Integrating Inaccessibility Control and Timer Management in CANELy

José Rufino
FCUL*
ruf@di.fc.ul.pt

Paulo Veríssimo
FCUL
pjv@di.fc.ul.pt

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

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

Abstract

The CAN Enhanced Layer (CANELy) is a CAN-based infrastructure capable of extremely reliable communication. This paper describes the mechanisms and the techniques used in CANELy to enforce system correctness in the time-domain despite the occurrence of network errors (inaccessibility). The paper discusses how to integrate in the existing CANELy machinery, the control of inaccessibility and the management of timers, at several levels of the system. In particular, application and low-level protocol layers are addressed. In addition, a relevant set of parameters are available for system monitoring, allowing the validation/enforcement of the system model.

1. Introduction

Fieldbus technologies, such as the Controller Area Network (CAN), play nowadays a fundamental role in the design and implementation of industrial communication systems and 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.

Therefore, they must ensure strict guarantees in regard to continuity of service and known and bounded message delivery latency. However, even if one excludes physical faults such as network partitioning, industrial fieldbuses are subject to periods of *inaccessibility*. They derive from incidents in the network operation that temporarily prevent communication and whose effect is to increase the network access delay as seen by one or more nodes.

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 industrial fieldbuses in a very comprehensive way.

This paper discusses in detail how CANELy integrates inaccessibility control and timer management. The paper is organized as follows: for completeness, Section 2 reviews the CANELy architecture and Section 3 presents the system model; the mechanisms being specified in CANELy to handle physical and virtual network partitioning are presented in Sections 4 and 5; integration of timer management functions is addressed in Section 6 and some final remarks in Section 7 conclude the paper. The following discussion assumes the reader to be fairly familiar with CAN operation [1, 2].

2. CANELy: Architecture Overview

The CAN fieldbus has a set of limitations in respect to the provision of strict availability, reliability and timeliness attributes [3]. However, 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 original Time-Triggered Protocol [4], or other CAN-based time-triggered variants, such as TTCAN [5] or FlexCAN [6], was indeed a set of fault tolerance and timeliness-related services. Moreover, these services 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).

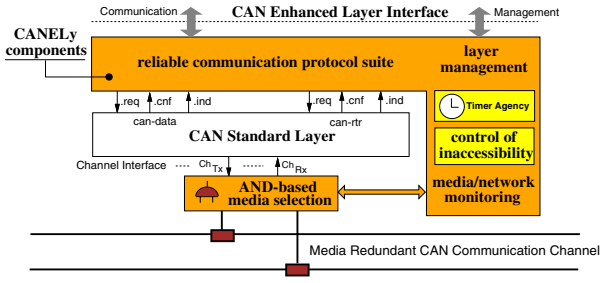


Figure 1. CAN Enhanced Layer architecture

The central component of the CANELy architecture (Figure 1) is naturally the standard CAN layer, comple-

*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.

mented/enhanced with some simple machinery and low-level protocols, which include: a network infrastructure resilient to physical partitioning [7]; a reliable communication protocol suite, offering a set of broadcast/multicast primitives [8]; clock synchronization [9]; node failure detection and membership services [10].

This paper discusses how the control of inaccessibility, addressed in [11], and the management of timers should be integrated in the CANELy architecture, at the different levels of the system.

CAN Standard Layer

The CAN fieldbus is a multi-master network that uses a twisted pair cable as transmission medium [1, 2]. The network maximum length depends on the data rate. A typical value is: 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. The uniqueness of frame identifiers, is exploited for bus arbitration. A *carrier sense multi-access with deterministic collision resolution* policy is used: the node transmitting the frame with the lowest identifier always wins arbitration. 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 control (can-rtr) messages[1]; *confirm* to the user a successful message transmission (.cnf); *indicate* a message arrival (.ind).

3. System Model

This section enumerates the fault assumptions for the system and discuss CAN protocol properties [8, 12].

We define: 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]. The following failure semantics are defined 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 [7]).

The weak-fail-silent assumption can be enforced with high coverage by fault confinement mechanisms [8, 12]. This is crucial for the preservation of CAN timeliness and for the parameterization of protocols operating on top of CAN, such as those defined in CANELy [8, 9, 10].

CAN MAC and LLC Properties

The upper part of Figure 2 enumerates a relevant set of CAN MAC-level properties [8, 12]. Property MCAN1 formalizes CAN built-in error handling, and it implies that frame errors are transformed into omissions. The residual probability of undetected frame errors is negligible [14]. Property MCAN2 maps the given failure semantics onto the operational assumptions of CAN, being $k \geq f_o$.

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

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} also includes the extra delays resulting from the additional queuing effects caused by the *periods of inaccessibility* [18, 19, 20]. The bounded frame transmission delay includes T_{ina}, a corrective term that accounts for the worst-case duration of inaccessibility events, given the bounds specified by property MCAN3. The inaccessibility characteristics of CAN are obtained by analysis of the CAN protocol [21].

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[3] implies that some j of the k omissions will show at the LLC interface as inconsistent omissions [8, 12].

[1] In the CANELy architecture, control messages are usually encapsulated in remote frames.

[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.

[3] Examples of causes for inconsistent bit-error detection are: electromagnetic interference or deficient receiver circuitry.

4. Handling Partitioning in CANELy

In CANELy [12], maintaining the connectivity between network nodes is achieved under a general inaccessibility model, thoroughly discussed in [11], and reviewed in this section for completeness.

The study of CAN accessibility constraints in [21, 12] has established analytical expressions for the inaccessibility periods, showing that their durations are bounded, and allowing the determination of those bounds.

The CAN built-in error handling functions provide: error detection; global signaling of errors, through the transmission of *error frames*; recovery from error situations, by allowing a minimum period of bus idleness.

The duration of inaccessibility events for a comprehensive set of single-bit errors (bit, bit-stuffing, CRC[4], acknowledgment and form errors; reactive overload conditions) is illustrated in the leftmost part of Figure 3. No inaccessibility control strategy can reduce such inaccessibility bounds, because none of those errors affect more than one frame transfer [11].

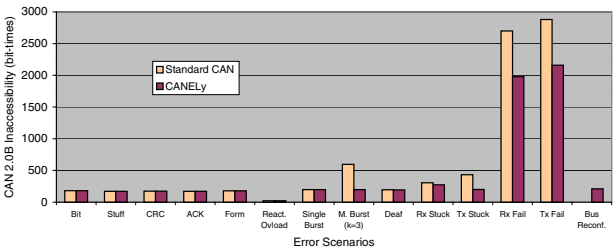


Figure 3. CANELy vs. standard CAN normalized inaccessibility bounds

The handling of multiple network errors by CAN fault confinement mechanisms is based on two counters recording, at each node, transmit and receive errors. In the presence of such *omission errors*, the error counters are updated, according to rules that make faulty nodes experience, with a very high probability, the highest error counter increase [22, 1, 23].

The CANELy architecture exploits these mechanisms to enforce the weak-fail-silent assumption for the network components: a node exceeding a pre-specified number of omission errors, should be shut-down, by forcing it to enter the *bus-off* state [1, 8, 12].

This inaccessibility control strategy: allows an interesting, though moderate, reduction of the inaccessibility durations for permanent node failures (deaf receiver, stuck-at and other internal transmitter/receiver failures), as studied in [12, 11] and illustrated in Figure 3; is an action that can be easily implemented on most of existing CAN controllers (e.g. [24]), which are able to issue a warning signal if any error counter exceeds a given threshold [22, 1].

In Figure 3 are also shown the effects of the innovative mechanisms introduced in [7] for handling CAN physical partitions: a *medium quarantine* scheme, allowing the reduction of the inaccessibility upper bound for bus multiple bit-error bursts; bus reconfiguration in event of medium failure, exhibiting a worst-case delay bound (209 μs @ 1Mbps) that should be compared with other failure scenarios and with the 100 ms of commercial systems currently available [25].

The avoidance of "babbling idiot" failures in CAN has further been studied [26]: it may take only 41 bit-times to detect the babbling node, perform its shutdown and wait for network recovery (a result not included in the diagram of Figure 3). Protection to such kind of timeliness-related failures has to be provided by specific machinery (bus guardian) [27, 26].

[4]Cyclic Redundancy Check.

5. Controlling Inaccessibility in CANELy

In a recent paper [11], we have analyzed a comprehensive set of alternatives for controlling inaccessibility in CAN-based systems. This section exploits the main results of [12, 11] to: provide a detailed analysis of how to assess the real values of a comprehensive set of dependability and timeliness-related parameters, having them available for monitoring on a system-wide basis; discuss why the assessment of those parameters needs to be integrated with inaccessibility control mechanisms.

5.1. Extended Channel Monitoring

The assessment of CAN with regard to inaccessibility calls for a *Channel Monitoring* module providing an extended set of frame-wise signals, with a specification drawn from previous works on CANELy [7, 12, 11] and summarized in Figure 4.

5.2. Inaccessibility Faults Assessment

The signals defined in Figure 4 are used in CANELy to assess CAN inaccessibility incidents with regard to the number of events and their duration.

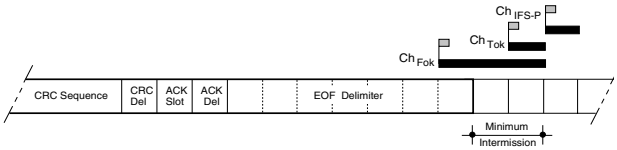
NUMBER OF INACCESSIBILITY EVENTS

The Channel monitoring signals, Ch_{Fok} and Ch_{Err} (cf. Figure 4), are used in the general definition of equations (1) and (2), to account for: Ch_{Ie}, the total number of inaccessibility incidents; Ch_{Oe}, the number of those events strictly due to Channel omission errors. The difference between the Ch_{Ie} and the Ch_{Oe} counters provides the number of inaccessibility incidents strictly due to overload conditions.

$$Ch_{Ie} \stackrel{=}{\uparrow_{Ch_{EOT}}} \begin{cases} 0 & \textbf{when } \textit{can-icu.req} \\ Ch_{Ie}+1 & \textbf{if } Ch_{Err} \end{cases} \quad (1)$$

$$Ch_{Oe} \stackrel{=}{\uparrow_{Ch_{EOT}}} \begin{cases} 0 & \textbf{when } \textit{can-icu.req} \\ Ch_{Oe}+1 & \textbf{if } Ch_{Err} \wedge \neg Ch_{Fok} \end{cases} \quad (2)$$

where: the notation $\stackrel{=}{\uparrow_{Ch_{EOT}}}$, means the **if** clause in equations (1) and (2) is evaluated upon the assertion of the



Ch_{SOF}	**Start Of Frame** asserted at beginning of frame transmission; one bit-time duration.
Ch_{Fok}	**Frame Correct** data or remote frame received without errors; negated upon assertion of Ch_{EOT}.
Ch_{Tok}	**Transmission Correct** no frame format error up to first bit of *intermission*; negated upon assertion of Ch_{EOT}. Ch_{Tok-P}, companion one bit-time duration pulse.
Ch_{IFS-P}	**Frame Termination Correct** no frame format error up to second bit of *intermission*; one bit-time duration.
Ch_{Err}	**Frame Error** asserted upon violation of CAN bit-stuffing rule; negated upon assertion of Ch_{EOT}.
Ch_{EOT}	**End Of Transmission** asserted after detection of minimum bus idle period; negated upon assertion of Ch_{SOF}.

Figure 4. Signals and relevant timing of Channel monitoring functions in CANELy

Ch_{EOT} signal. The Ch_{Ie} and the Ch_{Oe} counters are monotonically incremented, being cleared only through the issuing of a specific layer management action request for the inaccessibility control unit (*can-icu.req*).

DURATION OF INACCESSIBILITY PERIODS

Accounting for the duration of the periods of inaccessibility in CAN calls for a set of extra mechanisms, not available in commercial CAN controllers [12, 11].

One aspect of the problem is illustrated in Figure 5, where the periods of CAN inaccessibility are discriminated in their different components, for single error scenarios. For most of those errors, there may be a non-negligible time interval between the beginning of the period of inaccessibility and the start of error recovery[5], a period which is in general much shorter than the total duration of the inaccessibility event, as shown in Figure 5. This may be caused by a high latency in the detection of the error (e.g. CRC error) or because the error occurs only near the end of the frame transmission (e.g. form error).

Thus, to evaluate the duration of a CAN inaccessibility incident, one needs to mark when a period of inaccessibility begins and how long it lasts. That means, to accurately account for the duration of each CAN inaccessibility event, the transmission of a data/remote frame should be considered, *a priori*, as a period of inaccessibility. Should the frame transfer end without errors, no inaccessibility period is accounted for. Otherwise, the time

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

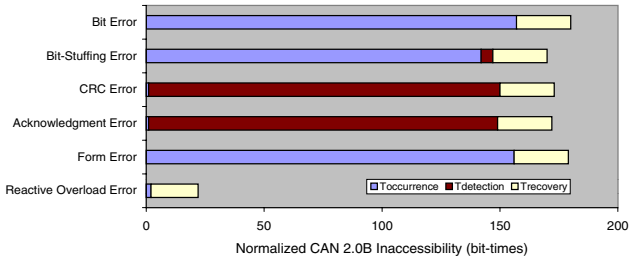


Figure 5. Breaking down worst-case inaccessibility times of single error scenarios

interval between the start of the frame and the first incorrect bit may need to be accounted for as part of the period of inaccessibility.

The analysis of CAN operation with regard to the transfer of a data/remote frame is detailed in Figure 6, where for each type of error, it is given: the action taken by the CAN controller, with regard to frame handling; the effective duration of the inaccessibility event. A conservative approach is followed to handle the subtle difference between a transmitter *omission error* and a receiver *early reactive overload* condition: on account of a possible inconsistency in the detection of the error (property LCAN1, in Figure 2), it is assumed the scheduling of the frame for retransmission in both cases. A similar approach is taken in handling *early/simple reactive overload* conditions. The guarantee that the frame will not be scheduled for retransmission can be achieved only if no error is detected before the second bit of intermission.

In the context of Figure 6, the auxiliary function, Ch_{Fc}, defined in equation (3), 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.

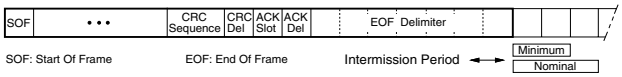
$$Ch_{Fc} = Ch_{SOF} \vee Ch_{Tok-P} \vee Ch_{IFS-P} \qquad (3)$$

The value of $\mathcal{T}_{e\_ina}$, the normalized duration of one inaccessibility event[6], is accounted for in expression (4):

$$\mathcal{T}_{e\_ina} = \begin{cases} 0 & \textbf{if } Ch_{Fc} \\ \mathcal{T}_{e\_ina} + \mathcal{T}_{bit} & \textbf{if } \neg Ch_{EOT} \\ \mathcal{T}_{e\_ina} & \textbf{if } Ch_{EOT} \end{cases} \qquad (4)$$

The value of $\mathcal{T}_{e\_ina}$ given by equation (4): is reset on each frame-level correctness boundary, as defined by Ch_{Fc}; is incremented at each nominal bit time, while a frame is being transmitted; maintains the accumulated value, after the end of a successful frame transmission, signaled through the assertion of the Ch_{EOT} signal, and

[6]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.



Frame-level Error Analysis	
Omission Error	
First incorrect bit:	up to the sixth bit of EOF (any node)
	seventh bit of EOF (transmitter)
Action on frame:	frame discarded
	scheduled for retransmission
Inaccessibility period:	*start-of-frame* $\rightarrow$ *assertion of* Ch_{EOT}
Early Reactive Overload	
First incorrect bit:	seventh bit of EOF (receiver)
Action on frame:	frame accepted
	transmitter may schedule for retransmission
Inaccessibility period:	*start-of-frame* $\rightarrow$ *assertion of* Ch_{EOT}
Simple Reactive Overload	
First incorrect bit:	first bit of intermission
Action on frame:	frame accepted
	transmitter may schedule for retransmission
Inaccessibility period:	*start-of-frame* $\rightarrow$ *assertion of* Ch_{EOT}
Late Reactive Overload	
First incorrect bit:	second bit of intermission
Action on frame:	frame accepted
	no retransmission
Inaccessibility period:	*assertion* Ch_{Tok-P} $\rightarrow$ *assertion of* Ch_{EOT}
No Error	
First incorrect bit:	none
Action on frame:	frame accepted
	no retransmission
Inaccessibility period:	no inaccessibility

Figure 6. Analysis of CAN inaccessibility during a data/remote frame transfer

before the start of a new frame transmission, signaled through the assertion of both Ch_{SOF} and Ch_{Fc} signals; $\mathcal{T}_{bit}$, is the normalized duration of a bit.

PERIOD OF ASSESSMENT

The assessment of the properties defined in Figure 2 should be performed during a time interval, T_{rd}, which: is relevant in the context of each given protocol [8, 9, 10]; may include one or more isolated inaccessibility incidents of individual duration given by equation (4); should include the entire period where the effects of inaccessibility last, thus calling for the use of the specific mechanisms introduced in [12, 11] to control inaccessibility in CANELy.

5.3. Control of Inaccessibility in CANELy

The occurrence of a period of inaccessibility prevents communication between some or all nodes, which may lead to the increase of the message queuing and network access delays, as seen by one or more nodes. Such an ill-effect may persist even after the CAN protocol has recovered from the inaccessibility incident, because any delay may, at least partially, propagate from message to message, in a given queue.

INACCESSIBILITY FLUSHING

An effective methodology to control inaccessibility in the CANELy architecture is based on an *inaccessibility flushing* strategy that detects when the (distributed) frame transmission queue becomes empty, after the occurrence of an inaccessibility event [12, 11]. If a queue has no queued messages, the delays induced at message/frame level by the occurrence of a period of inaccessibility cannot propagate anymore, being thus *flushed* from network operation. Given the observable behavior of CAN at the PHY-MAC interface, as defined in [12, 11]:

- Ch_{Bidle}, is a simple *inaccessibility flushing* signal asserted when the minimum bus idle period that identifies the absence of any frame transmission has elapsed. The signal is negated upon the reception of a *dominant* bus value.

The assertion of the Ch_{Bidle} signal defines the end of the period where the transmission of data/remote frames may be delayed by the effects of inaccessibility. We call this extended period, an *inaccessibility epoch*.

A companion signal, Ch_{Ina}, defines when an inaccessibility epoch begins and for how long it lasts. The Ch_{Ina} signal is asserted upon the detection of an inaccessibility event and negated upon the assertion of the Ch_{Bidle} signal, as specified by equation:

$$Ch_{Ina} \mapsto \begin{cases} true & \textbf{if } Ch_{Err} \\ false & \textbf{when } Ch_{Bidle} \end{cases} \quad (5)$$

The effective duration of an *inaccessibility epoch*[7], $\mathcal{T}_{p\_ina}$, is derived from equations (4) and (5), and accounts for the entire period where the effects of inaccessibility last, being given by equation:

$$\mathcal{T}_{p\_ina} = \begin{cases} 0 & \textbf{if } Ch_{Fc} \wedge \neg Ch_{Ina} \\ \mathcal{T}_{p\_ina} + \mathcal{T}_{bit} & \textbf{if } \neg Ch_{EOT} \vee Ch_{Ina} \\ \mathcal{T}_{p\_ina} & \textbf{if } Ch_{Bidle} \end{cases} \quad (6)$$

The total duration of the periods of inaccessibility within an *inaccessibility epoch*[8], $\mathcal{T}_{ina}$, is obtained by adding the value provided by equation (4) at the end of each period of network activity, signaled through the assertion of the Ch_{EOT} signal, as specified in equation:

$$\mathcal{T}_{ina} \overset{=}{\uparrow_{Ch_{EOT}}} \begin{cases} 0 & \textbf{when } \overset{can-icu.req}{\uparrow Ch_{Bidle}} \\ \mathcal{T}_{ina} + \mathcal{T}_{ina\_e} & \textbf{if } Ch_{Ina} \\ \mathcal{T}_{ina} & \textbf{if } Ch_{Bidle} \end{cases} \quad (7)$$

For simplicity of exposition, we have discussed the use of a single inaccessibility flushing event, provided by the $\overline{Ch_{Bidle}}$ signal. However, an optimized implementation may provide such indications for each network traffic access class or even on a message-by-message basis. This is not complex to support in CANELy.

[7] The duration of $t_{p\_ina} = \mathcal{T}_{p\_ina} \cdot t_{bit}$, is upper bounded by T_{p_ina}.
[8] The duration of $t_{ina} = \mathcal{T}_{ina} \cdot t_{bit}$, is upper bounded by T_{ina}.

The assessment of CAN inaccessibility incidents by CANELy specific mechanisms has been centered on the operation of the CAN native protocol, making parameter evaluation independent of each protocol defined above the standard CAN interface (e.g. [8, 9, 10]).

Fundamental parameters, such as t_{p_ina}, the effective duration of an *inaccessibility epoch*, or t_{ina}, the total duration of inaccessibility in an epoch, as weel as its relation to T_{rd}, have been identified of great relevance to: the accommodation of inaccessibility in protocol execution and timeliness calculations; the dimensioning of timeouts.

5.4. System-level Monitoring

The accurate assessment of the real values for a comprehensive set of dependability and timeliness-related parameters, having them available on system-wide basis, opens room for the design of additional dependability enforcement and system validation mechanisms.

The values of Ch_{Od} and Ch_{Ii}, inscribed in Figure 7, are obtained directly from equations (2) and (1) making the issuing of the *can-icu.req* layer management action equivalent to the assertion of the Ch_{Fok} and Ch_{Bidle} signals, respectively. Conversely, the values of t_{ina} and t_{p_ina}, also in Figure 7, are derived respectively from equations (7) and (6), through bit-time denormalization.

Dependability and timeliness-related parameters		
Parameter	Value	Bound
Channel omission degree	Ch_{Od}	k
Frame transmission delay	-	T_{td}
Number of inaccessibility events in an epoch	Ch_{Ii}	i
Total duration of inaccessibility in an epoch	t_{ina}	T_{ina}
Effective duration of an inaccessibility epoch	t_{p_ina}	T_{p_ina}

Figure 7. A relevant set of system-level parameters

The availability of system-level parameters constitutes an added-value extremely important to the design of a fault-tolerant real-time communication system: one can monitor/detect a potential lack of coverage of the system assumptions (given the bounds specified in Figure 2) and act accordingly (e.g. stopping in a fail-safe state).

The validation of system assumptions, through the monitoring of relevant dependability and timeliness parameters, can be incorporated: directly in the CANELy machinery, to be evaluated continuously, on a message-by-message basis; at the operating system level or in alternative as an independent *local support environment*, executing on top of a given operating system [28].

6. Timer Management in CANELy

This section is entirely dedicated to the management of timers in the CANELy architecture. Integration of inaccessibility control and timer management is addressed both for application and low-level protocol timers. Finally, a case-study addressing this integration process is discussed.

6.1. Application-level Timers

The simplest method to control the effects of inaccessibility in the operation of timeout-based protocols, is inspired from properties MCAN3 and MCAN4, in Figure 2: a corrective term, T_{ina}, the worst-case duration of inaccessibility in an epoch is added to the timeout value set in function of T_{td}, the maximum frame transmission delay. *No other action is needed to tolerate inaccessibility faults*.

The engineering of such a method is effective at application level, where it is usually equivalent to the use of a slightly longer timeout value. 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 [12, 11].

6.2. Low-level Protocol Timers

The application of the *inaccessibility flushing* technique to the management of protocol timers is specified in Figure 8. It uses the Ch_{Ina} signal to decide whether or not a protocol timer should be extended upon a timeout.

Timer Management - Inaccessibility Flushing

```
i00  timer_queue := empty                          // queue of timer descriptors
i01  // enqueue inserts a timer descriptor at the end of the timer queue
i02  // enqueue(timer_queue, tid⟨key⟩)
i03  // dequeue removes a timer descriptor from the timer queue
i04  // tid := dequeue(timer_queue, tid⟨key⟩)

t00  when lse_start_alarm.req (t_out, key) invoked do
t01       start alarm (t_out, tid⟨key⟩);          // key, is an user-level timer identifier
t02  od;
t03  when alarm (tid⟨key⟩) expires do             // timer expires at timer agency
t04       if Ch_Ina is asserted then
t05            enqueue (timer_queue, tid⟨key⟩);
t06       else
t07            lse_alarm.nty (tid);
t08       fi;
t09  od;
t10  when Ch_Ina is negated do                    // inaccessibility epoch ends
t11       while timer tid⟨key⟩ at the head of timer queue do
t12            tid := dequeue (timer_queue, tid⟨key⟩);
t13            lse_alarm.nty (tid);
t14       od;
t15  od;
t16  when lse_cancel_alarm.req (t_out, key) invoked do
t17       if timer tid⟨key⟩ is at timer queue then
t18            tid := dequeue (timer_queue, tid⟨key⟩);
t19       else
t20            cancel alarm (tid);
t21       fi;
t22  od;
```

Figure 8. Integrating CAN inaccessibility flushing and timer management

When a request to start a timer is issued at the programming interface (line *t00*, in Figure 8), a timer is started with the specified timeout value. As a general rule, the action of starting a timer at line *t01* can be easily mapped into the service interface of a standard timer agency [29, 30].

Should the Ch_{Ina} signal be asserted when a timer expires (line *t03*), the timer descriptor is inserted in a queue, waiting for the end of the corresponding inaccessibility epoch (lines *t04-t05*). Should the Ch_{Ina} signal be negated, meaning that no delays induced by inaccessibility events subsist in CAN operation, a notification that the timer has expired is issued (line *t07*).

For each timer that remains in the timer queue when the Ch_{Ina} signal is negated, meaning that an inaccessibility epoch has ended, the timer is removed from the timer queue and a notification that the timer has expired is issued (lines *t11-t13*). When a request to cancel a timer is issued, the timer descriptor is either removed from the timer queue or deleted from the system (lines *t17-t21*).

The user-level *key* identifier, has a dual role: it uniquely identifies the timer at protocol/user level; provided it also depends on message identifier, it will allow the use within timer management of a message-by-message inaccessibility flushing technique.

6.3. Case Study: CANELy TOTCAN Protocol

This section illustrates the importance of integrating the control of inaccessibility with timer management. The use of timeouts in TOTCAN, a totally ordered protocol included in CANELy, is thoroughly discussed in [8, 12].

A detailed description of the TOTCAN protocol can be found in [8, 12]. Protocol operation is sketched in Figure 9. The protocol works in two phases. In the first phase, the sender broadcasts a message using a CAN data frame. Recipients receiving the message do not deliver it immediately. Instead the message is enqueued and marked as UNSTABLE. To preserve network order, an UNSTABLE message is moved to the tail of the queue whenever a duplicate is received. In the second phase, the sender, upon confirmation of successful transmission, broadcasts an ACCEPT message, using EDCAN[9]. The ACCEPT message is a control message that is encapsulated in a remote frame, since no data field is needed. When the ACCEPT message is received, the associated data message is marked as STABLE and can be delivered as soon as it reaches the head of the queue.

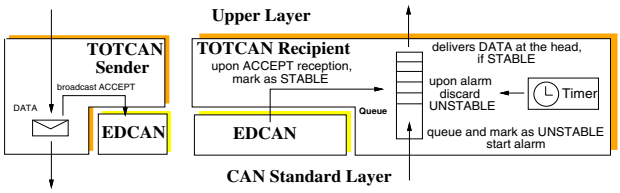


Figure 9. CANELy TOTCAN protocol

To prevent a possible deadlock in case of sender failure before sending the ACCEPT message, an alarm is used to detect sender failure and discard the UNSTABLE message. The timeout value, used to start the alarm, has important consequences on the protocol behavior.

[9]EDCAN, is an unordered reliable broadcast protocol included in the CANELy architecture [8, 12].

Optimum values for protocol timeouts are established, in general, as a function of T_{Td}, the maximum frame transmission delay. Should the network operate normally, an optimum protocol timeout value allows: a timely detection of sender failure; a low protocol recovery latency, e.g. the timely removal of the UNSTABLE message and ordered delivery of STABLE messages waiting in the TOTCAN queue.

However, an optimum sizing for protocol timers cannot be used without special inaccessibility control measures, or else they could timeout too early upon the occurrence of a period of inaccessibility. This might lead to protocol failure. Timeout-related faults have been identified as producing effects that may severely disturb the operation of CAN-based protocols and applications [31]. In TOTCAN, a timer expiring prematurely may cause a data message to be wrongly discarded because the corresponding ACCEPT message is delayed at the sender by the occurrence of a period of inaccessibility.

Adding a corrective term, given the worst-case duration of a period of inaccessibility, T_{ina}, to the timeout values is a sufficient condition for running synchronous (hard real-time) timeout-based protocols over a CAN infrastructure. However, timeouts do not have anymore an optimum sizing. Long timeouts unnecessarily increase the delay associated to the detection of a failure, in the absence of inaccessibility. Furthermore, T_{ina} may be much greater than T_{td}, at least at the low-levels of communication, causing the timeouts to be undesirably long. In TOTCAN, a typical T_{td} value for control messages is $560\mu s$ [12]. Since T_{ina} has a worst-case bound of $2160\mu s$, the value to use in the alarm would be $2720\mu s$, a value much higher than the optimum timeout [12].

Our approach to the control of inaccessibility in CANELy allows to take away inaccessibility from the timeout values. To the programmer it can be given a simple and elegant abstraction of a distributed environment which is always connected, with timers yielding optimal values with regard to the detection of failures.

The CANELy *inaccessibility flushing* technique, properly integrated with the management of timers, as specified in Figure 8, allows: the use of timeouts with optimum sizing; selectively add the real contribution of inaccessibility effects to the alarm period. In the TOTCAN protocol, this means the use of a timeout of $560\mu s$.

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. Furthermore, we have shown how the control of inaccessibility may be effectively integrated with timer management and protocol execution. As a case study, these concepts were applied to the operation of a totally ordered atomic broadcast protocol.

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 [21, 8, 9, 7, 10, 11].

Through CANELy we made the proof of concept of the possibility of building CAN-based highly fault-tolerant systems, thus contributing to dismiss ideas that CAN is not suited for designing hard real-time systems with very high dependability requirements.

References

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

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

[3] 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.

[4] 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.

[5] G. Leen and D. Heffernan, "TTCAN: A New Time-Triggered Controller Area Network", *Microprocessors and Microsystems Journal. Elsevier.*, vol. 26, no. 2, pp. 77–94, Mar. 2002.

[6] J. R. Pimentel and J. A. Fonseca, "FlexCAN: A Flexible Architecture for Highly Dependable Embedded Applications", in *3rd Int. Workshop on Real-Time Networks*, July 2004, Catania, Italy.

[7] 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, Madison, Wisconsin - USA. IEEE.

[8] 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.

[9] 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.

[10] 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.

[11] J. Rufino, P. Veríssimo, G. Arroz, and C. Almeida, "Control of Inaccessibility in CANELy", in *Proceedings of the 6th. International Workshop on Factory Communication Systems*, June 2006, pp. 35–44, Torino, Italy. 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. 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, Washington - USA. IEEE.

[22] Robert Bosch GmbH, *CAN Specification Version 2.0*, Sept. 1991.

[23] B. Gaujal and N. Navet, "Fault Confinement mechanisms on CAN:Analysis and Improvements", in *Proceedings of the 4th Conference on Fieldbus Technology*, 2001, Nancy, France. IFAC.

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

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

[26] 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.

[27] 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.

[28] H. Fonseca, L. Rodrigues, J. Rufino, and P. Veríssimo, "Local Support Environment: User Specification", Technical Report RT/50-90, INESC, Lisboa, Portugal, Aug. 1990.

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

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

[31] F. Corno, J. Acle, M. Reorda, and M. Violante, "A multi-level approach to the dependability analysis of networked systems based on the CAN protocol", in *Proceedings of the 17th symposium on Integrated circuits and system design*, Sept. 2004, pp. 71–75, Pernambuco, Brasil. ACM Press.