Control of Inaccessibility in CANELy

José Rufino
FCUL*
ruf@di.fc.ul.pt

Paulo Veríssimo
FCUL
pjv@di.fc.ul.pt

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

Carlos Almeida
IST-UTL†
cra@comp.ist.utl.pt

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 standard fieldbuses. The analysis and design of such networks w.r.t. timing properties has traditionally 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 functions, as it does in many application settings (e.g. industrial, automotive, avionics, aerospace). As part of our endeavor to design a CAN-based infrastructure capable of extremely reliable communication, dubbed CAN Enhanced Layer (CANELy), this paper provides a detailed analysis of CAN behavior in the presence of inaccessibility, discussing a generic and efficient 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.

Traditional analysis 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 physical faults such as network 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.

There are a number of causes for inaccessibility glitches. In some networks they may be linked to reconfiguration and/or recovery actions: e.g. entry/leave of stations or token loss, in the token-based PROFIBUS [1, 2]. Also, they may have origin in individual failures, such as bit corruption by electromagnetic interference [2, 3].

The consequences of such disturbances on real-time communication is the error they introduce in the time-domain, which may lead: to the violation of pre-specified timing bounds, such as message transmission deadlines; to the failure of protocol or task 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 ultra dependable 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 paper discusses in detail CAN in respect to inaccessibility, including a relevant set of control methodologies. The paper 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 paper, that is, the control of CAN inaccessibility, is discussed in Sections 5 and 6; reference to related work (Section 7) and some final remarks (Section 8) conclude the paper. 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 [4, 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 [6], 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

*Faculdade de Ciências da Universidade de Lisboa, Campo Grande - Bloco C8, 1749-016 Lisboa, Portugal. Tel: +351-217500254 - Fax: +351-217500084. Navigators Home Page: http://www.navigators.di.fc.ul.pt. This work was partially supported by FCT through Project POSC/EIA/56041/2004 (DARIO).

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

[7], 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).

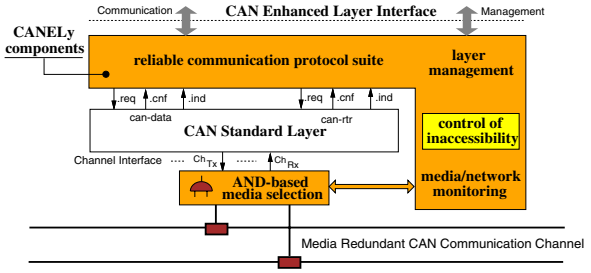

Figure 1. CAN Enhanced Layer architecture

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 [8]; a reliable communication protocol suite, offering a set of broadcast/multicast primitives [9]; clock synchronization [10]; node failure detection and membership services [11].

This paper discusses the mechanisms and the techniques used in CANELy to enforce system correctness in the time-domain despite the occurrence of network errors, that is *control of inaccessibility*. We show that support for *inaccessibility flushing* can be provided effectively in CANELy through an extremely simple method.

CAN Standard Layer

The CAN fieldbus is a multi-master network that uses a twisted pair cable as transmission medium [4, 5]. The network maximum length depends on the data rate. For example: 40m @ 1 Mbps. Bus signaling takes one out of two values: *recessive* (r), also 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. A *data frame* may contain a *message*, a user-level piece of information. A *remote frame* consists of control information only.

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 remote (can-rtr) message; *confirm* to the user a successful message transmission (.cnf); *indicate* a message arrival (.ind).

Basic Dependability of the CAN Protocol

CAN fault-confinement and error detection mechanisms ensure that most failures are perceived consistently by all nodes. Unfortunately, some subtle errors can lead to inconsistency and induce the failure of dependable communication protocols based on CAN operation alone [9]. Inconsistent frame omissions may occur when faults hit the last two bits of a frame at some nodes[1], 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 [9].

Fault-confinement in CAN protocol aims 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.

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 [9, 8, 12].

We introduce the following definition: a component is **weak-fail-silent** if it behaves correctly or crashes if it exhibits more than a given number of omission failures in an interval of reference, the component's *omission degree* [13]. Thus, we define the following failure semantics for **CAN network components**:

- 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[2];
- 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 [8]).

The weak-fail-silent assumption can be enforced with high coverage for the CAN controller by fault confinement mechanisms [9, 12]. 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 [9, 10, 11].

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 [13]. Then, on top of the basic MAC sub-layer functionality, CAN has error-recovery mechanisms defining message-level properties, in some way with 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.

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

[2]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.

CAN MAC and LLC properties

The upper part of Figure 2 enumerates a relevant set of CAN MAC-level properties, defined in previous works [9, 12]. Property MCAN1 formalizes the effect of CAN built-in error detection and signaling mechanisms, and it implies that frame errors are transformed into omissions. Most frame errors are handled consistently by all correct nodes. The residual probability of undetected frame errors is negligible [14]. Property MCAN2 maps the failure semantics introduced earlier 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 [12].

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. In the absence of faults, T_{td} includes the normal queuing, access and transmission delays, and depends on message latency classes and offered load bounds [15, 16, 17]. In general, T_{td} includes also the extra delays resulting from the additional queuing effects caused by the *periods of inaccessibility* [18, 19, 20]. Additionally, the bounded frame transmission delay includes T_{ina}, a corrective term which accounts for the worst-case duration of inaccessibility events, given the bounds specified by property MCAN3. This way, timing failures due to inaccessibility events can be avoided. The inaccessibility characteristics of CAN depend on the network alone and can be predicted by the analysis of the CAN protocol [2].

The LLC level, defines the message-level properties of CAN. 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 [9, 12]. 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 [9, 10, 11].

4. Hard Real-Time Operation of CAN

In conformity 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. In addition, let us assume that the network channel is not replicated, with the possible exception of the physical - medium and electrical signaling - level.

The following conditions must then be observed to secure reliable hard real-time communication in any CAN fieldbus infrastructure, CANELy included [21, 2]:

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. It depends on the offered load bounds, defining the temporal characteristics of message transmit requests and on the determinism of MAC-level mechanisms [12].

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). In CANELy, securing condition RT2 is supported by a simple software layer, built on top of an exposed CAN controller interface [9, 12].

Condition RT3 is related to the maintenance of connectivity between network nodes. Note that partitioning can be tolerated with complete network redundancy, so one might wonder why worry about inaccessibility glitches. At least two reasons make the approach worthwhile. Firstly, network redundancy - replicating both physical and network attachment layers - 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 [22]. Secondly, even with network redundancy, 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 develop in this paper would also be applicable to individual replicas of a redundant network. Achieving RT3 has its complexity, and

requires the utilization of appropriate design techniques [13]. The discussion of how to secure RT3 in CAN-based systems is the central issue of this paper.

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 [21, 13, 2]. This is also true of physical partitioning, if the system has repair (e.g. medium redundancy or reconfiguration) [13]. The problem of inaccessibility was thoroughly equated in previous works on LANs and fieldbuses [21, 2]. Its definition, in [21], 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 (e.g. enforcing the weak-fail-silent assumption). That means, it is necessary to complement the LAN or fieldbus functionality [21], in order to:

- *guarantee recovery from the conditions leading to partition 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, the partitions in our model 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. Controlling Partitions in CAN

This section is entirely dedicated to the discussion of how to control the effects of partitioning 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 design of the CANELy architecture [12].

5.1. Handling CAN Physical Partitions

In [8], we have presented an innovative and extremely simple scheme, depicted in Figure 3, 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.

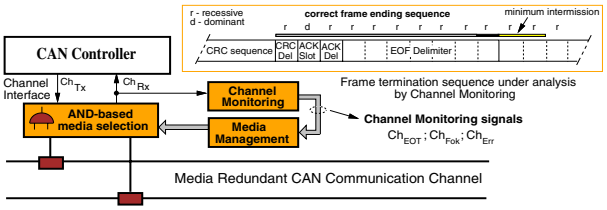

Figure 3. CAN Media Redundancy Mechanisms

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 [8], 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 [4]. It is negated upon the assertion of the Ch_{EOT} signal.

In the context of this paper, one use of these signals is in the evaluation of the real value of the Channel omission degree[3], as specified in equation:

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

where: the notation $\overset{=}{\uparrow_{Ch_{EOT}}}$ means equation (1) is evaluated upon the assertion of the Ch_{EOT} signal. A Channel exceeding the allowed omission degree bound, k, given by property MCAN2, should be considered failed.

[3]The Channel omission degree is the number of consecutive Channel omissions in a time interval of reference, T_{rd} [13]. Its evaluation by equation (1) assumes that $T_{rd} \rightarrow \infty$.

5.2. CAN Accessibility Constraints

The study of CAN accessibility constraints presented in [2] has established analytical expressions for all inaccessibility events, showing that their durations are bounded, and allowing the determination of those bounds.

The CAN built-in error handling mechanisms are responsible for detecting and recovering from a set of single-bit disturbances: bit corruption, bit-stuffing, CRC[4], acknowledgment and form errors, together with reactive overload signaling [2]. The resulting inaccessibility duration bounds, represented in the leftmost part of Figure 4, cannot be reduced by any control strategy because only one frame transfer is affected.

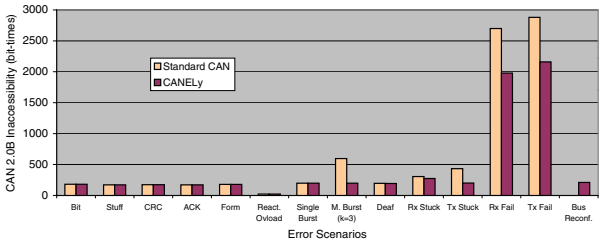

Figure 4. CANELy vs. standard CAN normalized inaccessibility bounds

Conversely, the control actions taken in the CANELy architecture to enforce the weak-fail-silent assumption for the network components [12], have provided an interesting, though moderate, reduction of the inaccessibility worst-case duration bounds (cf. Figure 4), for permanent node failures (deaf receiver, stuck-at and other internal transmitter/receiver failures) [12].

Finally, the CANELy architecture is extremely effective in handling partitioning [12]. It is worth noticing: the effects of *medium quarantine*, in the presence of multiple bit-error bursts (cf. Figure 4); the low worst-case figure of the bus reconfiguration delay (209 μs @ 1Mbps), compared with other failure scenarios and, namely, with the 100 ms of commercial systems currently available [23].

The avoidance of "babbling idiot" failures in CAN settings has further been studied in [24]. The corresponding inaccessibility duration, being much lower than those inscribed in Figure 4 (according to the discussion in [24], it takes only 41 bit-times to detect the babbling node, perform its shutdown and wait for network recovery), is not relevant in the context of our analysis: it does not represent a worst-case bound; babbling idiot failures are not detectable by the native CAN error handling mechanisms; protection to such kind of timeliness-related failures has to be provided by specific machinery (bus guardian) [25, 24].

5.3. Inaccessibility Control Methods

The next step toward 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 [21, 13], 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 [9, 12].

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 [9, 11, 12]. 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).

For the sake of simplicity, let us assume that: T_{td}, represents in general a 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 [26, 27]. 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 derived from LAN-based solutions is to use *inaccessibility trapping* [21, 13]. This method is based on the transformation of inaccessibility periods into omissions.

Protocol timers, such as $T_{waitRemote}$ in the diagram of Figure 5, do not include T_{ina}, being set to an optimum timeout value, defined in function of T_{td} (MCAN4). 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

[4]Cyclic Redundancy Check.

Figure 5). Furthermore, all inaccessibility events occurring between two consecutive network accessibility signals (e.g. situations I2 and I3 in Figure 5) are transformed into *one* "omission". A timer may expire and a recovery process may have to be initiated.

In LAN-based solutions, that recovery process usually involves some sort of sender-initiated retry mechanism (cf. Figure 5). 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).

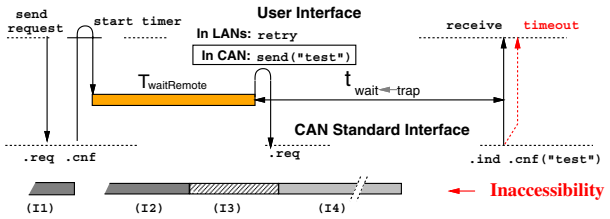

Figure 5. Timing of the CAN inaccessibility trapping method

The frame retransmission scheme of CAN: guarantees the absence of omissions at the LLC level when a frame transmission is confirmed, thus securing $k_{LLC}=0$; makes useless the incorporation of sender-based retry mechanisms on top of CAN [12].

In the CAN-oriented version of the inaccessibility trapping method (Figure 5), 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 (cf. Figure 5). Otherwise, the timer signal is ignored. Again, all the inaccessibility periods occurring inside the interval defined by the accessibility signals implicitly provided by the standard CAN communication primitives (e.g. I3 and I4 in Figure 5), are seen as "one" single event, securing $i_{LLC}=1$.

Each expiring timer is always extended by $t_{wait\leftarrow trap}$ (cf. Figure 5), the duration of the accessibility test, even in the absence of inaccessibility, making inaccessibility trapping in CAN a non-optimal method with respect to: the utilization of CAN bandwidth (a scarce resource in CAN); the delay in the detection of a failure[5].

Are CAN controller error notifications helpful?

Existing CAN controllers may issue an error notification signal, to be delivered to protocol entities above MAC/LLC levels, upon the detection of a disturbance in the transfer of a data/remote frame. One idea would then be the integration of those error notifications with inaccessibility control mechanisms: 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 [28]); the error notification signal may not be delivered at some nodes (property LCAN1). Furthermore, existing CAN controllers are helpless in the provision of an indication of whether or not the effects of inaccessibility last in the system.

6. Inaccessibility Control in CANELy

To control the real effects of inaccessibility on system timeliness one needs special-purpose mechanisms, not available in commercial CAN controllers. Though we do not advocate the use of such solutions as a general rule, in CANELy those mechanisms are not complex to implement, being common, in a great extent, to the bus media redundancy machinery.

One aspect of the problem concerns the evaluation of the real duration of the effects of inaccessibility incidents and its incorporation in the execution of (timeout-based) protocols. These issues are of fundamental importance. For example, 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 based on a strategy that detects when the (distributed) frame transmission queue becomes empty, after the occurrence of an inaccessibility incident [12]. In a sense, the *inaccessibility flushing* method is equivalent to the CAN-oriented inaccessibility trapping algorithm (cf. Figure 5), but it avoids the "accessibility test" transmissions, which waste network bandwidth. Let us assume that we assign to the accessibility test frame a "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. The "lack" of such transmissions can be detected by extremely simple mechanisms.

We made the following operational assumption concerning the observable behavior of CAN at the PHY-MAC interface, as *per* the standard [4]:

[5] Equivalent to absence of acknowledgment after $k+i+1$ tries, given the CAN LLC-level parameters: $k_{LLC}=0$ and $i_{LLC}=1$.

N1 - *there is a detectable and unique sequence that identifies the absence of frame transmissions in the CAN bus.*

Assumption N1 stipulates that the Ch_{Bidle} signal, given by equation (2), is asserted when the minimum bus idle period, which 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, $\mathcal{T}_B = \mathcal{T}_{EOT} + \mathcal{T}_{bit}$.

$$Ch_{Bidle} = \begin{cases} true & \textbf{if } \mathcal{T}_{Bidle} \geq \mathcal{T}_B \\ false & \textbf{if } \mathcal{T}_{Bidle} < \mathcal{T}_B \vee Ch_{Rx} = d \end{cases} \quad (2)$$

where: $\mathcal{T}_{Bidle} = \mathcal{T}(Ch_{Rx} = r)$ describes the time elapsed since the Channel is in the idle state.

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:

$$Ch_{Ina} \mapsto \begin{cases} true & \textbf{if } Ch_{Err} \\ false & \textbf{when } Ch_{Bidle} \end{cases} \quad (3)$$

The Ch_{Bidle} signal may also be used, together with the Channel monitoring signals, Ch_{Err} and Ch_{EOT}, to evaluate the number of inaccessibility incidents in a relevant period of reference (i.e. while the effects of inaccessibility last), as specified in equation:

$$Ch_{Ii} \overset{=}{\uparrow_{Ch_{EOT}}} \begin{cases} Ch_{Ii} + 1 & \textbf{if } Ch_{Err} \\ 0 & \textbf{when } Ch_{Bidle} \end{cases} \quad (4)$$

where: the notation $\overset{=}{\uparrow_{Ch_{EOT}}}$, in equation (4), means evaluation of the **if** clause upon the assertion of the Ch_{EOT} signal. A Channel exceeding the bound, i, specified by property MCAN3 with respect to the number of inaccessibility incidents, should be considered failed.

At this stage, we have the fundamental means to control inaccessibility even at the lower levels of the system. For example, a (low-level) protocol timer should be extended upon a timeout if the Ch_{Ina} signal is asserted, 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 [12].

In addition, to assess the exact duration of inaccessibility periods, we have to extend further our assumptions concerning the observable behavior of CAN at the PHY-MAC interface. The key points are: in a frame transfer disturbed by errors, a non-negligible time interval may exist between the beginning of the period of inaccessibility and the start of the, usually shorter, period of error recovery[6]; both periods need to be accounted for to obtain the total duration of the inaccessibility event. Thus:

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

N2 - *there is a detectable sequence that identifies the beginning of a possibly correct CAN data or remote frame transmission.*

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

N4 - *there is a detectable and unique fixed form sequence that identifies the correct termination of a CAN data or remote frame.*

Assumption N2 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 (5).

$$Ch_{SOF} \mapsto \begin{cases} true & \textbf{if } Ch_{EOT} \wedge Ch_{Rx} = d \\ false & \textbf{when } Ch_{SOF} \end{cases} \quad (5)$$

Assumptions N3 and N4 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-P} and Ch_{IFS-P} signals, specified by equations (6) and (7), are asserted (during one bit-time) only if no violation in the termination sequence of a data/remote frame occurs up to the first/second bit of the *intermission*, respectively.

$$Ch_{Tok-P} \mapsto \begin{cases} true & \textbf{if } Ch_{Rx} = rdrrrrrrrrr \\ false & \textbf{when } Ch_{Tok-P} \end{cases} \quad (6)$$

$$Ch_{IFS-P} \mapsto \begin{cases} true & \textbf{if } Ch_{Rx} = rdrrrrrrrrrr \\ false & \textbf{when } Ch_{IFS-P} \end{cases} \quad (7)$$

The auxiliary function, Ch_{Fc}, defined in equation (8), specifies the signaling of a relevant set of frame-level correctness boundaries: the start of a frame transmission; the correct ending of a data/remote frame transmission; the separation of consecutive data/remote frame transmissions by the *minimum intermission* period.

$$Ch_{Fc} = Ch_{SOF} \vee Ch_{Tok-P} \vee Ch_{IFS-P} \quad (8)$$

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 (cf. property LCAN1, in Figure 2); the transmission of a data/remote frame is *a priori* considered a period of inaccessibility.

Thus, the use of Ch_{Fc} in equation (9) specifies that accounting for a possible period of inaccessibility: is started when the Ch_{SOF} signal is asserted; it is restarted if the transmission of a data/remote frame succeeds and, again, when the *minimum intermission* period has elapsed.

The value of $\mathcal{T}_{e\_ina}$, the normalized duration of one inaccessibility event[7], accounted for in equation (9): is incremented at each nominal bit time, while a frame is being transmitted; it maintains the accumulated value after the end of a successful transmission of the frame, signaled through the assertion of the Ch_{EOT} signal, and before the start of a new frame transmission, signaled through the assertion of both Ch_{SOF} and Ch_{Fc} signals.

$$\mathcal{T}_{e\_ina} = \begin{cases} 0 & \textbf{if } Ch_{Fc} \\ \mathcal{T}_{e\_ina} + \mathcal{T}_{bit} & \textbf{if } \neg Ch_{EOT} \\ \mathcal{T}_{e\_ina} & \textbf{if } Ch_{EOT} \end{cases} \quad (9)$$

where: $\mathcal{T}_{bit}$, is the normalized duration of a bit.

The Ch_{Bidle} and the Ch_{Ina} signals are now used to derive from equation (9) the normalized duration of the entire period where the effects of inaccessibility last, $\mathcal{T}_{p\_ina}$, as given by equation:

$$\mathcal{T}_{p\_ina} = \begin{cases} 0 & \textbf{if } Ch_{Fc} \wedge \neg Ch_{Ina} \\ \mathcal{T}_{p\_ina} + \mathcal{T}_{bit} & \textbf{if } \neg Ch_{EOT} \vee Ch_{Ina} \\ \mathcal{T}_{p\_ina} & \textbf{if } Ch_{Bidle} \end{cases} \quad (10)$$

The real duration of the entire period where the effects of the inaccessibility last, $t_{p\_ina} = \mathcal{T}_{p\_ina} \cdot t_{bit}$, is upper bounded by T_{p_ina}.

At this point, we have succeeded in obtaining the means to assess CAN operation also with respect to inaccessibility timing parameters, such as t_{ina}[8], which can be constructed from equation (9) by layer management entities [12]. That means, enforcing correctness of CAN operation in the time-domain: assumes omission errors in CAN are transformed into periods of inaccessibility; implies worst-case message transmission delays include terms accounting for the effects of the periods of inaccessibility (property MCAN4, in Figure 2); allow omission failures to be inconsistent (property LCAN1, in Figure 2). This has been applied with success in the design of the CANELy reliable hard real-time protocol suite [9, 10, 11].

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 6.

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 detect latency	no inac.	$T_{td} + T_{ina}$	$2.T_{td}$	T_{td}
	inac.	$T_{td} + T_{ina}$	$2.T_{td}+t_{ina}$	$T_{td}+t_{p\_ina}$
Resilience to lack of coverage		none		detects violation: k, i, T_{td} T_{ina}, T_{p_ina}
Recommendation for usage		application level	none	CANELy low level protocols

Figure 6. Comparison of CAN inaccessibility control methods

The inaccessibility trapping method performs 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).

Implementation issues

The CANELy architecture, including the specialized mechanisms for the control of inaccessibility in the CAN-based infrastructure can be implemented, in a cost-effective way, using commercial off-the-shelf components, through the integration of a comprehensive set of (simple) machinery resources and specific low-level protocols. The overall architecture of CANELy hardware components is depicted in Figure 7 and comprises the CANELy processing infrastructure and the specialized support for low-level machinery functions.

In the CANELy prototype currently being implemented, the CANELy processing infrastructure required for the execution of CANELy low-level protocols (group communication [9], clock synchronization [10], node failure detection and site membership [11]) is materialized using the state-of-the-art Dallas/Maxim DS80C390 High-Speed Microprocessor [28].

On the other hand, the support for the low-level special-purpose functions is implemented by a single, medium capacity, programmable logic device (Field Programmable Gate Array - FPGA) [29], which comprises the machinery required for:

- the implementation of the bus media redundancy mechanisms, described in [8];

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

[8] The value of t_{ina} is upper bounded by T_{ina}.

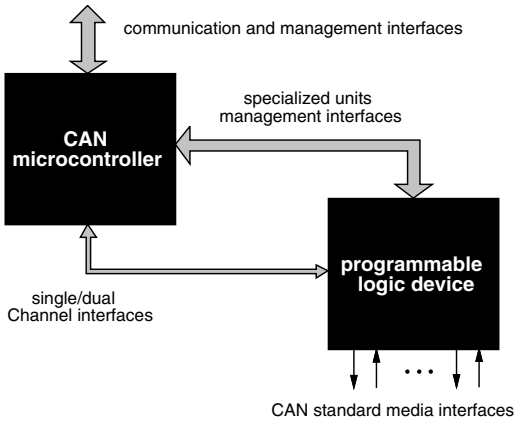

Figure 7. CANELy Hardware Architecture

- the inaccessibility flushing mechanisms, aiming to secure system correctness in the time-domain, despite the occurrence of network errors.

A VHDL-based [30] design of these mechanisms is being specified for the CANELy architecture. The FPGA device interfaces the bus transmission media, through standard CAN media interfacing components, and the CANELy Processing Infrastructure, through the Channel and the network management interfaces (cf. Figure 7).

7. Related Work

Given that network errors do occur, their effects must be taken into account in any realistic analysis of CAN message schedulability guarantees [31]. 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 [18, 19]; preservation of CAN message timeliness, by aborting late messages is discussed in [32]; a probabilistic analysis of CAN communications in the presence of faults, such as those with origin in electromagnetic interference, is addressed in [20, 33]; specific mechanisms to the experimental evaluation of CAN in the presence of faults have been proposed in [3]; a strategy to protect CAN operation against faulty nodes transmitting data/remote frames too often (babbling idiot) is discussed in [24].

The study of message schedulability guarantees, to be performed in the context of the CANELy architecture: assumes known traffic patterns and offered load bounds; makes use of both error free and worst-case error analysis. Error free analysis [15, 16, 17] is useful for optimal system configuration. Analysis of message schedulability guarantees, given a worst-case pattern of inaccessibility incidents [31, 18, 19], defines an upper bound for message delivery delays.

8. 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	CANELy
Omission handling	masking	detection/ recovery	both algorithms
	diffusion	retransmission	
Inaccessibility duration	unknown	14 - 2880 *bit-times*	14 - 2160 *bit-times*
Inaccessibility control	not completely addressed	no	application
			low-level
Media redundancy	no	no	yes
Channel redundancy	yes	no	yes (optional)
Babbling idiot avoidance	bus guardian	not provided	can be added (cf. [24])
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[12], a combination of the CAN standard layer with some simple machinery resources and low-level protocols, described in several publications [2, 9, 10, 8, 11]. 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 designing hard real-time systems with very high dependability requirements.

References

[1] "PROFIBUS: Technology and Application", PROFIBUS Trade Organization, Karlsruhe, Germany, Oct. 2002.

[2] 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, June 1997, pp. 112–121, Washington - USA. IEEE.

[3] J. Ferreira, A. Oliveira, P. Fonseca, and J. Fonseca, "An Experiment to Assess Bit Error Rate in CAN", in *Proceedings of the 3rd. International Workshop on Real-Time Networks*, June 2004, Catania, Italy.

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

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

[6] H. Kopetz, "A Comparison of CAN and TTP", in *Proceedings of the 15th IFAC Workshop on Distributed Computer Control Systems*, Sept. 1998, Como, Italy. IFAC.

[7] H. Kopetz and G. Grunsteidl, "TTP - A Protocol for Fault-Tolerant Real-Time Systems", *IEEE Computer*, vol. 27, no. 1, pp. 14–23, Jan. 1994.

[8] 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*, June 1999, pp. 286–293, Madison, Wisconsin - USA. IEEE.

[9] 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*, June 1998, pp. 150–159, Munich, Germany. IEEE.

[10] L. Rodrigues, M. Guimarães, and J. Rufino, "Fault-Tolerant Clock Syncronization in CAN", in *Proceedings of the 19th Real-Time Systems Symposium*, Dec. 1998, pp. 420–429, Madrid, Spain. IEEE.

[11] 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*, June 2003, pp. 331–340, San Francisco, California, USA. IEEE.

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

[13] P. Veríssimo, "Real-Time Communication", in S. Mullender, editor, *Distributed Systems*, ACM-Press, chapter 17, pp. 447–490. Addison-Wesley, 2nd edition, 1993.

[14] J. Charzinski, "Performance of the Error Detection Mechanisms in CAN", in *Proceedings of the 1st International CAN Conference*, Sept. 1994, pp. 1.20–1.29, Mainz, Germany. CiA.

[15] K. Tindell and A. Burns, "Guaranteeing Message Latencies on Controler Area Network", in *Proceedings of the 1st International CAN Conference*, Sept. 1994, pp. 1.2–1.11, Mainz, Germany. CiA.

[16] K. Zuberi and K. Shin, "Scheduling messages on Controller Area Network for real-time CIM applications", *IEEE Transactions on Robotics and Automation*, vol. 13, no. 2, pp. 310–314, Apr. 1997.

[17] M. Livani, J. Kaiser, and W. 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*, June 1998, Shantou, China. IFAC/IFIP.

[18] 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*, Sept. 2000, pp. 77–84, Porto, Portugal. IEEE.

[19] S. Punnekkat, H. Hansson, and C. Norstrom, "Response Time Analysis under Errors for CAN", in *Proceedings of the Real-Time Technology and Applications Symposium*, May 2000, pp. 258–265, Washington, USA. IEEE.

[20] I. Broster, A. Burns, and G. Rodríguez-Navas, "Probabilistic Analysis of CAN with Faults", in *In Proceedings of the 23rd Real-time Systems Symposium*, Dec. 2002, Austin, Texas. IEEE.

[21] 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*, Sept. 1991, Semmering, Austria. IFAC.

[22] D. E. Powell, *Delta-4 - A Generic Architecture for Dependable Distributed Computing*, ESPRIT Research Reports. Springer Verlag, Nov. 1991.

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

[24] I. Broster and A. Burns, "An Analysable Bus-Guardian for Event-Triggered Communication", in *Proceedings of the 24th Real-time Systems Symposium*, Dec. 2003, pp. 410–419, Cancun, Mexico. IEEE.

[25] K. Tindell and H. Hansson, "Babbling Idiots, the Dual-Priority Protocol and Smart CAN Controllers", in *Proceedings of the 2nd International CAN Conference*, Oct. 1995, pp. 7.22–7.28, London, England. CiA.

[26] 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.

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

[28] Maxim/Dallas Semiconductors, *DS80C390 Dual-CAN High-Speed Microprocessor*, Feb. 2005.

[29] Xilinx, *Spartan and Spartan-XL Families Field Programmable Gate Arrays Data Sheet*, June 2002.

[30] P. Ashenden, "The VHDL Cookbook", Department Computer Science, University of Adelaide, South Australia, July 1990, Available from the following URL: ftp://chook.cs.adelaide.edu.au/pub/VHDL-Cookbook.

[31] 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, Sept. 1994.

[32] 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*, June 2001, Delft, The Netherlands. IEEE.

[33] I. Broster, A. Burns, and G. Rodríguez-Navas, "Timing Analysis of Real-Time Communication Under Electromagnetic Interference", *Real-Time Systems*, vol. 30, no. 1-2, pp. 55–81, May 2005.