiver sites, several retransmissions by different nodes may be necessary [1], or the communication subsystem of the nodes must be implemented as an independent hardware/software module with a well-defined data interface to the host, e.g. a FIFO dual-ported memory in each direction [16]. While the first solution consumes at least 80% of the restricted bus bandwidth, the second one demands for at least an additional microprocessor and a dual-ported RAM component at each node, resulting in high additional hardware costs.

4 Fault-tolerant Broadcast Using Shadow Retransmitters

As we can see in Table 1, CAN ensures reliable frame transmission (sometimes with duplicates or ambiguous ordering) unless a sender crashes after an inconsistent transmission of a frame, and before its successful retransmission. In this case a permanent frame-level inconsistency is caused.

The approach presented in this paper, solves the problem of the crashed sender by dedicated nodes, which act as 'Shadow Retransmitters' (SHARE's). These nodes detect the situations where an inconsistent frame transmission is possible, and retransmit the frame *simultaneously* with the original sender. Since the sender and all SHARE's retransmit *identical* frames simultaneously, these frames are transmitted as a single physical frame. Hence the network components will detect no conflicts, and the redundancy is not visible to them.

A frame is retransmitted by a SHARE only in situations where a retransmission by the original sender may be required, too. Thus, in real-time systems adequate bus resources must be provided for such retransmissions even when no SHARE's are used. Unlike other fault-tolerant broadcast protocols, the SHARE approach causes no additional communication overhead in fault-free situations.

This results in a transparent reliability improvement in the sense that the SHARE mechanism can be applied in an arbitrary CAN bus to achieve consistent broadcasting, without having to change the system components and scheduling. Existing CAN solutions may add an appropriate number of SHARE's to the bus in order to ensure consistent frame delivery even in the case of an inconsistent frame transmission followed by immediate sender crash.

The detection of a possible frame-level inconsistency is based on a simple fact: a frame-level inconsistency is possible if at least one non-faulty node starts an error frame in the last bit of a data frame (let's call it bit EOF). In this case, the sender is expected to retransmit the frame immediately. Thus SHARE's should also immediately retransmit a frame, whenever they observe an error frame starting at bit EOF. To tolerate the crash of the SHARE components, several SHARE's can be added to the bus.

4.1 Analysis of different error scenarios

Since SHARE's retransmit CAN frames whenever they detect a sequence of 6 dominant bits starting at bit EOF, the retransmission is performed only if a frame-level inconsistency is likely. Following is a discussion on the possible scenarios of error detection, which shows that SHARE's perform the retransmission whenever a frame-level inconsistency is possible:

1. *No node detects an error before EOF but at least one node detects a dominant bit sequence starting at EOF*: in this case, all receivers will accept the frame. If any SHARE detects this bit sequence, and the sequence is at least 6 bits long, then the SHARE will retransmit the frame. This sequence is very likely to be an error frame. In this case, the original sender will detect it and retransmit the frame unless it crashes.
   If the original sender does not detect the bit sequence at EOF, it won't retransmit it. In overload situations, it is possible that the retransmission by the SHARE is delayed until the original sender attempts to send a new frame with the same identifier. This could cause multiple collisions between the old and the new frame, and eventually the sender and SHARE could switch to bus-off state. In order to prevent this serious failure, SHARE's cancel the currently retransmitted frame upon transmission error detection. Thus only one single collision is possible between the retransmission of old data and the transmission of the new data. But this also means that SHARE's do not *always* perform the necessary retransmission.
   However, under the assumption of a bit error rate of 10^{-5}, a node MTTF of 10^4 hours, a message length of 100 bits, and permanent bus overload, the mean time to permanent frame-level inconsistency will be as high as 3.98×10^{11} hours, which is acceptable for safety-critical applications.
2. *No node detects an error before EOF–1 but at least one node detects an error in EOF–1:* in this case, those nodes will start an error frame at EOF, which is observed by all non-faulty nodes including SHARE's. The first group will discard the frame, and the second group will accept it. To tolerate an immediate sender crash after this error, the frame must be retransmitted by SHARE's. In order to guarantee that SHARE's receive and retransmit the frame in this situation, the GAL component of the SHARE locks the bus input on the recessive level during EOF–1 (see the signals RxD1, locked-RxD, and bus-in in Fig. 2). This means, that in this scenario all non-faulty SHARE's will retransmit the frame.
3. *At least one node detects an error before bit EOF–1:* in this case, those nodes will start an error frame before EOF, which is observed by all non-faulty nodes. Thus, all non-faulty nodes will discard the frame, and the frame delivery status is consistent. If the error frame starts at EOF–1, and some SHARE's did not detect the error before EOF–1, then those SHARE's will accept and retransmit the frame simultaneously with the sender. Otherwise, either the sender retransmits the frame, or it crashes, but in any case the frame status at all receivers is consistent.

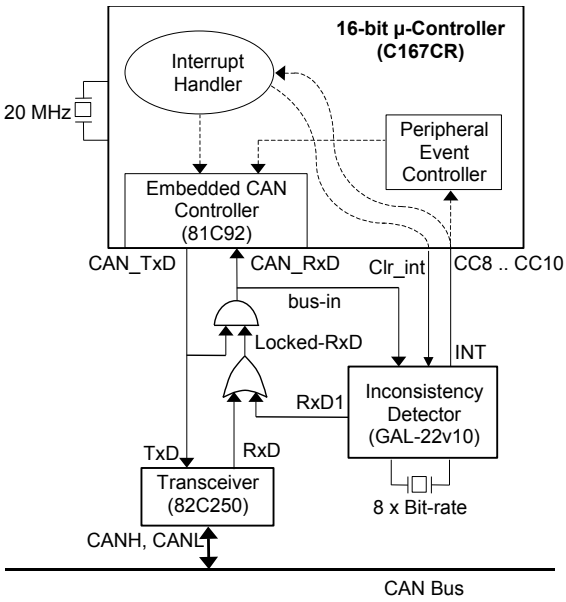

Fig. 2: The structure of a SHARE

4.2 Transparency and effectiveness of the SHARE approach

The main advantage of dedicated shadow retransmitters is their transparency. In order to perform the frame retransmission transparently, SHARE's must start the retransmission simultaneously with the sender. Thus after detecting an error frame beginning at the last bit of a frame, a SHARE must become ready to start its retransmission within 10 bit-times. At the maximum bit-rate of 1Mbit/s, a SHARE has no more than 10 µ-seconds to initiate the retransmission of a frame.

The main component of a SHARE is a C167CR micro-controller [18], which has an embedded CAN controller and a Peripheral Event Control (PEC) unit for fast interrupt handling. PEC is a feature that enables transferring a word or byte from a source address to a destination address upon an interrupt. SHARE uses 3 PEC transfer channels to:

1) convert the CAN buffer of the last received frame to an output buffer,
2) initiate its transmission, and
3) enable the next free buffer for receiving the next frame.

Using PEC channels, the whole procedure is completed in less than 3 µs at a processor clock rate of 20MHz, so the SHARE's are fast enough for operating at the highest bit-rate of 1Mbit/s. After the PEC transfers, a software interrupt handler prepares the PEC channels for the next activation.

The error detector is realized by a 22v10 Gate-Array Logic (GAL) which detects a sequence of a Zero, 7 One's, and 6 Zeros. This sequence appears on a CAN bus only in the following cases: 1) an error frame or a burst of at least 6 dominant bits starts at the bit EOF of a data frame, 2) an error frame or a burst of at least 6 dominant bits starts at the bit EOF of a RTR frame, or 3) a burst of at least 6 dominant bits is caused by a fault, 7 bit-times after the end of an error frame or overload frame. In the first case, SHARE has received a valid data frame, which is then retransmitted. In the second and third case, SHARE will transmit the original content of the receive buffer, which is an empty message with the lowest possible priority (ID= 2^{29}-1). Although this value is not a valid CAN ID, C167CR is able to transmit a message with this ID, which is not used by any correct CAN application. Of course, this situation is detected by the software interrupt handler, and the transmission is cancelled as soon as possible.

5 Conclusion

The paper addresses the fault scenarios where a frame-level inconsistency can arise in CAN. A transparent solution (the SHARE approach) which enforces the frame-level consistency in presence of faults, is proposed. In fault-free situations, SHARE's are passive components, and upon detection of a situation where a frame-level inconsistency is possible, SHARE's retransmit the frame simultaneously with the original sender in the same physical frame. Since SHARE components do not retransmit any frames that are not expected to be retransmitted by the original senders, adding SHARE components to a CAN bus does not affect the schedulability of the existing system.

The SHARE approach relies on the properties of the CAN protocol and provides reliable and consistent broadcast delivery. SHARE's can be added to arbitrary application systems with static [19] or dynamic [11] bus scheduling mechanisms.

The paper does not discuss the problem of consistent ordering of the frames. An approach to achieve total ordering of CAN frames, which relies on the frame-level consistency and timely message transmission, is proposed in [12]. Other approaches to achieve consistent ordering of messages in CAN are found in [17] and [21].

References

[1] Ö. Babaoglu and R. Drummond: "Streets of Byzantium: Network Architectures for Fast Reliable Broadcasts", *IEEE Tr. Software Eng. 11(6)546-554.*

[2] K.P. Birman and T.A. Joseph: "Reliable Communication in the Presence of Failures", *ACM Tr. Computer Systems, 5(1):47-76, Feb. 1987.*

[3] J.M. Chang and N.F. Maxemchuk: „Reliable broadcast protocols", *ACM Trans. on Computer Systems, 2(3), Aug. 1984, pp. 251-273.*

[4] CiA Draft Standards 201..207: „CAN Application Layer (CAL) for Industrial Applications", *may 1993.*

[5] F. Cristian et. al.: „Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement", IEEE *15th Int'l Symposium on Fault-Tolerant Computing Systems, Ann Arbor, Michigan, 1985.*

[6] F. Cristian: „Synchronous Atomic Broadcast for Redundant Broadcast Channels“, *The Journal of Real-Time Systems, Vol. 2, pp. 195-212, 1990*.

[7] DeviceNet Specification 2.0 Vol. 1, *Published by ODVA, 8222 Wiles Road - Suite 287 - Coral Springs, FL 33067 USA.*

[8] L.B. Fredriksson: „A CAN Kingdom (Rev. 3.01)“, *Published by KVASER AB, Box 4076, S-51104 Kinnahult, Sweden, 1996.*

[9] H. Kopetz and G. Grünsteidl: „TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems“, *Res. Report 12/92, Inst. f. Techn. Informatik, Tech. Univ. of Vienna, 1992.*

[10] M.A. Livani: „SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN“, *Research Report 98-14, University of Ulm, Faculty of Computer Science, Dec. 1998.*

[11] M.A. Livani, J. Kaiser, W. Jia: „Scheduling Hard and Soft Real-Time Communication in the Controller Area Network (CAN)“, *23rd IFAC/IFIP Workshop on Real Time Programming, Shantou, China, June 1998*.

[12] M.A. Livani and J. Kaiser: "A Total Ordering Scheme for Real-Time Multicasts in CAN", *24th IFAC/IFIP Workshop on Real Time Programming, Schloß Dagstuhl, Germany, May 1999*.

[13] P. Ramanathan and K.G. Shin: "Delivery of Time-Critical Messages Using a Multiple Copy Approach", *ACM Tr. Computer Systems, 10(2):144-166, May 1992*.

[14] ROBERT BOSCH GmbH: „CAN Specification Version 2.0“, *Sep. 1991*.

[15] L. Rodrigues and P. Veríssimo: „*x*AMP: a Multi-primitive Group Communication Service“, *IEEE Proc. 11th Symposium on Reliable Distributed Systems, Houston TX, Oct. 1992*.

[16] J. Rufino, N. Pedrosa, J. Monteiro, P. Veríssimo, G. Arroz: "Hardware Support for CAN Fault-Tolerant Communication", *Proceedings of the IEEE 5th Int'l Conference on Electronics, Circuits and Systems, Lisbon, Portugal, Sep. 1998.*

[17] J. Rufino, P. Veríssimo, C. Almeida, L. Rodrigues: „Fault-Tolerant Broadcasts in CAN“, *Proc. FTCS-28, Munich, Germany, June 1998*.

[18] Siemens AG: „C167 User’s Manual 03.96“, *Published by Siemens AG, Bereich Halbleiter, Marketing-Kommunikation, 1996*.

[19] K. Tindell and A. Burns: „Guaranteed Message Latencies for Distributed Safety-Critical Hard Real-Time Control Networks“, *Report YCS229, Department of Computer Science, University of York, May 1994*.

[20] P.J. Veríssimo, L. Rodrigues, M. Baptista: "AMp: A Highly Parallel Atomic Broadcast Protocol", *SIGCOMM'89 Symposium on Communications Architectures & Protocols, Austin, Texas, Sep. 1989..*

[21] K. M. Zuberi and K. G. Shin: „A Causal Message Ordering Scheme for Distributed Embedded Real-Time Systems“, *Proc. Symp. on Reliable and Distributed Systems, Oct 1996*.

University of Ulm
Department of Computer Structures
James-Franck Ring O27
89081 Ulm, Germany
Phone: +49 (731) 502 4177
Fax: +49 (731) 502 4182
Email: mohammad@informatik.uni-ulm.de
http://www.informatik.uni-ulm.de