Node Failure Detection and Membership in CAN*ELy*

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

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

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

Abstract

Fault-tolerant distributed systems based on fieldbuses may benefit to a great extent from the availability of semantically rich communication services, such as those provided by group communication, clock synchronization, membership and failure detection. This is specially true of distributed critical control applications. However, the migration of those services to the realm of simple fieldbuses, such as the native Controller Area Network (CAN) protocol family, presents non-negligible problems, since it lacks most of the functionality required of a fault-tolerant distributed system, such as reliable message broadcast guarantees, distributed node failure detection, and site membership services.

As part of our endeavor to design a CAN-based infrastructure support for extremely reliable distributed computer control, dubbed CAN Enhanced Layer (CANELy), we have been addressing the problem of fault-tolerant communications on fieldbuses in a comprehensive way. In this paper, we show that node failure detection and site membership services can be efficiently supported by a simple software layer built on top of an exposed CAN controller interface.

1 Introduction

In less than two decades, distributed system technologies have penetrated several areas related with computer control and embedded systems. Specially in these applications, distribution needs to be combined with fault-tolerance and real-time. Due to this evolution, fieldbuses, which were in general conceived as low-cost network infrastructures intended for cabling optimization in device level (sensors/actuators, Programmable Logic Controllers, etc.) interconnection, were sometimes called to higher duties: performing as distributed systems.

The shortcomings of fieldbuses in such specially demanding uses, revealed by several studies [14, 19], should be put in perspective. In fact, observing the seminal work on fault-tolerant fieldbus-like architecture presented by the Maintainable Real-Time System (MARS) and then by the Time-Triggered Architecture (TTA) over the past decade [10], we can learn the requirements that should be fulfilled by a fault-tolerant real-time distributed system designed from scratch and devoted to embedded systems and computer control.

The question is: *can we endow an existing fieldbus with the necessary functionality to perform as a fault-tolerant real-time distributed system?*

Our thesis [16] is that fault-tolerant real-time distributed systems based on fieldbuses must benefit from the availability of semantically rich support services, such as those provided by group communication, clock synchronization, process group membership and failure detection. This is specially true of distributed critical control applications. However, the migration of those services to the realm of simple fieldbuses, such as the native Controller Area Network (CAN) protocol family, presents non-negligible problems. As part of our endeavor to design a CAN-based infrastructure support for extremely reliable distributed computer control, dubbed **CAN Enhanced Layer** (CANELy), we have been addressing the problem of fault-tolerant communications on fieldbuses in a comprehensive way, detailed in Section 2.

In this paper we discuss the design of two fundamental parts of a highly dependable real-time communication system: the provision of reliable node failure detection and site membership services. We also show that they can be efficiently supported by a simple software layer built on top of an exposed CAN controller interface.

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

A comparison of TTP, the Time-Triggered Protocol, and CAN, with regard to a relevant set of dependability and timeliness-related parameters, is summarized in Figure 1. We use an adaptation of the analysis presented in [9].

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

[†] Faculdade de Ciências da Universidade de Lisboa, Campo Grande - Bloco C5, 1700 Lisboa, Portugal. Tel: +351-217500087 - Fax: +351-217500084. Navigators Home Page: http://www.navigators.di.fc.ul.pt.

The Time-Triggered Protocol [10] provides the services required to the implementation of the time-triggered architecture, namely message acknowledgment in group communication, clock synchronization and site membership. A TTP-based system consists of fail-silent nodes connected by two replicated broadcast communication channels. Each message is sent on both channels. Media-access is controlled by a conflict-free Time Division Multiple Access (TDMA) strategy. It is assumed that nodes have their clocks synchronized within a known precision.

Parameter	TTP	Standard CAN
Error detection	value and time domains	value domain
Omission handling	masking	detection/recovery
	frame diffusion	frame retransmission
Media redundancy	no	no
Channel redundancy	yes	
Babbling idiot avoidance	bus guardian	not provided
Communications	broadcast	broadcast
Membership service	provided	not provided
Clock synchronization	in μs range	

Figure 1. Comparison of TTP and CAN

From the data reported in Figure 1, one may be lead to the conclusion that CAN is not suited for hard real-time systems with high dependability requirements, given the limitations of the CAN standard layer, with respect to the provision of strict availability, reliability and timeliness attributes. However, this may be due to a fundamental inability, in which case there is little chance that any hardware or software will solve the problem, or else, it may be due to some insufficiency in functionality, in which case it can be mended by adding the latter.

We have realized that what was missing in the native CAN fieldbus to attain levels of dependability comparable to those of TTP-based systems, was indeed a set of fault tolerance and timeliness-related services. Moreover, we have shown that these can be provided off-the-shelf (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). The central component of CANELy is naturally the standard CAN layer, complemented/enhanced with some simple machinery and low-level protocols.

In the course of our work, we already have: addressed the effects of a subtle form of partitioning, through the study of CAN behavior in the presence of network errors [22]; dismissed the misconception that CAN supports a totally ordered atomic message broadcast service and designed a protocol suite which handles the problem effectively [18]; and designed an innovative and extremely simple redundancy scheme for CAN [17].

3 Basic Dependability of CAN

The CAN fieldbus is a multi-master network that uses a twisted pair cable as transmission medium [8, 3]. The network maximum length depends on the data rate. Typical values are: 40m @ 1 Mbps; 1000m @ 50 kbps. Bus signaling takes one out of two values: *recessive*, otherwise the state of an idle bus; *dominant*, 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. Frames that have lost arbitration or have been destroyed by errors are automatically scheduled for retransmission. A *frame* is a piece of encapsulated information traveling on the network. It may contain a *message*, a user-level piece of information.

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 [18]. 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 [18]. However infrequent they may be, the probability of its occurrence is high enough to be taken into account for highly fault-tolerant applications of CAN.

Fault-confinement in CAN aims at restricting the influence of defective nodes in bus operation. It is based on two counters recording, at each node, transmit and receive errors, that is, frame *omissions*. Though these mechanisms are extremely useful to the (local) control of *omission failures* [18], they are helpless in respect to the distributed signaling of such failures and the detection of node crash failures.

4 System Model

Next, we enumerate our fault assumptions for the system and discuss the CAN properties that underpin our system model, as established in [18] and [16].

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

The model addresses a set of communicating processes sitting on a message passing subsystem implemented by a CAN fieldbus infrastructure. Each process is attached to the network through a CAN controller. Together, they form a node. We assume that the processes are fail-silent and blame all temporary failures on the CAN network components. However, when a process crashes, the whole node crashes. In consequence, we may refer to *process* and *node* interchangeably.

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

The **CAN bus** is viewed as a single-channel broadcast local network with the following failure mode assumptions for the network components (anything between two processes, including network adapters and the physical layer path):

- 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. medium partition[3]).

The weak-fail-silent assumption can be enforced with high coverage for the CAN controller by fault confinement mechanisms [18, 16]. This is important to the preservation of CAN timeliness and to the parameterization of protocol operation. Furthermore, we assume that:

- in a time interval of reference, no more than f nodes/processes are affected by crash failures.

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 [21]. Then, on top of the basic MAC-level functionality, CAN has error-recovery mechanisms that yield interesting message properties. Again, this has the flavor of the logical link control (LLC) sub-layer in LANs. We describe both sets of properties below.

CAN MAC-level properties

Figure 2 enumerates from [18, 16] a relevant set of CAN MAC-level properties. Properties MCAN1 and MCAN2 impose correctness in the value domain, in a broadcast. Property MCAN1 formalizes that it is physically impossible for a node to send conflicting information to different nodes, in the same broadcast. Property MCAN2 derives directly from CAN built-in error detection and signaling; the residual probability of undetected frame errors is negligible [4].

MCAN1 - Broadcast: correct nodes receiving an uncorrupted frame transmission, receive the same frame.

MCAN2 - Error Detection: correct nodes detect any corruption done by the network in a locally received frame.

MCAN3 - Bounded Omission Degree: in a known time interval T_{rd}, omission failures may occur in at most k transmissions.

MCAN4 - Bounded Transmission Delay: any frame queued for transmission is transmitted on the network within a bounded delay of $T_{td}+T_{ina}$.

Figure 2. CAN MAC-level properties

Property MCAN3 maps the failure semantics introduced earlier onto the operational assumptions of CAN, being $k \geq f_o$. This property is crucial to implement protocols yielding bounded termination times.

The behavior of CAN in the time domain is described by MCAN4, which specifies a maximum frame transmission delay, which is T_{td} in the absence of faults. It depends on message latency classes and offered load bounds [20, 23, 12]. The bounded frame transmission delay also includes T_{ina}, the maximum duration of an inaccessibility fault, a period where the network refrains from providing service, although remaining operational [22].

CAN LLC-level properties

At the LLC level, the failure modes that we have identified cause the message-level properties of CAN to be somewhat different. Namely, while the omission failures specified by MCAN3 are masked in general at the LLC level by the retry mechanism of CAN, the existence of inconsistent omissions as discussed in [18] postulates two things:

- that there may be message duplicates when they are recovered;
- that some j of the k omissions will show at the LLC interface as inconsistent omissions.

Figure 3 enumerates from [18] a relevant set of the LLC-level properties of CAN. Properties LCAN2 and LCAN3 result from the frame retransmission scheme of CAN, in the presence of (inconsistent) omissions. Property LCAN4 specifies the probability of inconsistent omission failures j, where j is normally several orders of magnitude smaller than k. Property LCAN4 is extensively exploited in the design

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

[3]This assumption can be enforced through the media redundancy scheme described in [17].

LCAN1 - Validity: if a correct node broadcasts a message, then the message is eventually delivered to a correct node.

LCAN2 - Best-effort Agreement: if a message is delivered to a correct node, then the message is eventually delivered to all correct nodes, if the sender remains correct.

LCAN3 - At-least-once Delivery: any message delivered to a correct node is delivered at least once.

LCAN4 - Bounded Inconsistent Omission Degree: in a known time interval T_{rd}, inconsistent omission failures may occur in, at most, j transmissions.

Figure 3. CAN LLC-level properties

of reliable communication services [18, 15], node failure detection and site membership included.

5 CAN Standard Layer and Extension

The CAN standard layer is made from a CAN controller and the corresponding software driver, that includes the following primitives:

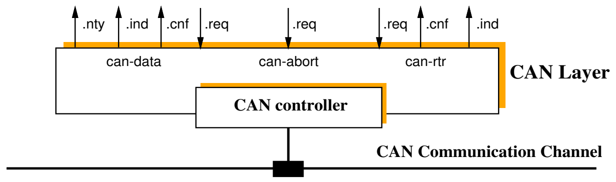
- *request* the transmission (.req) of data or control messages, being the control messages encapsulated in remote frames;
- *confirm* to the user a successful message transmission (.cnf);
- *indication* of a message arrival (.ind).

The semantics of each particular primitive is summarized in Figure 4. Most of the attributes are defined in the standard document [8] and have an appropriate support from the CAN controller. However, a few exceptions exist: i) reception of own transmissions is not assured in all controllers [6], so low-level engineering may be required; ii) the local execution environment must process frame arrivals with a latency low enough to guarantee that no receive buffer overrun incidents will ever occur[4].

Both data and remote frames have a message control field. Data frames may also include message data. The message control field or *message identifier* (mid) consists of a *type* reference, an (optional) *reference number* and a *node identifier*.

The set of primitives specified in Figure 4 includes an extension to the standard interface: a *notification* primitive (.nty), signaling the arrival of a data frame (own transmissions included). However, no message data is delivered. Only the message control information is included in the notification.

[4]This kind of omission failures have not been included in our model.



Primitives		Semantics summary
Data	Remote	
can-data.req		Only a node is allowed to transmit, at a time.
	can-rtr.req	Several nodes may simultaneously transmit the same remote frame.
can-data.cnf	can-rtr.cnf	Signals the successful transmission of a frame.
can-data.ind	can-rtr.ind	Signals the arrival of a data/remote frame, including own transmissions.
can-data.nty		*Extension to standard*: signals the arrival of a data frame, without delivering the message data.
can-abort.req		Aborts a frame transmission request. Has effect only on pending requests.

Figure 4. CAN standard layer structure and interface

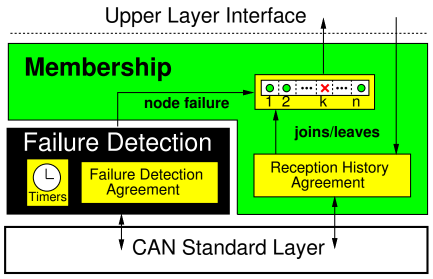
6 Failure Detection and Membership in CAN

A membership service is intended to provide, at any given time, consistent information about the state of a collection of participants. Though "participant" may generically refer either to a process in a group of processes or to a site/node (processor) in a distributed system, this paper is specially concerned with the provision of site membership services in the CANELy architecture.

The availability of a site membership service is extremely relevant to CAN reliable communication, in the sense that it is a crucial assistant for process group membership management and it may be used to simplify the design of other protocols (e.g. group communication, clock synchronization).

The architecture of a failure detection/membership protocol suite, supported by a software layer built on top of an exposed CAN controller interface, is informally described in Figure 5. Upper layer protocol entities may request the local node to join/leave the set of active nodes or obtain the current site/node membership view. A notification of a change in the composition of the set of currently active nodes is due whenever a node joins/leaves the service (an event primarily handled by the reception history agreement module) or upon the detection of a node crash failure, by the failure detection mechanisms.

Let us then assume a network composed of $\mathcal{N}$ nodes. We use $\mathcal{V}_s^t(n)$ to denote the site membership view at node $n \in \mathcal{N}$, after the t^{th} execution of the membership proto-



Description	*Parameters*	
Invocation Primitives (msh-can.req)		
Join/Leave	r	site/node identifier
Get Membership View	$\mathcal{V}_s$	set of active sites/nodes
Notification Primitives (msh-can.nty)		
Membership Change	$\mathcal{V}_s$	set of active sites/nodes
	$\mathcal{F}_s$	set of failed nodes

Figure 5. CAN node failure detection and site membership protocol suite

col. Naturally, $\mathcal{V}_s^t(n) \subseteq \mathcal{N}$. For simplicity of exposition, we omit: the superscript identifying the protocol round, when it refers to the last protocol execution; the variable identifying the node, when it refers to the membership view agreed by all correct nodes. The fundamental problem to be solved with regard to the provision of a site/node membership service in CAN concerns the ability of correct nodes to reach agreement on the $\mathcal{V}_s$ set, within a bounded and known time. In essence, this is equivalent to a *consensus problem* [7].

6.1 Signaling Node Activity

As a general rule, each node in the CANELy architecture may signal its active state through the broadcast of a life-sign message, which works as an *heartbeat*. A similar approach has been followed in LAN-based designs, such as those described in [5] and [11].

In CANELy, to save network bandwidth (a scarce resource in CAN) normal traffic is implicitly used to signal node activity. The issuing of explicit life-sign messages is restricted to nodes transmitting only periodic traffic with a period higher than the failure detection latency and/or sporadic/aperiodic traffic. This scheme is effective, provided that only a subset of the nodes need to explicitly issue life-sign messages. Given that all the relevant protocol information (i.e. message type and node identifier) can be included in the control field of life-sign messages, these can be encapsulated in CAN *remote* frames, with no data field.

The delivery of node activity signals (explicit or implicit) cannot be guaranteed when a given message transmission is affected by an inconsistent omission error and the sender fails before completing the transmission of such a message (cf. LCAN2, in Figure 3). This happens because CAN native mechanisms do not secure message delivery to all correct nodes in those circumstances [18]. In any case: the node crash failure is detected, by the subset of the correct nodes[5] not receiving the node activity signal; the consistency of node failure notifications can be enforced by other mechanisms; the upper bound specified for the delay in the detection of node crash failures is preserved.

6.2 Enforcing Agreement

This section thoroughly discusses how a fundamental set of low-level agreement enforcement mechanisms, concerning the consistent handling of node crash failures and node join/leave events, can be implemented as software modules (micro-protocols) built on top of an exposed CAN controller interface.

Failure Detection Agreement

In the event of sender failure, the dissemination of life-sign messages may be inconsistent. Additional mechanisms are required to secure that upper layer entities are consistently notified of the lack of node activity, which constitutes a signal of a node crash failure.

The *Failure Detection Agreement* (FDA) protocol, specified in Figure 6, is a simplified and optimized version of the "Eager Diffusion" (EDCAN) protocol described in [18] and aims to secure the reliable broadcast of a *failure-sign* message. The identifier of the failed node (r) is specified, e.g. by upper layer failure detection entities, when the protocol is invoked. The failed node identifier (r) and the message type (type is set to FDA) are the protocol parameters to be included in the control field (*mid*) of a CAN frame, meaning that *remote* frames may be used.

The use of CAN *remote* frames contributes to save network bandwidth, since the transmission of identical remote frames from two or more nodes can be "clustered" in a single physical frame, due to the wired-and nature of the CAN physical layer [18].

The FDA protocol works as follows: when invoked (through the fda-can.req primitive), the sender issues a single request to the CAN layer for the transmission of a failure-sign message; the recipients, deliver the first copy of the failure-sign message to the layer above (using the fda-can.nty primitive) and in the absence of an equivalent transmit request, invoke the CAN layer for failure-sign retransmission.

The design of the FDA protocol is based on the observation that: node crash failures are relatively rare events; it is reasonable to assume that in a period of reference, no

[5]This subset may have only one element.

```
Failure Detection Agreement (FDA)

Initialization
i00  fs_ndup(mid) := 0;                    // number of failure-sign duplicates
i01  fs_nreq(mid) := 0;                    // number of failure-sign transmit requests

Sender, at p                               // r, is the failed node identifier
s00  when fda-can.req(r) do                // $\boxed{\text{fda-can.req}}$ protocol invocation
s01      fs_nreq(mid) := fs_nreq(mid) + 1;
s02      if fs_nreq(mid) = 1 then
s03          can-rtr.req(mid⟨FDA,r⟩);      // failure-sign transmit request
s04      fi;
s05  od;

Recipient, at q
r00  when can-rtr.ind(mid⟨FDA,r⟩) do       // r, is the failed node identifier
r01      fs_ndup(mid) := fs_ndup(mid) + 1;
r02      if fs_ndup(mid) = 1 then
r03          fda-can.nty(r);               // $\boxed{\text{fda-can.nty}}$ delivery to layer above
r04          fs_nreq(mid) := fs_nreq(mid) + 1;
r05          if fs_nreq(mid) = 1 then
r06              can-rtr.req(mid⟨FDA,r⟩);          // failure-sign transmit request
r07          fi;
r08      fi;
r09  od;
```

Figure 6. Failure detection agreement micro-protocol

more than a given number of nodes, f, may fail; f exhibits moderately low values.

RECEPTION HISTORY AGREEMENT

We address next how to consistently handle node join/leave events. The *Reception History Agreement* (RHA) protocol, specified in Figure 7, aims at the efficient handling of multiple join/leave requests, using a small amount of network bandwidth.

The RHA protocol works as follows: each node in the site membership view proposes its own value for a *reception history vector* (RHV); protocol execution ensures that all correct nodes agree on the value of RHV to be delivered to upper layer entities.

We assume that the RHA protocol shares with upper layer entities the following local variables: $\mathcal{V}_s$, the site membership view; $\mathcal{V}_j$, the set of nodes in a joining process; $\mathcal{V}_l$, the set of nodes requesting to be withdrawn from the site membership view.

For a node included in the site membership view, the execution of the RHA protocol starts: upon a request from upper layer entities (line *s00*, in Figure 7); after the reception of an RHV signal from a remote peer entity (line *r00*). In any case, the protocol establishes an initial value for the reception history vector, based on the values of $\mathcal{V}_s$, $\mathcal{V}_j$ and $\mathcal{V}_l$, and requests the broadcast of the corresponding RHV signal[6], as specified in line *a07* of Figure 7.

```
Reception History Agreement (RHA)

Initialization
i00  rhv_ndup(mid) := 0;             // number of duplicates, kept for each RHV signal
i01  tid := NULL;                    // timer identifier
i02  $\mathcal{V}_{RHV} := \mathcal{V}'_{RHV} := \emptyset$;            // local and auxiliary Reception History Vectors
i03  // $\mathcal{V}_s, \mathcal{V}_j, \mathcal{V}_l$
i04  // Shared Variables: full-member ($\mathcal{V}_s$), joining ($\mathcal{V}_j$) and leaving ($\mathcal{V}_l$) node sets

rha-init-send (auxiliary function)
a00  rha-init-send($\mathcal{V}_m$,r) do                 // invoked when protocol execution starts
a01      tid := start_alarm ($T_{RHA}$);   // local timer: RHA max. termination time
a02      if $r \in \mathcal{V}_s$ then
a03          $\mathcal{V}_{RHV} := ((\mathcal{V}_s \cup \mathcal{V}_j) - \mathcal{V}_l) \cap \mathcal{V}_m$;    // full-member initial vector
a04      else
a05          $\mathcal{V}_{RHV} := \mathcal{V}_m$;               // non-members use the received vector
a06      fi;
a07      can-data.req (mid⟨RHA,$\#\mathcal{V}_{RHV}$,r⟩, $\mathcal{V}_{RHV}$);
a08      rha-can.nty (INIT, $\emptyset$);               // $\boxed{\text{rha-can.nty}}$ delivery to layer above
a09  od;

Sender, at p                                   // action executed only by full-members
s00  when rha-can.req() and $p \in \mathcal{V}_s$ do      // $\boxed{\text{rha-can.req}}$ protocol invocation
s01      if tid = NULL then
s02          rha-init-send ($\mathcal{N}$,p);               // $\mathcal{N}$, is the set of all sites/nodes
s03      fi;
s04  od;

Recipient, at q
r00  when can-data.ind(mid⟨RHA,$\#\mathcal{V}'_{RHV}$,r⟩, $\mathcal{V}'_{RHV}$:=mess⟨$\mathcal{V}_{RHV}$⟩) do
r01      rhv_ndup(mid⟨RHA,$\#\mathcal{V}'_{RHV}$⟩) := rhv_ndup(mid⟨RHA,$\#\mathcal{V}'_{RHV}$⟩) + 1;
r02      if tid = NULL then
r03          rha-init-send ($\mathcal{V}'_{RHV}$,q);
r04      else if ($(\mathcal{V}_{RHV} \cap \mathcal{V}'_{RHV}) \neq \mathcal{V}_{RHV}$) then
r05              can-abort.req (mid⟨RHA,$\#\mathcal{V}_{RHV}$,q⟩);
r06              $\mathcal{V}_{RHV} := \mathcal{V}_{RHV} \cap \mathcal{V}'_{RHV}$;          // new $\mathcal{V}_{RHV}$ value
r07              can-data.req (mid⟨RHA,$\#\mathcal{V}_{RHV}$,q⟩, $\mathcal{V}_{RHV}$);
r08          else if rhv_ndup(mid⟨RHA,$\#\mathcal{V}_{RHV}$⟩) > j then  // j, see LCAN4
r09                  can-abort.req (mid⟨RHA,$\#\mathcal{V}_{RHV}$,q⟩);
r10              fi;
r11          fi;
r12      fi;
r13  od;
r14  when alarm (tid) expires do               // RHA protocol timer expires
r15      rha-can.nty (END, $\mathcal{V}_{RHV}$);          // $\boxed{\text{rha-can.nty}}$ delivery to layer above
r16      tid := NULL;
r17      $\mathcal{V}_{RHV} := \emptyset$;
r18  od;                                       // protocol execution has ended
```

Figure 7. Reception history agreement micro-protocol

Should the initial value of $\mathcal{V}_{RHV}$ be the same for all the nodes included in the site membership view[7], the RHA protocol simply ensures the reliable broadcast of the RHV signal, using a technique inspired by the "Eager Diffusion" (EDCAN) protocol [18]. Each node requests the transmission of an RHV signal containing the value locally assigned

[6] A message containing the value obtained for $\mathcal{V}_{RHV}$. In the message control field (*mid*) it is specified: the message type (set to RHA); the number of active nodes in $\mathcal{V}_{RHV}$, as given by $\#\mathcal{V}_{RHV}$.

[7] Meaning: $\mathcal{V}_j$ and $\mathcal{V}_l$ have identical values for any node $r \in \mathcal{V}_s$.

to $\mathcal{V}_{RHV}$. This transmit request is valid, until: the RHV signal is transmitted; the number of message copies associated to the current RHV value does not exceed j, the inconsistent omission degree bound (line *r08*, in Figure 7); an RHV signal, requiring the removal of a node currently included in $\mathcal{V}_{RHV}$, is received (lines *r04-r06*).

The possible issuing of inconsistent $\mathcal{V}_{RHV}$ values by some nodes in the $\mathcal{V}_s$ set is a consequence of the occurrence of inconsistent omission failures during the transmission of node join/leave requests, that lead to inconsistent values for $\mathcal{V}_j$ and/or $\mathcal{V}_l$. Upon the processing of the remote RHV signal, any node not included in both RHV sets (local and remote) is removed from the current $\mathcal{V}_{RHV}$ value and the broadcast of the new $\mathcal{V}_{RHV}$ value is requested (line *r07*, of Figure 7). The number of rounds of the RHA protocol that need to be executed to reach *consensus* on the value of $\mathcal{V}_{RHV}$ to be delivered to upper layer entities is bounded and can be known [16].

Given that nodes in a joining process may not have a valid $\mathcal{V}_s$ value, such nodes are not allowed to start the RHA protocol in isolation (line *s00*, of Figure 7). However, those nodes are obliged to: initiate the execution of the RHA protocol as soon as they receive an RHV signal from another node; use the contents of the RHV signal as the initial value of $\mathcal{V}_{RHV}$ (line *a05*); participate in the dissemination of the different $\mathcal{V}_{RHV}$ values; deliver to the upper layer entities the final $\mathcal{V}_{RHV}$ value (line *r15*, in Figure 7).

6.3 Node Failure Detection

This section discusses the design of a node failure detection service for CAN, that uses the FDA micro-protocol to secure a consistent signaling of node failures. The failure detection service has been designed to reduce the network bandwidth consumed by explicit life-sign messages; node activity may be signaled implicitly, transmitting normal traffic. This scheme may be very efficient, namely when CAN applications exhibit a cyclic traffic pattern [20] and the message periods are smaller than the specified node failure detection latency.

The node failure detection protocol is specified in Figure 8. The node failure detection service is started (or stopped) on a node-by-node basis, through the issuing of a request primitive (fd-can.req) by upper layer entities.

Upon a request to start the service (lines *f00-f01*, in Figure 8), a surveillance timer is started for the specified node: local timers have a duration equal to T_m, the heartbeat period, i.e. the maximum time interval between consecutive life-sign transmit requests; timers monitoring the activity of remote nodes should additionally include T_{md}, the network message transmission delay bound[8]. A request to stop the

Failure Detection Protocol

```
Initialization
i00  tid(r) := NULL;                            // timer identifiers, kept for each node

fd-alarm-start (auxiliary function)
a00  fd-alarm-start(r) do
a01      if r = p then
a02          tid(r) := start_alarm ($T_m$);                  // local timer
a03      else
a04          tid(r) := start_alarm ($T_m + T_{md}$);         // remote node monitoring
a05      fi;
a06  od;

Node failure detector, at p
f00  when fd-can.req(START,r) do        // [fd-can.req] protocol invocation: start
f01      fd-alarm-start (r);
f02  od;
f03  when can-data.nty(r) or can-rtr.ind(mid⟨ELS,r⟩) do   // node activity detected
f04      fd-alarm-start (r);                               // restart timer
f05  od;
f06  when alarm(tid⟨r⟩) expires do                         // timer expires
f07      if r = p then
f08          can-rtr.req (mid⟨ELS,r⟩);      // explicit life-sign (ELS) broadcast
f09      else
f10          fda-can.req (r);                              // remote node has failed
f11      fi;
f12  od;
f13  when fda-can.nty(r) do
f14      cancel_alarm(tid⟨r⟩);
f15      fd-can.nty (r);                 // [fd-can.nty] delivery to layer above
f16  od;
f17  when fd-can.req(STOP,r) do         // [fd-can.req] protocol invocation: stop
f18      cancel_alarm(tid⟨r⟩);
f19  od;
```

Figure 8. Failure detection protocol

service (lines *f17-f18*, of Figure 8) results in the termination of the timer associated to the given node.

The can-data.nty primitive (cf. Figure 4) allows the node failure detection protocol to be notified each time a data message is received from the network (own transmissions included). This simple mechanism allows the high-level data messages to be implicitly used as heartbeats. Explicit life-sign messages may need to be issued, but only if and when the time between message transmit requests is higher than the heartbeat period.

Upon the notification a given node is active, the corresponding surveillance timer is restarted (lines *f03-f04*, of Figure 8). If a given node remains silent during a period longer than the specified detection latency, the surveillance timer is not restarted and it will expire (line *f06*, in Figure 8). If the timer is associated to the local node, the CAN standard layer is invoked for the broadcast of a life-sign message (line *f08*, in Figure 8). Otherwise, the expired timer is an indication that a remote node has failed and the FDA micro-protocol is used to consistently disseminate such a sign (line *f10*, in Figure 8). The failure notification is delivered, as soon as it is received (lines *f13-f15*, of Figure 8), to a companion site membership protocol.

[8]Being: $T_{md} = T_{td} + T_{ina}$, as defined by MCAN4 (Figure 2).

6.4 Membership Protocol

This section describes a CAN membership service intended to consistently handle node crash failures and node join/leave events. The operation of the membership protocol is specified in Figure 9. Some details have been omitted for simplicity of exposition.

The site membership protocol aims to ensure a consistent value for $\mathcal{V}_s$, the site membership view, at all the correct nodes. In the design of the membership protocol it is assumed that any node removed from $\mathcal{V}_s$, in the sequence of a withdrawn request or after the failure of the node, does not initiate a reintegration attempt before a period much higher than T_m, the membership cycle period, has elapsed.

The operation of the membership protocol of Figure 9 is quite straightforward. The request of a node to be integrated in the set of active sites is broadcast using the standard CAN interface. Local and remote requests issued during a given membership cycle are handled in the same way, being included in $\mathcal{V}_j$, the set of nodes in the joining process. A similar treatment is given to node leave requests, that are included in $\mathcal{V}_l$, the set of nodes requesting to be withdrawn from the site membership view.

When the membership cycle timer expires at some node and there is at least one pending join/leave request, the RHA micro-protocol is invoked to secure the consistency of join/leave operations. Should no request be pending when the membership cycle timer expires, the execution of the RHA micro-protocol is skipped, in order to save CAN bandwidth. If the membership timer[9] expires at a non integrated node, meaning no full-member is active, the current value of $\mathcal{V}_j$ is (temporarily) assigned to $\mathcal{V}_s$, the site membership view, before the RHA micro-protocol is invoked.

When the execution of the RHA micro-protocol ends, the value of $\mathcal{V}_s$ is updated, using the values of a reception history vector, $\mathcal{V}_{RHV}$, established by RHA execution, and of $\mathcal{F}_s$, the set of node crash failures detected during the current membership cycle. If the composition of $\mathcal{V}_s$ is modified, in the sequence of a node join/leave operation: a membership change notification is delivered to the upper layer entities; the $\mathcal{V}_j$ and $\mathcal{V}_l$ sets are updated[10]; the failure detection service is started/stopped for each node in those circumstances.

The issuing of a membership change notification is used at the relevant nodes as an indication of success of a node join/leave operation. At fully integrated nodes, a membership change notification is also delivered to upper layer entities, upon the signaling of a node crash failure, by the companion failure detection services.

[9]Initially set with a timeout value, $T_{waitjoin}$, much longer than the membership cycle period, T_m.

[10]An auxiliary set of node identifiers, $\mathcal{V}_j'$, allows to remove from $\mathcal{V}_j$, within a period of two membership cycles, any node that on account of an inconsistent failure, does not succeed to be included in $\mathcal{V}_s$.

Site Membership Protocol

```
Initialization
i00  tid := NULL;                                            // timer identifier
i01  $\mathcal{V}_s := \mathcal{V}_j := \mathcal{V}_j' := \mathcal{V}_l := \mathcal{F}_s := \emptyset$;     // membership protocol data sets
msh-view-proc (auxiliary function)
a00  msh-view-proc($\mathcal{V}_m$) do
a01      $\mathcal{V}_s := \mathcal{V}_m - \mathcal{F}_s$;  $\mathcal{F}_s := \emptyset$;                // updating membership view
a02  od;
msh-data-proc (auxiliary function)
a03  msh-data-proc() do
a04      for each $s \in (\mathcal{V}_j \cap \mathcal{V}_s)$ do                  // processing node joins
a05          fd-can.req(START,s); od;
a06      $\mathcal{V}_j := (\mathcal{V}_j - \mathcal{V}_s) - \mathcal{V}_j'$;   $\mathcal{V}_j' := \mathcal{V}_j$;
a07      for each $s \in (\mathcal{V}_l \cap (\neg\mathcal{V}_s))$ do              // processing node leaves
a08          fd-can.req(STOP,s); od;
a09      $\mathcal{V}_l := \mathcal{V}_l - (\neg\mathcal{V}_s)$; od;
msh-chg-nty (auxiliary function)
a10  msh-chg-nty($\mathcal{V}_m$,$\mathcal{F}_m$) do                          // delivery to layer above
a11      if $p \in \mathcal{V}_s$ then                                  // p, is the local node identifier
a12          msh-can.nty ($\mathcal{V}_m$, $\mathcal{F}_m$);                     // $\boxed{\text{msh-can.nty}}$ full-member
a13      else if $p \in \mathcal{V}_l$ then
a14              cancel_alarm(tid);
a15              msh-can.nty ($\mathcal{V}_s$, $\{p\}$);                 // $\boxed{\text{msh-can.nty}}$ leaving node
a16          fi;
a17      fi;
a18  od;

Site membership, at p
s00  when msh-can.req(JOIN,r:=p) and $r \notin \mathcal{V}_s$ do  // $\boxed{\text{msh-can.req}}$ join request
s01      tid := start_alarm ($T_{waitjoin}$);                   // timer: max. join wait delay
s02      can-rtr.req(mid⟨JOIN,p⟩);
s03  od;
s04  when can-rtr.ind(mid⟨JOIN,r⟩) do
s05      $\mathcal{V}_j := \mathcal{V}_j \cup \{mid\langle r\rangle\}$;
s06  od;
s07  when msh-can.req(LEAVE,r:=p) and $r \in \mathcal{V}_s$ do  // $\boxed{\text{msh-can.req}}$ node leave
s08      can-rtr.req(mid⟨LEAVE,p⟩);
s09  od;
s10  when can-rtr.ind(mid⟨LEAVE,r⟩) do
s11      $\mathcal{V}_l := \mathcal{V}_l \cup \{mid\langle r\rangle\}$;
s12  od;
s13  when fd-can.nty(r) do                            // node failure notification
s14      $\mathcal{F}_s := \mathcal{F}_s \cup \{r\}$;
s15      msh-chg-nty ($\mathcal{V}_s - \mathcal{F}_s$, $\{r\}$);          // membership change: node failure
s16  od;
s17  when rha-can.nty(INIT,$\emptyset$)) or alarm (tid) expires do  // timer expires at a node
s18      if timer tid has expired and $p \notin \mathcal{V}_s$ then          // no full-member nodes
s19          $\mathcal{V}_s := \mathcal{V}_j$;
s20      fi;
s21      tid := start_alarm ($T_m$);                 // timer: $T_m$, is the msh. cycle period
s22      if $\mathcal{V}_j \neq \emptyset \vee \mathcal{V}_l \neq \emptyset$ then
s23          rha-can.req ();
s24      else
s25          msh-view-proc ($\mathcal{V}_s$);
s26      fi;
s27  od;
s28  when rha-can.nty(END,$\mathcal{V}_{RHV}$) do          // RHA protocol finishes execution
s29      msh-view-proc ($\mathcal{V}_{RHV}$);
s30      if $(\mathcal{V}_j \cap \mathcal{V}_s) \neq \emptyset \vee (\mathcal{V}_l \cap (\neg\mathcal{V}_s)) \neq \emptyset$ then
s31          msh-chg-nty ($\mathcal{V}_s$,$\emptyset$);               // membership change: node join/leave
s32      fi;
s33      msh-data-proc ();
s34  od;
```

Figure 9. Membership protocol

6.5 Protocol Efficiency

The fraction of CAN bandwidth used by the site membership protocol suite with regard to the overall duration of the membership cycle period, T_m, obtained by analytical evaluation [16], is represented in Figure 10, for a relevant set of operating conditions. In the absence of node crash failures and of join/leave events, no changes in the site membership view need to be disseminated. The network bandwidth consumed by the site membership protocol concerns the issuing of at most b life-sign messages. If a given number of nodes, f, fail within a membership cycle period, the FDA micro-protocol is invoked and the corresponding network bandwidth needs to be added to the CAN bandwidth consumed in the dissemination of explicit life-sign messages. Similar considerations apply to the processing of a given number, c, of join/leave requests, by the RHA micro-protocol.

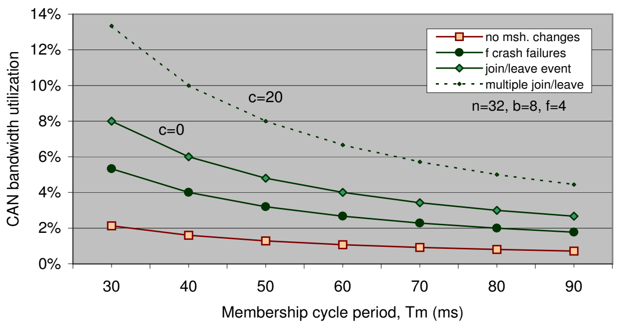


Figure 10. CAN bandwidth utilization by the site membership protocols

A very conservative approach is taken in the analysis of the CAN bandwidth used by the site membership micro-protocols, in a period of reference (Figure 10): multiple events occur in the same period of reference; every micro-protocol consumes the maximum amount of network bandwidth, meaning that both protocol and network-related overheads are accounted for [16]; extremely harsh operating conditions are assumed. In the given period of reference, $b = 8$ nodes issue a life-sign message and $f = 4$ nodes may fail.

Should the number of requests to join/leave the site membership view be moderate, the utilization of CAN bandwidth in the period of reference, is acceptably low, given the values of T_m specified in Figure 10. A significant increase in the utilization of CAN bandwidth, by the membership protocol suite, occurs in the presence of a massive number of join/leave requests[11], as illustrated in Figure 10.

[11] Each join/leave request contributes with an increase of $0,27\%$ (assuming $T_m = 30ms$) to the overall utilization of CAN bandwidth, by the site membership protocol suite.

6.6 Related Work

The industry standard CAN Application Layer (CAL), e.g. used in the *CANopen* communication profile [1], specifically defines *network management* service elements for the detection of node crash failures. A *master-slave* architecture is used: one master node cyclically inquires each slave node, through the issuing of a CAN *remote* frame; the slave node replies with its actual state. Alternatively, a *producer-consumer* communication model can be used: nodes broadcast a *heartbeat* message containing their status. The main disadvantages of this approach are related to: its centralized nature; the lack of an effective support to fault-tolerant node failure detection and site membership services.

Distributed network management services are provided in OSEK, the industry standard for automotive electronics [13]. In OSEK network management, every node is actively monitored by every other node in the network, using a logical ring organization that includes the set of currently active nodes. Node join/leave events and node failures are handled in two-phases, with nodes being temporarily included in a transient configuration of the logical ring, before their addition/removal from the stable configuration. The disadvantages of this method are concerned with: a potentially high utilization of network bandwidth and a high node failure detection latency. For example, for a reference value of $T_m = 70...100ms$, the period required to detect the failure of a node may be in the order of one second [13].

7 Conclusions

There is an increasing demand for distributed fault-tolerant systems based on fieldbus technologies. Given the utility of services such as node failure detection and site membership in CAN-based fault-tolerant distributed systems, we have: defined a systemic model of CAN; investigated the impairments of the native CAN protocol to the implementation of those services; provided a protocol suite that handles those problems effectively.

That is a major component of CANELy, the CAN Enhanced Layer, a combination of the CAN standard layer with some simple machinery resources and low-level protocols, achieving reliability, availability and timeliness guarantees needed by highly fault-tolerant real-time systems and applications. The main attributes of CANELy and their comparison with the CAN standard layer and with the industry standard Time-Triggered Protocol (TTP), are summarized in Figure 11. The attributes of CANELy are the consequence of a continuous research effort [22, 18, 15, 17, 16], aimed at defining a CAN-based infrastructure capable of supporting highly fault-tolerant real-time distributed systems.

The results summarized in Figure 11 dismiss ideas that CAN is not suited for hard real-time systems with high

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	yes
Media redundancy	no	no	yes
Channel redundancy	yes		yes (optional)
Babbling idiot avoidance	bus guardian	not provided	
Communications	broadcast	broadcast	broadcast/ multicast
Membership	provided	not provided	tens of ms latency
Clock synch.	in μs range		tens of μs precision

Figure 11. Comparison of TTP, CAN and CANELy

dependability requirements, and open the way to the utilization of CAN in highly fault-tolerant hard real-time systems (babbling idiot avoidance has further been studied in [2]). Other architectural attributes, such as replica determinism and temporal composability, are system level concerns that can be built on top of the CANELy reference architecture.

References

[1] H. Boterenbrood. *CANopen high-level protocol for CAN-bus*. NIKHEF, Amsterdam, Mar. 2000.

[2] I. Broster and A. Burns. The babbling idiot in event-triggered real-time systems. In G. Fohler, editor, *Proceedings of the Work-In-Progress Session, 22nd Real-Time Systems Symposium, YCS 337*, pages 25–28, United Kingdom, Dec. 2001. IEEE, Department of Computer Science, University of York.

[3] CAN in Automation. *CAN Physical Layer for Industrial Applications - CiA/DS102-1*, Apr. 1994.

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

[5] F. Cristian. Agreeing on who is present and who is absent in a synchronous distributed system. In *Digest of Papers, The 18th International Symposium on Fault-Tolerant Computing*, pages 206–211, Tokyo - Japan, June 1988. IEEE.

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

[7] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. Mullender, editor, *Distributed Systems*, ACM-Press, chapter 5, pages 97–145. Addison-Wesley, 2nd edition, 1993.

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

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

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

[11] H. Kopetz, G. Grunsteidl, and J. Reisinger. Fault-tolerant membership service in a synchronous distributed real-time system. In *Proceedings of the IFIP Int'l Working Conference on Dependable Computing for Critical Applications*, pages 167–174, Sta Barbara - USA, Aug. 1989. IFIP.

[12] 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*, Shantou, China, June 1998.

[13] OSEK/VDX Working Group. *OSEK/VDX Network Management Concept and Application Programming Interface*, May 2000. (Version 2.51).

[14] P. Pleinevaux and J. Decotignie. Time critical communication networks: Field buses. *IEEE Network*, 2(3):55–63, May 1988.

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

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

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

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

[19] J. Thomesse and M. Chavez. Main paradigms as a basis for current fieldbus concepts. In D. Dietrich, P. Neumann, and H. Schweinzer, editors, *Fieldbus Technology - System Integration, Networking and Engineering*, pages 2–15. Springer, Sept. 1999. (Proceedings of the Fieldbus Conference FeT'99, Magdeburg, Germany).

[20] K. Tindell and A. Burns. Guaranteeing message latencies on Controler Area Network. In *Proceedings of the 1st International CAN Conference*, pages 1.2–1.11, Mainz, Germany, Sept. 1994. CiA.

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

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

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