Centro de Sistemas Telemáticos e Computacionais

Instituto Superior Técnico

NavIST Group
Fault-Tolerant Real-Time Distributed
Systems and Industrial Automation

Control of Inaccessibility in CAN*ELy*

CSTC Technical Report RT-03-03

J. Rufino, P. Veríssimo, G. Arroz

December 2003

Centro de Sistemas Telemáticos e Computacionais - NavIST Group
Fault-Tolerant Real-Time Distributed Systems and Industrial Automation

Control of Inaccessibility in CAN*ELy*

This report has been submitted for publication. Please, do not distribute widely.

Technical Report: CSTC RT-03-03
Authors: J. Rufino, P. Veríssimo, G. Arroz
Date: December 2003

Avenida Rovisco Pais, 1049-001 Lisboa - PORTUGAL, Tel: +351-1-8418397/99, Fax: +351-1-8417499.
NavIST WWW Page URL - http://pandora.ist.utl.pt.

Control of Inaccessibility in CAN*ELy*

José Rufino
ruf@digitais.ist.utl.pt
IST-UTL*

Paulo Veríssimo
pjv@di.fc.ul.pt
FC/UL†

Guilherme Arroz
egsa@alfa.ist.utl.pt
IST-UTL

Abstract

Continuity of service and bounded and known message delivery latency, are reliability requirements of a number of real-time applications, such as those served by fieldbuses. The analysis and design of such networks w.r.t. timing properties has, with few exceptions, been based on no-fault scenarios, rather than under a combined performance and reliability perspective. We have shown in earlier works that the performability of fieldbuses in normal operation is hindered by periods of inaccessibility. These derive from incidents in the protocol operation that affect non-faulty components, leading to failures of the expected hard real-time properties of the network.

This is specially relevant if the fieldbus supports critical control applications (e.g. avionics, automotive). As part of our endeavor to design a CAN-based infrastructure capable of extremely reliable communication, that we have dubbed CAN Enhanced Layer (CANELy), this document provides a detailed analysis of CAN behavior in the presence of inaccessibility, discussing a generic methodology to enforce system correctness in the time-domain, despite the occurrence of network errors.

1 Introduction

Continuity of service and bounded and known message delivery latency are two fundamental requirements of fault-tolerant real-time systems and applications. Fieldbus technologies, such as the Controller Area Network (CAN), play nowadays a fundamental role in the design and implementation of embedded distributed systems. Those network infrastructures are expected to exhibit reliable hard real-time behavior in the presence of disturbing factors such as overload or faults.

The major consequence of such disturbances on real-time communication is the error they introduce in the specification of timing bounds, such as deadlines. Most analyses of message transmission delays or of network schedulability assume the local area network (LAN) or fieldbus as always operating normally. However, even if one excludes solid faults such as physical partitioning, LANs and fieldbuses are subject to periods of *inaccessibility*. They derive from incidents in the LAN or fieldbus operation that temporarily prevent communication and whose effect is to increase the network access delay as seen by one or more nodes. This may lead to the failure of task or protocol timing specifications and ultimately, to the failure of the hard real-time system.

As part of our endeavor to design a CAN-based infrastructure support for extremely reliable distributed computer control, dubbed **CAN Enhanced Layer** (CANELy) we have been addressing the problem of fault-tolerant real-time communications on fieldbuses in a comprehensive way, reviewed in Section 2 for completeness.

This document provides a detailed study of CAN with respect to inaccessibility and discusses how to enforce system correctness in the time-domain, despite the occurrence of network errors.

*Instituto Superior Técnico - Universidade Técnica de Lisboa, Avenida Rovisco Pais, 1049-001 Lisboa, Portugal. Tel: +351-218418397/99 - Fax: +351-218417499. NavIST Group CAN WWW Page - http://pandora.ist.utl.pt/CAN.

†Faculdade de Ciências da Universidade de Lisboa, Portugal. Navigators Home Page: http://www.navigators.di.fc.ul.pt.

The report is organized as follows: Section 2 reviews relevant details of the CANELy architecture; Section 3 presents the system model; the hard real-time operation of CAN is addressed in Section 4 and the crux of the document, that is the control of CAN inaccessibility is discussed in Section 5; a reference to related work (Section 6) and some final remarks (Section 7) conclude the document.

The following discussion assumes the reader to be fairly familiar with CAN operation. In any case, we forward the reader to the relevant standard documents [8, 5], for details about the CAN protocol.

2 CANELy: a CAN-based Fault-Tolerant Real-Time Distributed System

In the course of analyzing existing studies of CAN limitations with respect to the provision of strict availability, reliability and timeliness attributes [9], we have realized that what was missing in the native CAN fieldbus to attain levels of dependability comparable to those of similar technologies, such as the Time-Triggered Protocol [10], was indeed a set of fault tolerance and timeliness-related services. Moreover, we have shown that these can be provided off-the-shelf (i.e. without modifications to the CAN standard or to existing CAN controllers), through the use of properly encapsulated additional software/hardware components. We call the materialization of this concept **CAN Enhanced Layer** (CANELy).

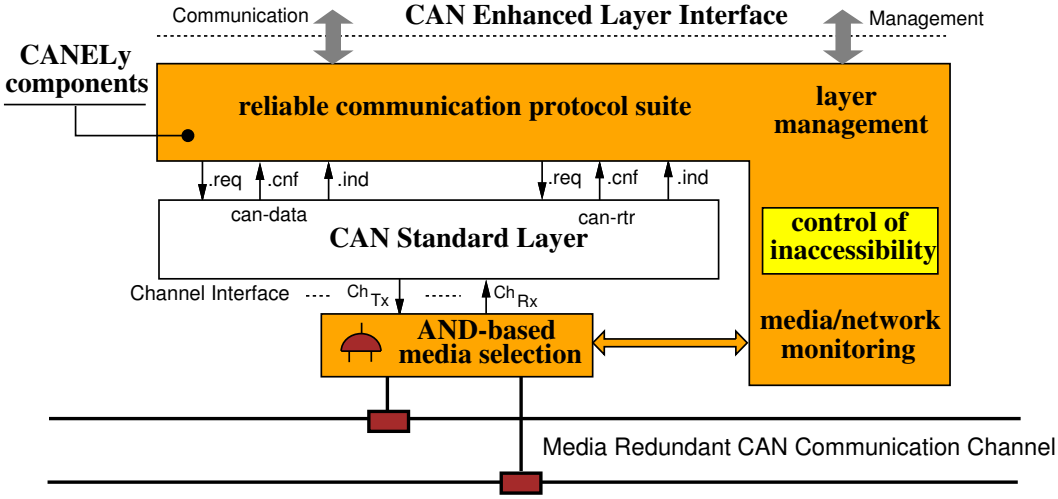

Figure 1: CAN Enhanced Layer architecture

The central component of the CANELy architecture (Figure 1) is naturally the standard CAN layer, complemented/enhanced with some simple machinery and low-level protocols, which include: a network infrastructure resilient to physical partitioning [18]; a reliable communication protocol suite, offering a set of broadcast/multicast primitives [20]; clock synchronization [16]; node failure detection and membership services [19].

The objective of this report is to discuss the mechanisms and the techniques used in the CANELy architecture to enforce system correctness in the time-domain despite the occurrence of network errors, that is *control of inaccessibility*.

CAN Standard Layer

The CAN fieldbus is a multi-master network that uses a twisted pair cable as transmission medium [8, 5]. The network maximum length depends on the data rate. Typical values are: 40m @ 1 Mbps; 1000m @ 50 kbps. Bus signaling takes one out of two values: *recessive* (r), otherwise the state of an idle bus; *dominant* (d), which always overwrites a recessive value. This behavior, together with the uniqueness of frame identifiers, is exploited for bus arbitration. A *carrier sense multi-access with deterministic collision resolution* policy is used. When several nodes compete for bus access, the node transmitting the frame with the lowest identifier always goes through and gets the bus. A *frame* is a network-level piece of encapsulated information. It may contain a *message*, a user-level piece of information. In CAN, a *data frame* is used for that purpose. However, it may consist of control information only, such as a *remote frame*.

The CAN standard layer is made from a CAN controller and the corresponding software driver that includes the following primitives (cf. Figure 1): *request* the transmission (.req) of data (can-data) or control (can-rtr) messages[1]; *confirm* to the user a successful message transmission (.cnf); *indicate* a message arrival (.ind).

Basic Dependability of the CAN Protocol

The fault confinement mechanisms built in the native CAN protocol aim at restricting the influence of defective nodes in bus operation. They are based on two counters recording, at each node, transmit and receive errors, that is, *omission errors* causing frames not to be received at their destinations. Most omissions are perceived consistently by the error detection mechanisms of all nodes. However, some subtle errors can lead to inconsistency and induce the failure of dependable communication protocols based on CAN operation alone [20]. Inconsistent frame omissions may occur when faults hit the last two bits of a frame at some nodes[2], which may cause: the message to be accepted in duplicate by a subset of recipients; inconsistent message omission, if the sender fails before retransmission. A thorough discussion of these failure scenarios can be found in [20].

Furthermore, the fault-confinement mechanisms themselves may be a source of inconsistency. Inconsistent frame omissions may occur if a node is allowed to enter the so-called *error passive* state, where it is able to transmit and receive frames, but can only signal errors while transmitting. Other states for the node are: *error active*, the normal operating condition, where the node is fully-integrated, being able to transmit/receive frames and to fully participate in error detection/signaling actions; *bus off*, where the node does not participate in any bus activity, being unable to send or receive frames.

3 System Model

In this section, we enumerate our fault assumptions for the system and discuss the CAN properties that underpin our system model, as established in [20, 18, 17].

We introduce the following definition: a component is **weak-fail-silent** if it behaves correctly or crashes if it does more than a given number of omissions – called the component's *omission degree* – in a time interval of reference [22].

The **CAN bus** is viewed as a single-channel broadcast local network with the following failure semantics for the network components:

[1] Control messages are encapsulated in remote frames.

[2] The subset may have only one element. Examples of causes for inconsistent detection are: electromagnetic interference or deficient receiver circuitry.

- individual components are **weak-fail-silent** with *omission degree* f_o;
- failure bursts never affect more than f_o transmissions in a time interval of reference[3];
- omission failures may be inconsistent (i.e., not observed by all recipients);
- there is no permanent failure of the channel (e.g. the simultaneous partitioning of all redundant media [18]).

The weak-fail-silent assumption can be enforced with high coverage for the CAN controller by fault confinement mechanisms [20, 17]. This is important for the preservation of CAN timeliness and for the parameterization of protocols operating on top of the CAN standard layer, such as those specified in the CANELy architecture [20, 16, 19].

The CAN fieldbus has a medium access control (MAC) sub-layer that in essence exhibits the same kind of properties identified in previous works on LANs [22]. Then, on top of the basic MAC sub-layer functionality, CAN has error-recovery mechanisms defining message-level properties that, again, have the flavor of the logical link control (LLC) sub-layer in LANs. Next, we summarize a relevant set of error detection/recovery and timeliness-related properties, at MAC and LLC levels.

CAN MAC and LLC properties

The upper part of Figure 2 enumerates a relevant set of CAN MAC-level properties, defined in previous works [20, 17]. Property MCAN1 formalizes the effect of CAN built-in error detection and signaling mechanisms, and it implies that frame errors are transformed in omissions. Most frame errors are handled consistently by all correct nodes. The residual probability of undetected frame errors is negligible [4]. Property MCAN2 maps the failure semantics introduced above onto the operational assumptions of CAN, being $k \geq f_o$. This property is crucial to achieve reliable hard real-time operation of CAN-based infrastructures, CANELy included [17].

MAC-level properties
MCAN1 - Error Detection: correct nodes detect any corruption done by the network in a locally received frame.
MCAN2 - Bounded Omission Degree: in a known time interval T_{rd}, omission failures may occur in at most k transmissions.
MCAN3 - Bounded Inaccessibility: in a known time interval T_{rd}, the network may be inaccessible at most i times, with a total duration of at most T_{ina}.
MCAN4 - Bounded Transmission Delay: any frame queued for transmission is transmitted on the network within a bounded delay of $T_{td}+T_{ina}$.
LLC-level properties
LCAN1 - Bounded Inconsistent Omission Degree: in a known time interval T_{rd}, inconsistent omission failures may occur in, at most, j transmissions.

Figure 2: Relevant CAN MAC and LLC-level properties

The behavior of CAN in the time-domain is described by the remaining MAC-level properties. Property MCAN4 specifies a maximum frame transmission delay, which is T_{td} in the absence of

[3]For instance, the duration of a message broadcast round. Note that this assumption is concerned with the total number of failures of possibly different components.

faults. The value of T_{td} includes the queuing, access and transmission delays. It depends on message latency classes and offered load bounds [21, 25, 11] and in general it may also include the extra delays resulting from the queuing effects caused by the periods where the network refrains from providing service, although remaining operational (i.e. the periods of *inaccessibility*). The bounded frame transmission delay naturally includes T_{ina}, a corrective term which accounts for the worst-case duration of inaccessibility, given the bounds specified by property MCAN3. The inaccessibility characteristics of CAN depend on the network alone and can be predicted by the analysis of the CAN protocol [23].

Finally, at the LLC level, the failure modes that we have identified cause the message-level properties of CAN to be somewhat different. While the omission failures specified by MCAN2 are masked in general at the LLC level by the retry mechanism of CAN, the existence of inconsistent omissions implies that some j of the k omissions will show at the LLC interface as inconsistent omissions [20, 17].

Property LCAN1 (cf. lower part of Figure 2) specifies then the probability of inconsistent omission failures j, where j is normally several orders of magnitude smaller than k. Property LCAN1 has been addressed in the design of the CANELy reliable communication protocol suite [20, 16, 19].

4 Hard Real-Time Operation of CAN

Before attempting to obtain reliable hard real-time behavior out of a standard LAN or fieldbus, CAN included, one has to stipulate the failure modes that must be taken into account. Thus, in conformance to the discussion in Section 3, let us assume the following failure modes for CAN-based systems: timing failures (delays) due to transient overloads; omission failures (lost frames) due to transmission errors; network partitions (physical/virtual). In addition, let us assume that the network channel is not replicated.

The following conditions must be observed to secure reliable hard real-time communication in the CAN fieldbus [24, 23]:

RT1 - *enforce bounded delay from request to transmission of a frame, given the worst-case load conditions assumed (prevent timing faults);*

RT2 - *ensure that a message is delivered despite the occurrence of omissions (tolerate omission faults);*

RT3 - *control partitioning.*

Condition RT1 if met, assures that a frame is sent within a known time bound, even if it does not arrive. Condition RT2 stipulates the tolerance to omission faults and ensures that a message is delivered, even if that implies the transmission of several frames (using either time or space redundancy).

Condition RT3 is related to the maintenance of connectivity between network nodes. Note that partitioning can be tolerated with complete network replication, so one might wonder why worry about inaccessibility glitches. At least two reasons make the approach worthwhile. Firstly, network replication implies a more expensive infrastructure and more complex protocols. Since partitioning typically affects the physical medium, a solution based on a simplex network with dual-media, such as depicted in Figure 1, is extremely effective. Secondly, even with network replication, inaccessibility affecting individual replicas would lower their fault coverage, that is, the probability that each of them is correct (in the time domain). Thus, the mechanisms we present in this document would also be applicable to individual replicas of a redundant network.

Achieving RT3 has its complexity, and requires the utilization of appropriate design techniques [22]. The discussion of how to secure RT3 in CAN-based systems is the central issue of this report. For completeness, we start by giving guidelines on how to achieve RT1 and RT2.

Enforcing Bounded Transmission Time

Enforcing condition RT1, i.e. securing an upper bound on frame transmission delay (property MCAN4, in Figure 2), depends on:

- the offered load bounds, defining the temporal characteristics of message transmit requests (e.g. interarrival time, message length and urgency level).
- the own functionality of the CAN fieldbus, which includes: the determinism of the MAC-level mechanisms; the (global) scheduling of message transmissions, taking into account message delivery constraints; network sizing and parameterizing.

The CANELy architecture [17] is flexible enough to accommodate the use of a comprehensive set of message scheduling techniques, such as those described in [21, 25, 11]. The low-level engineering of such techniques should provide, at the interface with the CAN controller, adequate support with respect to: the management of message buffers [17]; avoidance of priority inversion incidents [12].

Handling Omission Failures

The standard CAN protocol has been designed to enforce condition RT2 at the lower levels of communication. Upon the detection/signaling of a frame error: the recipients discard the incorrect frame; the sender automatically submits the same message for retransmission.

However, we have identified in [20] a set of subtle failure modes that may lead to an inconsistent delivery of a message: a node in the *error passive* state detects an incorrect incoming frame and it is unable to signal the error to other nodes; a sender fails, after the inconsistent dissemination of a message and before message retransmission.

In the CANELy architecture, the erratic behavior of an *error passive* node is avoided, by forcing it to enter the *bus off* state, after the node has given a pre-specified number of omission errors, prior to reaching the *error passive* state. This secures the weak-fail-silent assumption [17].

On the other hand, given that condition RT2 cannot be secured by the CAN protocol alone, the CANELy architecture includes a software module (micro-protocol), built on top of the exposed CAN interface, that ensures a given message is delivered, despite sender failure. The implementation of this micro-protocol uses a diffusion-based technique that exploits property LCAN1 (cf. Figure 2) to optimize the utilization of CAN bandwidth.

Controlling Partitions: Inaccessibility

Condition RT3 implies the control of the effects of network partitioning on the timeliness of the system and applications. Our approach to this problem relies on a general inaccessibility model, which allows to treat the effects of the different causes for partitioning in a uniform way [24, 22, 23]. This is also true of physical partitioning, if the system has repair (e.g. medium redundancy or reconfiguration) [22].

Let then a network be partitioned when there are subsets of the nodes which cannot communicate with each other[4].

[4]The subsets may have a single element. When the network is completely down, *all* subsets have a single element, since each node can communicate with no one.

However, even in a physically connected network the occurrence of certain events in its operation (e.g. entry or leave of nodes) or of individual failures (such as: bit errors; transmitter or receiver glitches; node failures) may produce side-effects on the other nodes, which are a subtle form of partitioning, virtual rather than physical. Standard LANs and fieldbuses have their own means of recovering from these situations, but since this recovery process takes time, the network will exhibit periods where service is not provided to some or all of the nodes. If such kind of (inaccessibility) faults are not tolerated, timeliness of applications is at stake, which is not acceptable in a hard real-time system.

The problem of inaccessibility was thoroughly equated in previous works on LANs and fieldbuses [24, 23]. Its definition, in [24], is recapitulated next.

Inaccessibility*:*

i) *a component* **temporarily refrains** *from providing service;*

ii) *its users perceive that state;*

iii) *the limits of the periods of inaccessibility (duration, rate) are specified;*

iv) *violation of those limits implies the permanent failure of the component.*

The approach taken and the techniques used to tolerate inaccessibility faults will: allow a set of temporary network partitions; stipulate limits for the duration of the resulting glitches on protocol execution; require from the network components some self-assessment capability. Thus, it is necessary to complement the LAN or fieldbus functionality [24], in order to:

- *guarantee recovery from all conditions leading to partition (physical or virtual) in a given failure scenario, i.e. reestablish connectivity among affected nodes;*
- *ensure that the number of inaccessibility periods and their duration have a bound and that it is suitably low for the service requirements;*
- *accommodate inaccessibility in protocol execution and timeliness calculations, at all the relevant levels of the system.*

This way, all partitions are *controlled*. Uncontrolled partitions are of course still possible, because systems do fail, but that event means the total and permanent failure of the real-time communication system.

5 Implementing Inaccessibility Control in CAN

This section is entirely dedicated to the discussion of how to control inaccessibility in CAN-based systems. Without loss of generality, we address how the standard CAN error-handling mechanisms can be combined with some simple protocol extensions and other resources (e.g. a timer agency), being then integrated into efficient inaccessibility control methods. This has been accomplished with success in the CANELy architecture [17].

5.1 Handling CAN Physical Partitions

In [18], we have presented an innovative and extremely simple scheme for handling CAN physical partitions. The receive signals of each medium are combined in a conventional AND gate before interfacing the MAC layer. This secures resilience to medium partitions and stuck-at-recessive failures in the network cabling.

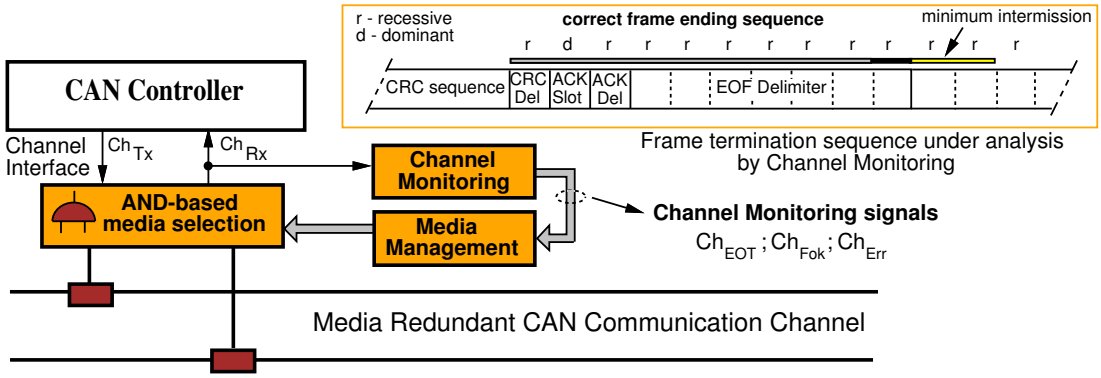

Figure 3: CAN Media Redundancy Mechanisms

The channel monitoring and media management modules, represented in Figure 3, implement additional fault treatment functions, such as the provision of resilience to stuck-at-dominant failures. Those modules are connected through the set of signals that we have specified in [18], given the observable behavior of CAN at the PHY-MAC interface:

- Ch_{EOT}, this signal is asserted at the end of each frame transmission, when the minimum bus idle period that precedes the start of every data or remote frame transmission has elapsed. It is negated at the start of a frame transmission.
- the *Frame correct* signal (Ch_{Fok}) is asserted if a data or remote frame transmission ends without errors. It is negated upon the assertion of Ch_{EOT}.
- conversely, the Ch_{Err} signal is asserted whenever an *error flag*, violating the bit-stuffing coding rule is detected [8]. It is negated upon the assertion of the Ch_{EOT} signal.

In the context of this document, one use of these signals is in the evaluation of the real value of the Channel omission degree, as specified[5] in equation (1). A Channel exceeding the allowed omission degree bound, k, given by property MCAN2, should be considered failed.

$$Ch_{Od} \overset{=}{\uparrow_{Ch_{EOT}}} \begin{cases} Ch_{Od} + 1 & \textbf{if } Ch_{Err} \wedge \neg Ch_{Fok} \\ 0 & \textbf{if } Ch_{Fok} \end{cases} \quad (1)$$

5.2 CAN Accessibility Constraints

The study of CAN accessibility constraints presented in [23] has established analytical expressions for all inaccessibility events, showing that their durations are bounded, and allowing the determination of those bounds.

In CANELy [17], the corrective actions taken to enforce the weak-fail-silent assumption for the network components have allowed an interesting, though moderate, reduction of the inaccessibility worst-case duration bounds, in some scenarios. These results are summarized in Figure 4. It is worth noticing the low worst-case figure of the bus reconfiguration delay (209 μs @ 1Mbps), compared with other failure scenarios and, in particular, with the 100 ms of commercial systems currently available[13].

[5] The notation $\overset{=}{\uparrow_{Ch_{EOT}}}$, in equation (1), means it is evaluated upon the assertion of the Ch_{EOT} signal.

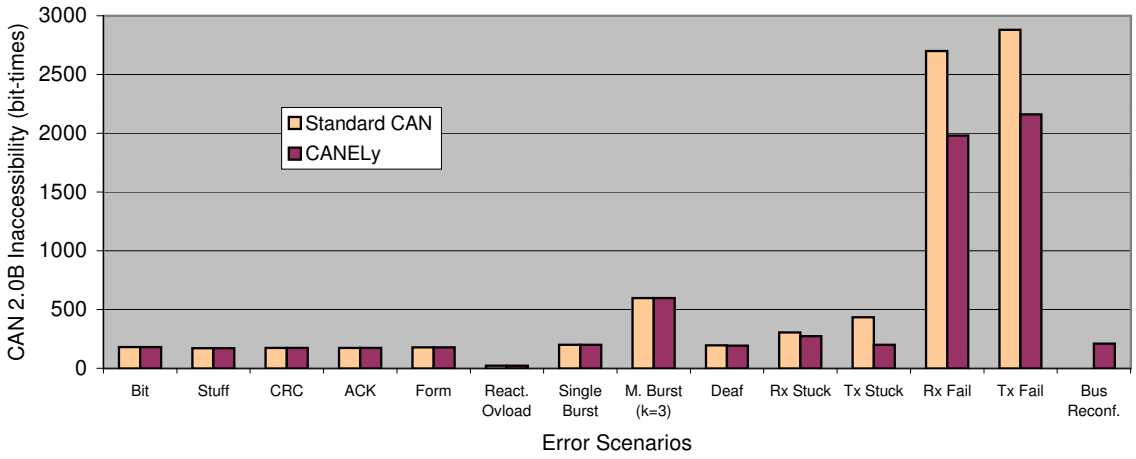

Figure 4: CANELy vs. standard CAN normalized inaccessibility bounds

5.3 Inaccessibility Control Methods

The next step towards a reliable hard real-time operation of CAN concerns the accommodation of the periods of inaccessibility in the timeliness model. This includes: the calculation of the real worst-case message transmission delays; the calculation of the real worst-case protocol execution times; the dimensioning of timeouts. These issues were addressed for the first time in [24, 22], which provided a general analysis for LANs. We reanalyze the problem in the context of CAN.

Are LAN-based solutions effective in CAN?

Diffusion-based masking algorithms allow tolerance to k omission faults, without needing to use timeouts: the protocol systematically repeats a transmission $k+1$ times; network accessibility is implicitly signaled, when a transmission is confirmed. A protocol inspired by this scheme is used in CANELy, handling inconsistent omission failures [20].

Conversely, acknowledge-based algorithms or protocol entities performing the surveillance of remote parties, make use of timeouts. In CANELy, timeout-based protocols are extensively used [20, 19]. Timeout values are set as a function of protocol execution times, which in general include terms that depend on the worst-case message transmission delay, T_{td} (MCAN4). Let us assume that: T_{td}, represents the optimum value for the delay to detect a timing, omission or crash failure; local clocks (timers) are used to implement timeouts.

Inaccessibility Addition

The simplest method to control the effect of inaccessibility on the operation of timeout-based protocols, is to add a corrective term, T_{ina}, to the timeout values set in function of T_{td} (MCAN4). T_{ina} is defined in MCAN3 and accounts for the worst-case duration of an inaccessibility incident. *No further action is needed to tolerate inaccessibility faults*.

The engineering of such a method, called herein *inaccessibility addition*, is extremely simple, given that the management of protocol timers can be easily mapped into the service interface of a standard timer agency [1, 7]. However, the use of this method is not interesting if T_{ina} has a value much greater than T_{td}, as it happens at the lower levels of CAN communication.

Inaccessibility Trapping

An alternative is to use *inaccessibility trapping* [24, 22]. This method is based on the transformation of inaccessibility periods into omissions.

Protocol timers, such as $T_{waitRemote}$ in the diagram of Figure 6, do not include T_{ina}, being set to the optimum timeout value T_{td}. A timer is started only after the confirmation that the transmission was issued. Should the network be inaccessible, the confirmation does not come. The protocol is prevented from proceeding and the timer start is postponed until the network becomes accessible again (e.g. situation I1, in Figure 6). Furthermore, all inaccessibility events occuring between two consecutive network accessibility signals (e.g. situations I2 and I3 in Figure 6) are transformed into *one* "omission". A timer may expire and a recovery process may have to be initiated. However, a protocol designed to take care of k omissions in the system, will keep working for $k+i$ "omissions", being i the maximum number of inaccessibility occurrences during protocol execution (MCAN3).

The analysis of the performance of LAN-based inaccessibility control methods, with respect to the delay for the detection of a failure, is presented in the upper half of Figure 5. The inaccessibility trapping method is effective in LANs, because the delay to detect the failure, i.e. absence of acknowledgment after $k+i+1$ tries, is increased at most by t_{ina}, the real duration of an inaccessibility incident, which may be much lower than T_{ina}.

Failure Detection Latency			
Network	Scenario	Inac. Control Method	
		Addition	Trapping
LANs	*no inac.*	$(k+1)\,.\,(T_{td}+T_{ina})$	$(k+i+1)\,.\,T_{td}$
$(k_{LLC}>0, i_{LLC}\geq 1)$	*inac.*	$(k+1)\,.\,(T_{td}+T_{ina})$	$(k+i+1)\,.\,T_{td}+t_{ina}$
CAN	*no inac.*	$T_{td}+T_{ina}$	$2\,.\,T_{td}$
$(k_{LLC}=0, i_{LLC}=1)$	*inac.*	$T_{td}+T_{ina}$	$2\,.\,T_{td}+t_{ina}$

Figure 5: Effectiveness of LAN-based inaccessibility control in CAN

The results presented in the lower half of Figure 5, for CAN, are derived directly from the LAN results, using the CAN LLC-level specific parameters. The frame retransmission scheme of CAN: secures $k_{LLC}=0$, when a frame transmission is confirmed; makes useless the incorporation of sender-based retry mechanisms on top of CAN; represents the fundamental reason why the inaccessibility trapping method should be adapted and optimized for CAN operation.

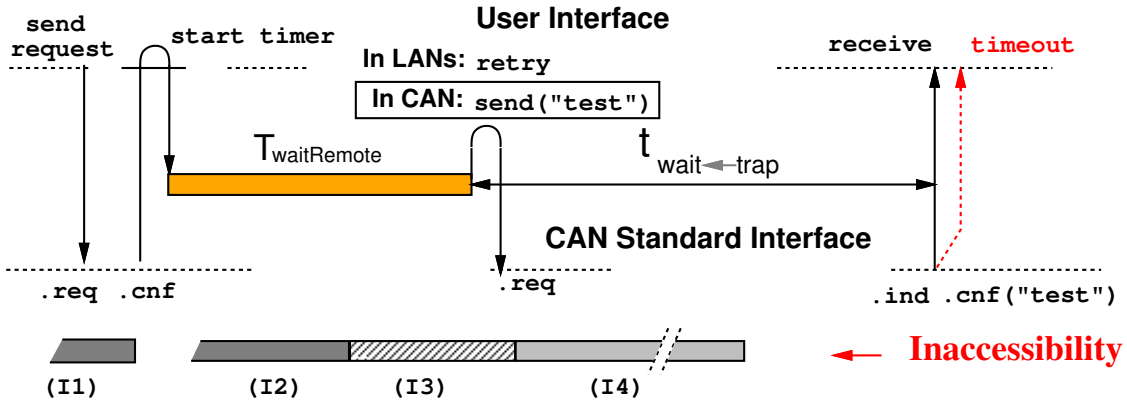

Figure 6: Timing of the CAN inaccessibility trapping method

In a CAN-oriented version of the inaccessibility trapping method (Figure 6), an accessibility test is started, upon a timeout. The properties of the priority-based CAN arbitration mechanism are exploited for this purpose. A CAN remote frame, having a "dummy identifier" with an urgency level lower than the message that is being awaited for by the protocol, is submitted for transmission. If it is confirmed before the arrival of the expected message that is an implicit signal of network accessibility and the protocol is notified that the timer has expired. Otherwise, the timeout signal is ignored.

In consequence, all the inaccessibility periods occuring inside the interval defined by the accessibility signals implicitly provided by the standard CAN communication primitives, are seen as "one" single event, securing $i_{LLC}=1$. Another consequence is that, upon a timeout, each expiring timer is always extended by $t_{wait \leftarrow trap}$ (cf. Figure 6), the effective duration of the accessibility test, making inaccessibility trapping in CAN a non-optimal method with respect to: the utilization of CAN bandwidth; the delay in the detection of a failure.

That is, none of the studied methods provide an effective solution for the problem of controlling CAN inaccessibility at low-level protocols. The question that remains to be answered is whether such a solution, even if tied to CAN, exists?

Towards CAN-oriented inaccessibility control methods

One idea would be the integration of inaccessibility control with the error notifications provided by existing CAN controllers: protocol timers would be started with an optimum timeout value, being extended by T_{ina} upon the notification of an error. Unfortunately: some CAN controllers do not generate error notifications in all situations (e.g. overload errors [6]); the error notification signal may not be delivered at some nodes (property LCAN1).

Other approach is the use of special-purpose mechanisms. Though we do not advocate the use of such solutions as a general rule, in CAN those mechanisms are not complex to implement, being common, in a great extent, to the bus media redundancy machinery.

For example, to evaluate CAN inaccessibility with respect to the number of events, we use the Channel monitoring signals, Ch_{Err} and Ch_{EOT}, as specified[6] in equation (2).

$$Ch_{Ie} \overset{=}{\uparrow_{Ch_{EOT}}} \begin{cases} Ch_{Ie}+1 & \textbf{if } Ch_{Err} \\ 0 & \textbf{when } \textit{mngt. request} \end{cases} \tag{2}$$

The Ch_{Ie} counter is monotonically incremented, being cleared only through the issuing of a specific layer management action request. In a period of reference T_{rd}, the effective number of inaccessibility incidents, i_e, is given by the variation of Ch_{Ie} (MCAN3).

On the other hand, to evaluate the duration of each CAN inaccessibility incident, one needs to mark when a period of inaccessibility begins and of how long it lasts. Should a frame transfer be disturbed by errors, a non-negligible time interval may exist between the beginning of the period of inaccessibility and the start of error recovery[7]. That period needs to be accounted for, in addition to the usually shorter period of error recovery, to obtain the total duration of the inaccessibility event.

This calls for extra mechanisms, not available in commercial CAN controllers, and that did not have to be included in the machinery of Figure 3, but that can be easily added to the latter.

We made the following operational assumptions concerning the observable behavior of CAN at the PHY-MAC interface, as *per* the standard [8]:

[6] Again, the notation $\overset{=}{\uparrow_{Ch_{EOT}}}$, means evaluation upon the assertion of the Ch_{EOT} signal.

[7] Signaled through the assertion of the Ch_{Err} signal.

N1 - *there is a detectable sequence that identifies the beginning of a possibly correct CAN data or remote frame transmission.*

N2 - *there is a detectable and unique fixed form sequence that identifies the correct transmission of a CAN data or remote frame.*

N3 - *there is a detectable and unique fixed form sequence that identifies the correct termination of a CAN data or remote frame.*

Assumption N1 describes the method used by standard CAN controllers to detect the beginning of a frame transmission, after a *minimum intermission* (cf. Figure 3) period has elapsed. The Ch_{SOF} signal is asserted during one bit-time when a dominant (d) bit is detected after the assertion of the Ch_{EOT} signal, as described by equation (3).

$$Ch_{SOF} \mapsto \begin{cases} true & \textbf{if } Ch_{EOT} \wedge Ch_{Rx} = d \\ false & \textbf{when } Ch_{SOF} \end{cases} \tag{3}$$

Assumptions N2 and N3 specify, respectively, the conditions whereby a frame transfer is completed without being disturbed by omission failures or by other inaccessibility events (e.g. overload errors). The Ch_{Tok} and Ch_{IFS} signals, specified by equations (4) and (5), are asserted only if no violation in the termination sequence[8] of a data/remote frame occurs up to the first/second bit of the *intermission*, respectively.

$$Ch_{Tok} \mapsto \begin{cases} true & \textbf{if } Ch_{Rx} = rdrrrrrrrrr \\ false & \textbf{when } Ch_{EOT} \end{cases} \tag{4}$$

$$Ch_{IFS} \mapsto \begin{cases} true & \textbf{if } Ch_{Rx} = rdrrrrrrrrrr \\ false & \textbf{when } Ch_{EOT} \end{cases} \tag{5}$$

In addition, we define an auxiliary signal, Ch_{Tok-p}, as a one bit-time pulse signal generated whenever the Ch_{Tok} is asserted.

A conservative approach is taken in assessing the duration of inaccessibility events: the scheduling of a frame for retransmission is assumed, on account of a possible inconsistent omission; the transmission of a data/remote frame is *a priori* considered a period of inaccessibility. Thus, equation (6) specifies that: a timer is started when the Ch_{SOF} signal is asserted; it is restarted when the transmission of a data/remote frame succeeds and, again, when the *minimum intermission* period has elapsed.

$$\mathcal{T}_{e\_ina} = \begin{cases} 0 & \textbf{if } Ch_{SOF} \vee Ch_{Tok-p} \vee Ch_{IFS} \\ \mathcal{T}_{e\_ina} + \mathcal{T}_{bit} & \textbf{if } \neg Ch_{EOT} \\ \mathcal{T}_{e\_ina} & \textbf{if } Ch_{EOT} \end{cases} \tag{6}$$

where: $\mathcal{T}_{e\_ina}$, is the normalized duration of one inaccessibility event; $\mathcal{T}_{bit}$, is the normalized duration of a bit.

At this stage, we have succeeded in obtaining the means to assess CAN operation with respect to the effective number, i_e, and duration[9], t_{e_ina}, of a single inaccessibility event[10].

[8] A fixed form sequence of recessive(r)/dominant(d) bits.

[9] The real duration of an inaccessibility incident $t_{e\_ina} = \mathcal{T}_{e\_ina} \, . \, t_{bit}$, where t_{bit} is the nominal bit time.

[10] Other relevant inaccessibility timing parameters, such as t_{ina} and its upper bound T_{ina}, can be constructed from equation (6) by layer management entities [17].

Inaccessibility control in CANELy

In addition, we have to evaluate the duration of the effects of inaccessibility events and incorporate them on (timeout-based) protocol execution. These issues are fundamental.

In a lightly loaded network, one may expect that the retransmission of a frame upon a network error may succeed before the protocols above CAN initiate recovery, due to a timeout. In those cases, no extension of protocol timers is required. Conversely, in a heavily loaded network it may happen that even a protocol timer started after the inaccessibility incident will need to be compensated for the effects of CAN inaccessibility. A (simple) methodology able to treat all these cases in an uniform manner is required.

Inaccessibility Flushing

The method that we call *inaccessibility flushing* is inspired by the CAN inaccessibility trapping algorithm. However, we avoid the "accessibility test" transmissions, that waste network bandwidth (a scarce resource in CAN).

Let us assume that we assign to the accessibility test frame (cf. Figure 6) the "dummy identifier" with the lowest urgency level in the system. This frame would only be transmitted after servicing any other frame transmit request. In addition, let us assume that no node actually submits for transmission the accessibility test frame. A corrective term, t_{c_ina}, equivalent to the $t_{wait \leftarrow trap}$ delay in Figure 6, can still be obtained, provided one have a mechanism able to detect the "lack" of those transmissions. This implies to extend our assumptions concerning the observable behavior of CAN at the PHY-MAC interface:

N4 - *there is a detectable and unique sequence that identifies the absence of frame transmissions in the CAN bus.*

Assumption N4 stipulates that the Ch_{Bidle} signal, given by equation (7), is asserted when the minimum bus idle period that identifies the absence of any frame transmission, has elapsed. The normalized duration of such a period exceeds exactly in one bit-time ($\mathcal{T}_{bit}$) the period corresponding to the normalized duration of the *End-Of-Transmission* sequence ($\mathcal{T}_{EOT}$). Thus, $\mathcal{T}_B = \mathcal{T}_{EOT} + \mathcal{T}_{bit}$.

$$Ch_{Bidle} = \begin{cases} true & \textbf{if } \mathcal{T}(Ch_{Rx} = r) \geq \mathcal{T}_B \\ false & \textbf{if } \mathcal{T}(Ch_{Rx} = r) < \mathcal{T}_B \vee Ch_{Rx} = d \end{cases} \tag{7}$$

An additional signal, Ch_{Ina}, is asserted upon the detection of an inaccessibility event and remains asserted as long as their effects last, being specified by equation:

$$Ch_{Ina} \mapsto \begin{cases} true & \textbf{if } Ch_{Err} \\ false & \textbf{when } Ch_{Bidle} \end{cases} \tag{8}$$

The Ch_{Ina} signal is used to derive from equation (6) the normalized duration of the entire period where the effects of inaccessibility last, $\mathcal{T}_{c\_ina}$, as given by equation[11]:

$$\mathcal{T}_{c\_ina} = \begin{cases} 0 & \textbf{if } (Ch_{SOF} \vee Ch_{Tok-p} \vee Ch_{IFS}) \wedge \neg Ch_{Ina} \\ \mathcal{T}_{c\_ina} + \mathcal{T}_{bit} & \textbf{if } \neg Ch_{EOT} \vee Ch_{Ina} \\ \mathcal{T}_{c\_ina} & \textbf{if } Ch_{Bidle} \end{cases} \tag{9}$$

At this point, we have the means to control inaccessibility at all the relevant levels of the system. For example, a protocol timer should be extended upon a timeout if the Ch_{Ina} signal is asserted,

[11] The duration of $t_{c\_ina} = \mathcal{T}_{c\_ina} \, . \, t_{bit}$, is upper bounded by T_{c_ina}.

waiting for the end of the effects of network inaccessibility on network delays, signaled through the negation of Ch_{Ina}. The details of this timer management procedure can be found in [17].

In addition, we have obtained other important system attribute: the capability of monitoring a relevant set of dependability and timeliness-related parameters, having them available on a system-wide basis.

Comparison of CAN inaccessibility control methods

The main attributes of the different CAN inaccessibility control methods we have been discussing are compared in Figure 7.

Since in CAN settings T_{ina} assumes rather low values (2160 μs @ 1 Mbps), the inaccessibility addition method may be used in application level protocols, leading only to slightly longer waiting periods. The main advantage of this method is its simplicity. On the other hand, at low-level of communication T_{ina} assumes values in the same order of magnitude or even higher than timeouts. The inaccessibility addition method should not be used at this level, to ensure the preservation of protocol timeliness-related parameters.

Attribute		Inac. Control Method		
		Addition	Trapping	Flushing
Complexity		low	fair	high
Bandwidth overheads			•	
Specialized hardware				•
Real timeout value		$T_{td} + T_{ina}$	T_{td}	
Failure detection latency	no inac.	$T_{td} + T_{ina}$	$2.T_{td}$	T_{td}
	inac.	$T_{td} + T_{ina}$	$2.T_{td}+t_{ina}$	$T_{td}+t_{c\_ina}$
Resilience to lack of coverage		none		detects violation of $k, i, T_{td}, T_{ina}, T_{c\_ina}$
Recommendation for usage		application level	none	CANELy low-level protocols

Figure 7: Comparison of CAN inaccessibility control methods

The inaccessibility trapping method performs very poorly in CAN and its use should be avoided at all.

The inaccessibility flushing method exhibits optimum delays with regard to the detection of failures. Being designed at the low-levels of communication, it exhibits the significant advantage of allowing to monitor/detect a potential lack of coverage of the system assumptions and permit management entities to act accordingly (e.g. stop in a fail-safe state).

6 Related Work

Given that network errors due occur, their effects must be taken into account in any realistic analysis of CAN message schedulability guarantees [21]. In recent years, this issue has received considerable attention from a number of authors. For example: the integration of inaccessibility in the response time analysis of the CAN fieldbus is addressed in [14, 15]; preservation of CAN message

timeliness, by aborting late messages is discussed in [2]; a strategy to protect CAN operation against faulty nodes transmitting data/remote frames too often (babbling idiot) is discussed in [3].

7 Conclusions

We addressed a hard but important problem that may hinder the operation of CAN in critical hard real-time settings: controlling its periods of inaccessibility. Whilst mainly relevant to simplex configurations with dual-media, the solution is also applicable to individual replicas of a redundant network.

Given that the operation of CAN can be disturbed by periods of inaccessibility, i.e. by faults that temporarily prevent communication, we have provided the mechanisms to control inaccessibility in CAN-based systems. This implies: ensuring that the number of inaccessibility periods and their duration have a bound; verifying that the bound is in conformity with the service specification; accommodating inaccessibility in protocol execution and timeliness calculations.

Parameter	TTP	CAN Standard Layer	CANELy
Omission handling	masking	detection/recovery	both algorithms
	frame diffusion	frame retransmission	
Inaccessibility duration bound	unknown	2880 *bit-times*	2160 *bit-times*
Inaccessibility control	not completely addressed	no	application level
			low-level protocols
Media redundancy	no	no	yes
Channel redundancy	yes	no	yes (optional)
Babbling idiot avoidance	bus guardian	not provided	can be added
Resilience to lack of coverage	never-give-up strategy	none	detects violation of bounds

Figure 8: Comparison of timeliness-related attributes of TTP, CAN and CANELy

Finally, it is important to mention that this work is a brick in the CANELy architecture, the CAN Enhanced Layer[17], a combination of the CAN standard layer with some simple machinery resources and low-level protocols, described in several publications [23, 20, 16, 18, 19].

Through CANELy we made the proof of concept of the possibility of building CAN-based highly fault-tolerant systems. For example, the main timeliness-related attributes of CANELy and their comparison both with the stand-alone CAN and with the industry standard Time-Triggered Protocol (TTP) architecture, are summarized in Figure 8. We hope these results present a contribution to dismiss ideas that CAN is not suited for hard real-time systems with very high dependability requirements.

References

[1] ANSI/IEEE. *1003.1b-1993 Portable Operating System Interface (POSIX) - Part 1: API C Language - Real-Time Extensions*. IEEE Standard, 1993. ISBN 1-55937-375-X.

[2] I. Broster and A. Burns. Timely use of the CAN protocol in critical hard real-time systems with faults. In *Proceedings of the 13th Euromicro Conference on Real-time Systems*, Delft, The Netherlands, June 2001. IEEE.

[3] I. Broster and A. Burns. An analysable bus-guardian for event-triggered communication. In *Proceedings of the 24th Real-time Systems Symposium*, pages 410–419, Cancun, Mexico, December 2003. IEEE.

[4] J. Charzinski. Performance of the error detection mechanisms in CAN. In *Proceedings of the 1st International CAN Conference*, pages 1.20–1.29, Mainz, Germany, September 1994. CiA.

[5] CiA - CAN in Automation. *CAN Physical Layer for Industrial Applications - CiA Draft Standard 102 Version 2.0*, April 1994.

[6] Dallas Semiconductors. *DS80C390 Dual-CAN High-Speed Microprocessor*, September 1999.

[7] Digital. *Digital Unix - Guide to Realtime Programming*, chapter Clocks and Timers. Digital Equipment Corporation, March 1996.

[8] ISO. *International Standard 11898 - Road vehicles - Interchange of digital information - Controller Area Network (CAN) for high-speed communication*, November 1993.

[9] H. Kopetz. A comparison of CAN and TTP. In *Proceedings of the 15th IFAC Workshop on Distributed Computer Control Systems*, Como, Italy, September 1998. IFAC.

[10] H. Kopetz and G. Grunsteidl. TTP - a protocol for fault-tolerant real-time systems. *IEEE Computer*, 27(1):14–23, January 1994.

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

[12] A Meschi, M. Natale, and M. Spuri. Priority inversition at the network adapter when scheduling messages with earliest deadline techniques. In *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*, pages 243–248, L' Aquila, Italy, June 1996. IEEE.

[13] RED-CAN a fully redundant CAN-system. NOB Elektronik AB Product Note - Sweden, 1998. http://www.nob.se.

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

[15] S. Punnekkat, H. Hansson, and C. Norstrom. Response time analysis under errors for CAN. In *Proceedings of the Real-Time Technology and Applications Symposium*, pages 258–265, Washington, USA, May 2000. IEEE.

[16] L. Rodrigues, M. Guimarães, and J. Rufino. Fault-tolerant clock syncronization in CAN. In *Proceedings of the 19th Real-Time Systems Symposium*, pages 420–429, Madrid, Spain, December 1998. IEEE.

[17] J. Rufino. *Computational System for Real-Time Distributed Control*. PhD thesis, Technical University of Lisbon - Instituto Superior Técnico, Lisboa, Portugal, July 2002.

[18] J. Rufino, P. Veríssimo, and G. Arroz. A Columbus' egg idea for CAN media redundancy. In *Digest of Papers, The 29th International Symposium on Fault-Tolerant Computing Systems*, pages 286–293, Madison, Wisconsin - USA, June 1999. IEEE.

[19] J. Rufino, P. Veríssimo, and G. Arroz. Node failure detection and membership in CANELy. In *Proceedings of the 2003 International Conference on Dependable Systems and Networks*, pages 331–340, San Francisco, California, USA, June 2003. IEEE.

[20] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in CAN. In *Digest of Papers, The 28th International Symposium on Fault-Tolerant Computing Systems*, pages 150–159, Munich, Germany, June 1998. IEEE.

[21] K. Tindell and A. Burns. Guaranteed message latencies for distributed safety-critical hard real-time control networks. Technical Report YCS 229, Real-Time Systems Research Group, University of York, England, September 1994.

[22] P. Veríssimo. Real-time Communication. In S.J. Mullender, editor, *Distributed Systems*, ACM-Press, chapter 17, pages 447–490. Addison-Wesley, 2nd edition, 1993.

[23] P. Veríssimo, J. Rufino, and L. Ming. How hard is hard real-time communication on field-buses? In *Digest of Papers, The 27th International Symposium on Fault-Tolerant Computing Systems*, pages 112–121, Washington - USA, June 1997. IEEE.

[24] P. Veríssimo, J. Rufino, and L. Rodrigues. Enforcing real-time behaviour of LAN-based protocols. In *Proceedings of the 10th IFAC Workshop on Distributed Computer Control Systems*, Semmering, Austria, September 1991. IFAC.

[25] K. Zuberi and K. Shin. Scheduling messages on Controller Area Network for real-time CIM applications. *IEEE Transactions on Robotics and Automation*, 13(2):310–314, April 1997.