Centro de Sistemas Telemáticos e Computacionais

Instituto Superior Técnico

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

Hard Real-Time Operation of CAN

CSTC Technical Report RT-97-02

José Rufino, Paulo Veríssimo

January 1997

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

Hard Real-Time Operation of CAN

Prepared in middle of 1996 while the authors were researchers at INESC

Technical Report: CSTC RT-97-02
Authors: José Rufino, Paulo Veríssimo
Date: January 1997

Avenida Rovisco Pais - 1096 Lisboa Codex - PORTUGAL, Tel: +351-1-8418397/99, Fax: +351-1-8417499.
NavIST WWW Page URL - http://pandora.ist.utl.pt.

Hard Real-Time Operation of CAN

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

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

Abstract

Continuity of service and bounded and known message delivery latency, are reliability requirements of a number of real-time applications. One key issue with this regard, is that networks are subject to periods of inaccessibility. While some are genuine partition failures, others derive from incidents in the LAN or Field-bus protocol operation (e.g. token loss). While both are undesirable, the latter are often disregarded, leading to failures of expected hard real-time properties of the network.

We address this problem by treating both events under a general inaccessibility model, whereby temporary glitches in network operation are allowed, provided that their duration is bounded and known. Most time-critical applications can indeed live with these glitches, provided that they are short and infrequent enough.

In this paper, we do a detailed study on reliable real-time operation of the CAN bus. We show that hard real-time behaviour can be achieved under the inaccessibility model. In the end, we compare the results of previous studies concerning real-time local area networks, and show that CAN exhibits rather short inaccessibility periods.

Keywords: *Control Area Networks, Local Area Networks, Real-time Computing Systems, Communications Reliability, Communications Protocols.*

1 Introduction

In reliable real-time systems, the fundamental requirement of communications is that there be a bounded and known message delivery latency, in the presence of disturbing factors such as overload or faults. For applications where this requirement is very strict (e.g. safety-critical), specialised space-redundant architectures like point-to-point graphs [7] or multiple local area networks [1, 6] are clearly the solution. These architectures are however costly and complex. When that requirement is not so strict, one can find a number of design alternatives for implementing cost-effective network infra-structures, through the use of existing technology. An example of such approach is the DELTA-4 system [8], which uses standardised local area networks (LANs), whose only redundancy may exist in the physical layer.

On the other hand, Controller Area Networks (CAN) are gaining growing acceptance for sensing and actuating in control and automation, an area where cost-effectiveness, continuity of service, and determinism in transmission delays, are requirements of a number of applications. With the adequate techniques, non-replicated LANs and CANs can be used to satisfy reliable

*Instituto Superior Técnico - Universidade Técnica de Lisboa , Avenida Rovisco Pais - 1096 Lisboa Codex - Portugal. Tel: +351-1-8418397/99 - Fax: +351-1-8417499. NavIST CAN WWW Page - http://pandora.ist.utl.pt/CAN.

†Faculdade de Ciências da Universidade de Lisboa, Portugal.

hard real-time requirements. These imply correctness in the time-domain, tolerating faults such as omissions and partitions.

The crux of fault-tolerant operation of a standardised network such as CAN, is its behaviour upon periods of inaccessibility. Some of them correspond to real physical partitions. Others however, are virtual: they are glitches in the protocol operation that prevent communication temporarily, and are neglected in many analysis. One example is the period subsequent to token loss in a token network. In [15], a technique for achieving hard real-time behaviour of non-replicated local area networks is presented. It is based on treating real and virtual partitions under a common inaccessibility model.

Having in mind the goals stated above, this paper provides a framework for reliable hard real-time operation of the CAN bus. Section 2 summarises our technique and its application to the CAN bus. Sections 3 and 4 perform an analysis of CAN inaccessibility and discuss control mechanisms. Section 5 does a comparison with previous LAN inaccessibility studies.

2 Reliable Real-Time Communication

A reliable hard real-time network exhibits bounded and known message delivery latency, in the presence of disturbing factors such as overload or faults.

We consider that the network is not replicated, and use a divide-and-conquer strategy that treats in isolation each of the faults we consider in our model: timing, omission, inaccessibility.

- **1 -** enforce bounded delay from request to transmission of a frame[1], given the worst case load conditions assumed;
- **2 -** ensure that a message[2] is delivered despite the occurrence of omissions;
- **3 -** control inaccessibility.

Condition 1 aims at handling timing faults due to overload. It assures that any frame is sent within a known time bound, even if it does not arrive. Condition 2 has been introduced to tolerate omission faults and ensures that a message is delivered, even if that implies transmission of several frames. Achieving condition 3 is the fundamental issue behind a simplex (non-replicated) hard real-time network. It will be the central issue of this paper.

The Abstract Network Model

The conditions expressed above are sufficient to achieve fault-tolerant hard real-time communication. A model fulfilling those conditions, as well as other important characteristics[3], has been thoroughly discussed in [15]. This model, termed *abstract network*, synthesizes a set of common network properties abstracting from the network's physical characteristics, based on which upper layer protocols can be built. Table 1 presents the sub-set of time-related properties, relevant for our discussion.

The broadcast networks we are considering are made from several components. The network channel may omit to deliver frames (omission errors) due to failures in those components. Omission errors may have many origins: mechanical defects in the cable, EMI corruption of a passing frame, modem synchrony loss, receiver overrun, transmitter underrun, etc. If it is possible to make assumptions about the number of components with failures during an arbitrary interval

[1] Network level information packet.
[2] User level information packet.
[3] Such as recognition of *urgency*.

An1 - Bounded Omission Degree: In a known interval T_{rd}, omission errors may affect at most k transmissions.
An2 - Bounded Transmission Delay: Every frame queued at a network access point, is transmitted within a bounded delay $T_{td} + T_{ina}$.
An3 - Bounded Inaccessibility: In a known interval T_{rd}, the network may be inaccessible at most i times, with a total duration of at most T_{ina}.

Table 1: Abstract Network Properties

of concern, T_{rd}, and the number of consecutive omission errors produced by a given component (omission degree - Od). Property **An1** can be justifiably defined and represents the foundation of the abstract network error processing protocols [15, 16].

Property **An2** specifies a maximum transmission delay, which is T_{td} in the absence of faults. It depends on the particular network, its sizing, parameterising and loading conditions. This issue has been studied in the literature, not only for LANs [5, 4], but also for CAN [13]. On the other hand, T_{ina} is added to account for the *inaccessibility* time, yielding the worst-case figure. Inaccessibility and its control specifically concern property **An3**, and will be discussed at some length in the sequel of the paper.

Controlling Partitions: Inaccessibility

Let a network be partitioned when there are subsets of the nodes which cannot communicate with each other[4]. This is the classical definition, and it is normally attached to a notion of physical disconnection. However, since the network is not replicated, there are a number of causes for partition as defined above, some of which are virtual rather than physical: bus medium failure (cable or tap defect), ring disruption, transmitter or receiver defects/perturbations; reformation of LAN logical ring, multiple station joins; active monitor loss, token loss; etc. Some standard networks have embedded means of recovering from some of the situations described above (e.g. token regeneration for token-based LANs), but since this recovery process takes time, the network will exhibit periods where no service is provided.

If this is not tolerated, timeliness of applications is at stake, which is not acceptable in a hard real-time system. In consequence, a semantics of *inaccessibility* fault must be defined, so that we may devise adequate fault-tolerance techniques[15]:

Inaccessibility*:*

i) *a component* refrains *from providing service;*
ii) *its users perceive that state;*
iii) *inaccessibility limits (duration, rate) are specified;*
iv) *violation of those limits implies permanent failure of the component.*

The approach taken to tolerate inaccessibility is based on the following idea: if one knows for how long it lasts, then reliable real-time operation of the system is possible, provided that those glitch periods are acceptably short, with regard to service requirements. The conditions

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

underlying the techniques to implement such tolerance, discussed in the following section, are the following:

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

Uncontrolled partitions are of course still possible, because systems do fail, but that event means the total and permanent failure of the real-time communications subsystem. With the methodology we have just described, **An3** is attained, and a network exhibiting this property satisfies the third and last condition to achieve reliable hard real-time behaviour.

3 Implementing Inaccessibility Control

Given the concept and shown it is the foundation for reliable real-time operation of simplex networks, design criteria and protocols to set-up an abstract network fulfilling **An3** are presented next, as applied to the CAN bus.

First, one must recover from all conditions leading to partition. Thus, recovering from physical – mechanical or electrical – partition by means of medium and physical layer redundancy is required. For example, this is assured in standard FDDI, through a dual-reconfiguring ring, capable of surviving one interruption of the ring [2]. In other networks, without standardised redundancy provisions, some similar measures should be custom-implemented. For example, in [14] it is described a glitch-free method for real-time switch-over between buses of a dual-media token-bus.

Afterwards, one needs to show that all the recovery glitches are time-bounded and determine the upper bound. Those include operating situations intrinsic to each network (e.g. addition of a new member to a logical ring) and recovery actions resulting from failures (e.g. physical reconfiguration subsequent to ring breakage). Boundedness of recovery glitches will be thoroughly discussed, later in this paper. To illustrate the required procedure, we reserve Section 4 for an inaccessibility analysis of the CAN. A comparison between CAN and other LANs inaccessibility, addressed in previous studies [9, 11, 12], will also be discussed.

Finally, inaccessibility must be included in the timeliness model. Let us consider a system only with local clocks (timers), used to implement timeouts. Timeouts detect timing, omission and crash failures in the abstract network protocols. They can also be used by upper-layer protocols to perform surveillance of remote parties, upon waiting for the reply to a request (e.g. RPC[5]). Inaccessibility must be accounted for in the calculation of real execution times (for real-time processing purposes) and in dimensioning of timeouts.

According to **An3**, in an interval of concern, T_{rd}, there may be at most i inaccessibility periods adding-up to at most T_{ina}. Let us assume that T_{rd} is the longest protocol execution duration. Calculation of real worst-case execution times of any protocol simply implies that T_{ina} be added to the worst-case protocol execution time expression computed under normal conditions (i.e. without inaccessibility).

Timeout values are set in function of execution times. Low-level and upper-layer protocol timers must in general include T_{ina}, or else they could timeout too early should inaccessibility occur, which may lead to protocol failure. At this point, note that incorporating T_{ina} in the timeout

[5] Remote Procedure Call.

values is a sufficient condition for running synchronous (real-time) timeout-based protocols over the abstract network. Protocols that do not use timeouts may require additional measures [16].

However, T_{ina} may be much greater than T_{td}, causing the timeouts to be undesirably long. A solution taking away inaccessibility from the timeouts would be welcome, with two obvious advantages: giving the programmer the simple and elegant abstraction of a distributed environment which is always connected; yielding shorter timeouts. Two engineering techniques making this possible, presented in [16], are re-iterated here.

TIMER-FREEZING MECHANISM

The mechanism proposed to hide inaccessibility is very simple. It concerns all timeouts controlling timeliness of remote interactions, be them in low-level or upper-layer protocols. The value of the timeouts does not include inaccessibility. The mechanism is based on stopping (freezing) all timers when inaccessibility is detected and while it lasts, and restarting them when the network becomes accessible again.

All protocols can then be constructed as if the network were always accessible. The worst-case frame transmission time in all situations, from the programmer or protocol designer's point of view[6], is merely the maximum queue plus access plus transmission delay, T_{td}, expected in the absence of inaccessibility. This yields an optimal sizing of timers, with regard to detection of other errors, such as omission and node failures.

On the other hand, worst-case timeliness calculations, i.e. determining the real duration of activities, are tuned by adding T_{ina} – which must be determined for each specific network – to the computed protocol execution time expressions.

Freezing the timers is an engineering issue, which implies: inaccessibility to be detected by the communications adapter part; information between the adapter and the timers to flow quickly enough; timers to be stopped/resumed on arrival of in-/accessibility notifications.

While this mechanism can be easily implemented on specially built hardware, the conditions named above are not always met in existing hardware/software platforms. Namely: the timer package may not have the functionality for lightweight start-restart; some LAN controllers do not signal all inaccessibility situations, or do not do it consistently at all nodes.

INACCESSIBILITY TRAPPING MECHANISM

Since implementing inaccessibility control in existing environments is a requirement, an alternative had to be seeked. We devised a mechanism which masks inaccessibility at the network interface level. It merely requires fulfillment of the following system requirements:

- **i)** in the interface with the abstract network, all requests are positively confirmed *when the frame is transmitted*;
- **ii)** in a layered architecture: the higher-level request is also confirmed; the low-level confirmation that the request was served propagates up the layers;
- **iii)** timers to control a remote request are only started after the confirmation that it was issued.

If the network becomes inaccessible, it refrains from providing service, and confirmations do not come. Let us discuss the implications of this fact in system activity by following the several situations depicted in Figure 1, where timer $TwaitRemote$ equals the desired waiting time at any level of the system (to receive a RPC result, or a transmission reply) that, as in the previous method, do not include inaccessibility:

[6] I.e. for the sake of dimensioning timeouts.

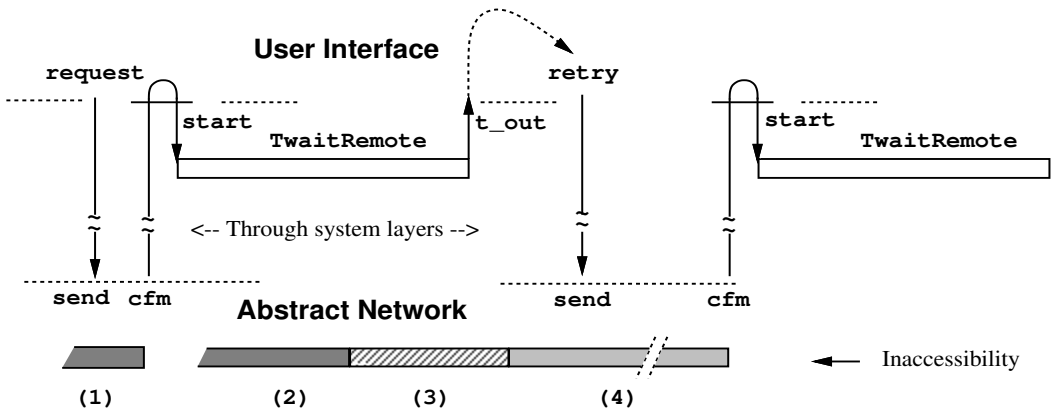

Figure 1: Timing of the Inaccessibility Trapping Mechanism

Situation (1): since a remote interaction is started by a **send** request, a timer controlling it at whatever level will be pending of the confirmation from the network. Should it be inaccessible, timer start will be postponed until the network becomes accessible again.

Situation (2): after the timer is started, it will count up to $TwaitRemote$. In this situation, the network becomes inaccessible after the timer is started and accessible again before it expires. If there is not enough time for the remote frame to arrive, the timer will expire.

Situation (3): inaccessibility lasts beyond $TwaitRemote$, but ends before anything is attempted in order to recover. A timeout is issued here as well.

Situations (2) and (3) are analogous. They are not disturbing: the timer is there to detect these facts. Inaccessibility has just been transformed into an omission. Given **An1** and **An3**, whatever methods used to take care of k omissions at a given system level, will keep working for $K = k + i$ (i - number of inaccessibility occurrences in a known interval, see **An3**). This means that inaccessibility is in fact masked out of the system algorithms above the abstract network. In fact, the system programmer can be given the consolidated K omission bound figure.

Note the very little difference between the effect of inaccessibility in situations (2) or (3), and that of a slightly longer LAN access time. In fact, "inaccessibility" would not be needed for this, it would suffice to make timers a little longer. However, this approach is justified by a bi-modal network characteristic, with the more serious inaccessibility situations lasting much longer than T_{td} and thus $TwaitRemote$. That is the case of:

Situation (4): inaccessibility still lasts when recovery is attempted (the second send request).

The protocol failures due to short timers [16] we have mentioned earlier, occur because the protocols might proceed during an inaccessibility period. In order for this mechanism of *short timers* and *transforming inaccessibility into omissions* to work, it is necessary that each inaccessibility period is seen as *one* "omission". The confirmed send request performs this role, preventing the protocol from proceeding until the period ends: each inaccessibility period is therefore *trapped* inside two consecutive transmission *send/confirmation* signal pairs issued at the abstract network interface. Note also why the duration of inaccessibility does not affect system timers: it is either simultaneous with the $TwaitRemote$ period, or with the request-to-confirmation period.

On the quantitative side, execution times are not increased by T_{ina} anymore when faults occur (the case with including T_{ina} in the timer values). Besides, when inaccessibility occurs, in lieu of increasing by T_{ina}, they increase *at most* by t_{ina} (the effective duration, which may be much lower than T_{ina}).

The disadvantage with regard to the timer freezing scheme, which "stops" the time in this part of the system, is that timeouts do occur due to inaccessibility, and recoveries have to be initiated.

4 CAN Inaccessibility Analysis

This section will be entirely devoted to a detailed study of CAN inaccessibility, extending the results advanced in [10]. The analysed scenarios are the causes for inaccessibility foreseen in the standard specification and this procedure allow us to show that glitches in CAN operation are time-bounded and determine the corresponding upper bound.

The Controller Area Network (CAN) is a broadcast bus with a multi-master architecture [3]. The transmission medium is usually a twisted pair cable. The network maximum length depends on data rate. Typical values are: 40m @ 1 Mbps; 1000m @ 50 kbps.

Bit transmission takes two possible representations: *recessive*, which only appears on the bus when all the nodes send recessive bits; *dominant*, which needs only to be sent by one node to stand on the bus. This means that a dominant bit sent by one node can overwrite recessive bits sent by other nodes. This feature is exploited for bus arbitration. A given bit-stream is transmitted using the NRZ[7] code. Data transmission is subject to a bit-stuffing technique that prevents more than five consecutive bits of identical polarity to be transmitted, through automatic insertion of a complementary bit.

Accordingly with CAN terminology, data to be transferred is encapsulated within "communication objects": a unique identifier is assigned to each communication object. The Controller Area Network [3] is a *carrier sense multi-access with collision avoidance* network: nodes delay transmissions if the serial bus-line is in use; when a bus idle condition is detected, any node may start transmitting; bus access conflicts are resolved through comparison of communication object identifiers and work as follows:

- while transmitting a communication object identifier, each node monitors the serial bus-line;
- if the transmitted bit is recessive and a dominant bit is monitored, the node gives up from transmitting and starts to receive incoming data;
- the node transmitting the object with the highest identifier goes through and gets the bus.

That means: arbitration is *non-destructive*, since transmission of the highest identifier object undergoes no delay; bus access has priorities, allowing transmission of more urgent data to takeover less urgent one. Automatic retransmission of a communication object is provided after a loss in an arbitration process. For our purposes, two important parameters characterise the basic operation of any CAN network:

- ◇ **Data Rate** - The nominal rate of data signalling, on the bus. It gives a meaning to t_{bit}, the nominal duration of a single bit.
- ◇ **Intermission Field Period** - t_{IFS} - The minimum bus idle period that mandatory precedes any data or remote frame transmission.

Only four types of protocol data units are used by CAN. Their functions are described next:

- ◇ **Data Frame** - used for dissemination of communication objects. It begins by an header, that includes the arbitration field, followed by the data, CRC[8] and trailing fields.
- ◇ **Remote Frame** - to explicitly request the dissemination of a communication object. For the same identifier, the data frame takes precedence over the remote transmission request.

[7] Non-Return-to-Zero.
[8] Cyclic Redundancy Code.

◇ **Error Frame** - used for error signalling. It is simply made from an *error flag*, followed by an *error delimiter*.

◇ **Overload Frame** - used to extend the interframe spacing, upon detection of overload conditions. It is simply made from an *overload flag* plus an *overload delimiter*.

Frame	Symbol	Duration (μs)	
		min.	*max.*
Data frame	t_{data}	44.0	132.0
Remote frame	t_{rdata}	44.0	52.0
Error frame	t_{error}	14.0	20.0
Overload frame	t_{oload}	14.0	20.0

Table 2: Duration of CAN frames (Data Rate = 1 Mbps)

The integrity of data and remote frames is checked at each receiver by resorting to a CRC 15-bit sequence, particularly well suited to check the integrity of frames with a total bit length less than 127, thus providing a good error detection coverage (see [3], for further details). CRC generation and bit-stuffing operations are only performed over data and remote frames, but do not include the fixed form sequence that ends both these frames: the 1-bit *CRC delimiter*; the 1-bit *acknowledge slot* plus the 1-bit *acknowledge delimiter*; the seven-bit *end-of-frame* delimiter.

These were the details on CAN operation [3] required ahead for our analysis. Additional issues will be brought-up as necessary.

Accessibility Constraints

The study of CAN inaccessibility is presented next. We start with very simple situations that then evolve to less restrictive – and thus more realistic – operating conditions/fault assumptions. Unless stated otherwise, we assume all the nodes are in the so-called *active* state [3], thus being fully able to participate in error detection/signalling activities. For most of the cases, we explicitly derive best and worst case figures, that we signal with superscripts ^{bc} and ^{wc}, respectively.

BIT ERRORS

Let us start our analysis of CAN inaccessibility by considering that bus operation is disturbed only once, through the corruption of a single bit within a bit-stream, for instance, due to electromagnetic interference. Globally, both transmitting and receiving nodes cooperate in the detection of deviations from normal bus operation.

Transmitter-based error detection aims to supply each sending node with the mechanisms required to detect single-bit modifications, in its own transmitted bit-stream. Such error detection technique lays down on serial bus-line monitoring, performed on a bit-by-bit basis, while transmitting. A recessive level issued on the bus, by a given transmitter, can only be overwritten by a dominant level, without that be considered an error, in the following two circumstances:

- inside the *arbitration field*, where such an event will be interpreted, by the recessive bit sender, as a loss in an arbitration process;
- during the *acknowledge slot*, meaning that the previously transmitted data stream has been heard without errors, at least by one node in the system.

Otherwise, should the monitored level differ from the one transmitted, it will be considered an error. Upon its detection, the error will be signaled on the bus, by starting the transmission of an error frame at the next bit slot.

Bit error detection, as performed by a transmitting node, can take place as soon as the first bit of a data frame is being transmitted. Therefore, the best-case error detection latency equals bit-time, which added to the best time required to signal the error gives the corresponding network inaccessibility duration:

$$t^{bc}_{ina \leftarrow berr} = t_{bit} + t^{bc}_{error} + t_{IFS} \tag{1}$$

On the other hand, corruption of the transmitted bit-stream cannot occur latter than transmission of *end-of-frame* delimiter last bit. The worst-case inaccessibility time, is obtained when considering the transmission of a data frame with maximum size. Its value is expressed through equation:

$$t^{wc}_{ina \leftarrow berr} = t^{wc}_{data} + t^{wc}_{error} + t_{IFS} \tag{2}$$

This transmitter-oriented error detection scheme contributes for the reduction of the average network inaccessibility time, since is designed to detect errors immediately upon their occurrence. It provides an effective answer for the treatment of errors that manifest their effects in sets of nodes that also include the sender. However, it is helpless in the detection of errors affecting only sets of receiving nodes. Thus, the CAN access method includes several receiver-oriented error detection mechanisms, to be discussed next.

BIT-STUFFING ERRORS

As mentioned earlier, in the opening remarks of this section, transmission of data and remote frames bit-streams is performed, until the end of their 15-bit *CRC sequence*, by resorting to a bit-stuffing coding and data transparency scheme.

Thus, frame reception entities must monitor such bit-stream, providing the required bit-destuffing and error detection functions. Whenever a receiving node monitors l_{stuff} consecutive bits of identical level, it automatically deletes the next received (stuff) bit. Under error free operation, the deleted bit presents a polarity opposite to the preceding ones; should this condition be violated, an error will be signaled on the bus, by starting the transmission of an error frame. The earliest moment, within a given bit-stream reception, where a bit-stuffing error can be detected is while sampling the bit following the first l_{stuff} bits of each frame. Therefore, the best-case duration of an inaccessibility period, due to a bit-stuffing error, is given by equation:

$$t^{bc}_{ina \leftarrow stuff} = (l_{stuff} + 1) \, . \, t_{bit} + t^{bc}_{error} + t_{IFS} \tag{3}$$

On the other hand, a bit-stuffing error cannot occur after reception of the 15-bit *CRC sequence*. The worst-case inaccessibility duration, for this particular scenario, must consider the transmission of a maximum size data frame. Having both these aspects in mind:

$$t^{wc}_{ina \leftarrow stuff} = t^{wc}_{data} - t_{EFS} + t^{wc}_{error} + t_{IFS} \tag{4}$$

where t_{EFS}, accounts the duration of the fixed form sequence – not subject to bit-stuffing coding – that ends every data or remote frame.

Bit-stuffing errors can be due to some kind of bit corruption, occurring in a way that a sequence containing more than the allowed l_{stuff} consecutive bits of identical polarity, is generated. This

type of errors are always detected by bit-stuffing monitoring. However, the same kind of interference may falsify one or more bits in the sequence that has originally led to stuff bit insertion. In this case, bit-stuffing monitoring only reacts to the presence of the fault if the non-removal of the stuffed bit leads to a later bit-stuffing error. Otherwise, the error will be detected only when the end of frame sequence is received, either by CRC checking or through detection of a frame format violation.

CRC ERRORS

The CAN protocol uses a 15-bit *CRC sequence* plus a 1-bit *CRC delimiter*, transmitted with each data and remote frames, to check correctness of information transfer. After checking frame correctness, a receiving CAN node should take into account that:

- if the frame has been heard without errors, this condition should be signaled to the transmitter, by changing the bus level from recessive to dominant, during the *acknowledge slot* [9].
- otherwise, the receiving node should take no action during the *acknowledge slot*. The detected CRC error is signaled on the bus through the transmission of an error frame, but this action only starts at the bit following the *acknowledge delimiter*.

The corresponding inaccessibility time is generically given by expression:

$$t_{ina \leftarrow crc} = t_{data} - t_{EOF} + t_{error} + t_{IFS} \tag{5}$$

where t_{EOF} represents the duration of the *end-of-frame* delimiter.

The best and worst-case bounds of equation (5) are obtained considering, respectively, the transaction of the shortest and of the longest data frames, as well as error frame variability.

FORM ERRORS

All the frames used by the CAN protocol obey to a few pre-defined formats. For data or remote frames, the origin of the frame ending sequence is established taking into account the length specified in the *control field* [3]. The fixed form sequence that ends both these frames includes specific elements, transmitted on the bus with a recessive level and whose value should not be changed by any other node in the system. These elements are:

- the *CRC delimiter*,
- the *acknowledge delimiter*,
- the *end-of-frame* field, transmitted as a sequence of seven consecutive *recessive* bits.

Likewise, error and overload frames ending delimiters also obey to a fixed form rule: a sequence of eight consecutive recessive bits should follow the last dominant bit of error and overload flags. Violation of those frames fixed form usually only occur as a consequence of multiple errors, happening within a given time period. So, their study will be postponed to less restrictive scenarios, where such behaviour is allowed.

Under a single-fault assumption, one needs to consider only the violation of data and remote frames fixed form ending sequence. In the best-case, such errors can be detected while receiving

[9] Ahead, we will discuss what happens when all the nodes in the system detect a CRC error, thus binding the *acknowledge slot* value to its transmitted recessive level.

the 1-bit *CRC delimiter* of a minimum size data frame. The corresponding inaccessibility period is given by:

$$t^{bc}_{ina \leftarrow form} = t^{bc}_{data} - t_{EFS} + t_{bit} + t^{bc}_{error} + t_{IFS} \tag{6}$$

Conversely, a form error may occur, at the latest, while receiving the penultimate bit of the *end-of-frame* delimiter, because a receiver monitoring a *dominant* level at the last bit of this delimiter does not take that as a form error[10]. Evaluation of the worst-case inaccessibility time, for this particular scenario, must considerer the transaction of a maximum size data frame. With that understood:

$$t^{wc}_{ina \leftarrow form} = t^{wc}_{data} - t_{bit} + t^{wc}_{error} + t_{IFS} \tag{7}$$

ACKNOWLEDGE ERROR

All CAN nodes receiving a data or remote frame without CRC errors should signal this condition to the sender, by transmitting a dominant bit level on the bus, during the *acknowledge slot*. This action should be performed regardless of the acceptance filter test result, i.e. signalling is due either when the frame is about to be discarded, on account of a reject result, or when it is to be actually accepted and delivered to network user-level entities.

Whenever a transmitter does not monitor a dominant level on the bus during the *acknowledge slot*, it interprets that as an error, and the transmission of an error frame is started at the next bit. So, the duration of the corresponding network inaccessibility period is generically given by equation:

$$t_{ina \leftarrow ack} = t_{data} - t_{EFS} + 2\,.\,t_{bit} + t_{error} + t_{IFS} \tag{8}$$

As in a previous scenario, the difference between best and worst-case bounds depends only on the variability of data and error frame durations.

In the presence of multiple bit disturbances, it may happen the sender monitors a falsified dominant level, during the *acknowledge slot*, instead the correct recessive level, binded on the bus as a consequence of former CRC errors, detected by all the nodes in the system. The transmitting node lacks to detect such error, but bus operation disturbance will be signaled anyway, as a CRC error, by receiving nodes not acknowledging frame reception.

OVERLOAD ERRORS

For correct operation, the CAN protocol requires data and remote frames to be separated from each other and from any other frame by a minimum amount of three recessive bits, known in CAN terminology as *intermission field*. However, the next data or remote frame transmission can be explicitly delayed, whenever the receiver circuitry is not ready to receive those frames.

That is achieved through transmission of a special data unit known, in CAN terminology, as *requested overload* frame. Transmission of such frame is only allowed to be started at the first bit of an expected intermission and, at most, two overload frame transmissions may be consecutively requested, in the sequence of the same data or remote frame reception. Taking into account the facts laid above, the corresponding lower and upper inaccessibility bounds, is simply given by:

$$t^{bc}_{ina \leftarrow oload} = t^{bc}_{oload} \tag{9}$$

[10] As we will see ahead, this will be considered a *reactive overload error*. On the other hand, we assume that whenever the transmitter monitors this same event, it will be interpret as a *bit error*.

$$t^{wc}_{ina \leftarrow oload} = 2 \, . \, t^{wc}_{oload} \tag{10}$$

REACTIVE OVERLOAD ERRORS

There is another use foreseen in the CAN specification for *overload frames*. It concerns the signalling of certain error conditions. Transmission of the so-called *reactive overload* frames are due whenever a *dominant* bit is detected either within the *intermission field* or at the last bit of the *end-of-frame* delimiter. Signalling of such CAN protocol violations is always started one bit after its detection.

The best-case inaccessibility bound assumes that the transmission of a single overload frame is started at the first bit of the expected *intermission field*, thus:

$$t^{bc}_{ina \leftarrow roload} = t^{bc}_{oload} \tag{11}$$

On the other hand, the disturbance causing the *reactive overload* frame transmission may be only detected at the third bit of the *intermission field*. In the presence of single disturbances, the corresponding worst-case inaccessibility bound will be given by:

$$t^{wc}_{ina \leftarrow roload} = t^{wc}_{oload} + t_{IFS} \tag{12}$$

OVERLOAD FRAME FORM ERRORS

As aforementioned, the CAN protocol signals the absence of a minimum bus idle period, preceding the transmission of data and remote frames, through the issuing of an overload frame.

The CAN receiver detecting the overload condition starts, at the next bit slot, the transmission of an *overload flag*, consisting of six dominant bits; all other nodes will react to this action, and as a consequence, also initiate the transmission of an *overload flag*.

After transmitting its own *overload flag* each CAN node monitors the bus-line until a transition from a dominant to a recessive bit level is detected. Under normal circumstances[11], such an event means that the first of an eight-bit *overload delimiter* has just been issued. All CAN nodes will then transmit the remaining seven recessive bits of the fixed form *overload delimiter*.

Should a CAN node detect any deviation from the aforementioned overload frame fixed form, it will be considered an error and accordingly with the CAN protocol, the transmission of an error frame is started at the next bit slot.

The best-case inaccessibility bound for this particular scenario is obtained under the assumption that overload frame fixed form violation occurs at the first transmitted bit. Therefore:

$$t^{bc}_{ina \leftarrow oform} = t_{bit} + t^{bc}_{error} \tag{13}$$

On the other hand, for the evaluation of this scenario worst-case inaccessibility bound, we considerer the transmission of two consecutive overload frames, with the fixed form violation occurring only at the last bit of the *overload delimiter* transmitted in second place. The corresponding inaccessibility time is then given by:

$$t^{wc}_{ina \leftarrow oform} = t^{wc}_{ina \leftarrow oload} + t^{wc}_{error} \tag{14}$$

[11] That is, assuming the recessive level does not result from bit corruption.

INCONSISTENT OVERLOAD ERROR

In this scenario we considerer that only a subset[12] of all the nodes in a given CAN network detect, either correctly or erroneously, a dominant level in the third bit of the *intermission field*.

The set of nodes detecting such event initiate then the transmission of a *reactive overload* frame, but this action will be perceived in a different manner by the set of nodes that did not monitored the *intermission field* violation, because the first of the six dominant bits that made up the *overload flag* will be interpreted as a *start-of-frame* delimiter, by such nodes.

Under the assumption that the entire *overload flag* is correctly received by all network nodes, the sixth dominant bit will violate the bit-stuffing rule and cause an error condition signalling. In these circumstances, the inaccessibility time for this scenario equals the one given by equation (3):

$$t^{bc}_{ina \leftarrow iload} = (l_{stuff} + 1) \, . \, t_{bit} + t^{bc}_{error} + t_{IFS} \tag{15}$$

In a less favourable fault scenario, some bits within the overload frame may also get corrupted. Under such circumstances, there are no guarantees that bit-stuffing monitoring is able to detect the disturbance in bus operation, since a sequence containing more than l_{stuff} consecutive bits of identical polarity, may never be received by the relevant CAN nodes. Nevertheless, as pointed out before in this paper, the error will still be detectable either by CRC checking or through detection of a frame format violation.

For the evaluation of the worst-case inaccessibility time for this scenario, we take the largest of those error detection bounds and additionally assume that the inconsistency in *intermission field* monitoring only occurs after the extension of the *interframe space* trough two consecutive *requested overload* frames. Therefore:

$$t^{wc}_{ina \leftarrow iload} = 2 \, . \, t^{wc}_{oload} + t^{wc}_{data} - t_{bit} + t^{wc}_{error} + t_{IFS} \tag{16}$$

MULTIPLE CONSECUTIVE ERRORS

As previously mentioned, CAN nodes signal errors through the transmission of an error frame: as soon as a node detects an error condition it starts transmitting an *error flag*, consisting of six consecutive dominant bits; all other nodes will react to this action, and in consequence, also initiate the transmission of an *error flag*. After transmitting its own *error flag* each CAN node monitors the bus-line until a transition from a dominant to a recessive bit level is detected. In the absence of further errors, this means that the first of an eight bit *error delimiter* has just been issued. All CAN nodes will then transmit the remaining seven recessive bits of the fixed form *error delimiter*.

For most of the scenarios we have analysed so far, disturbances in CAN bus operation have been restricted to single errors. The bounded omission degree (**An1**) assumption, defined in Section 2, allow us to withdraw this restriction, by establishing an upper-bound for inaccessibility duration, when in the presence of multiple errors: no more than n error and/or reactive overload frame transmissions are required to recover from errors[13].

Furthermore, let us assume that all the n errors are consecutive in the network[14]. In this case, error signalling will also be subject to disturbances: any node detecting a deviation from the error frame fixed form, starts the transmission of a new error frame. Deriving the best-case inaccessibility bound, under these conditions, assumes that errors, in the forerunner data frame

[12] This subset may be restricted to a single node.

[13] Under given circumstances, CAN data frame losses are masked-off at the MAC layer, thus rendering the user-level omission degree bound, k, different from the network-level omission degree bound, n.

[14] Meaning they are not interleaved with good transmissions.

and in the subsequent *error flag*, are both detected at the first transmitted bit; a second error frame is correctly received by all CAN nodes. In consequence:

$$t^{bc}_{ina \leftarrow mcerr} = 2 \, . \, t_{bit} + t^{bc}_{error} + t_{IFS} \tag{17}$$

Conversely, for the worst-case scenario, we consider that the first error occurs at the end of a maximum length data frame transmission and that the following $(n-1)$ errors are only detected at the end of a maximum duration error signalling period. The recovery actions end with the successful transmission of the n^{th} error frame. The time elapsed in these operations is given by:

$$t^{wc}_{ina \leftarrow mcerr} = t^{wc}_{data} + n \, . \, t^{wc}_{error} + t_{IFS} \tag{18}$$

MULTIPLE SUCCESSIVE ERRORS

Let us now assume that errors occur in a way that only data or remote frame transmissions are affected; the corresponding error signalling, always succeeds. This scenario is realistic enough to be considered: a failure in a transmitter may lead to this behaviour.

The corresponding worst-case inaccessibility bound directly results from application of **An1**, taking into account the worst-case durations of data and error frames:

$$t^{wc}_{ina \leftarrow mserr} = n \, . \, (t^{wc}_{data} + t^{wc}_{error} + t_{IFS}) \tag{19}$$

ERROR CONFINEMENT MECHANISMS

The error confinement mechanisms provided by the CAN protocol are based on two different error counters, that at each node, record transmit and receive errors. These counters have a non-proportional update method, with each error causing a counter increase larger than the decrease resulting from a successful data or remote frame transfer. Details on *Transmit* and *Receive-Error-Count* update procedures can be found in [3].

The rules used in error counting have been defined in order that nodes closer to the error-locus will experience, with a very high probability, the highest error count increase. This way, disturbances due to a faulty node can be localised and their influence restricted:

Error-Active - the normal operating state. Such a node is able to transmit and receive frames and in both operations fully participates in error detection/signalling activities.

Error-Passive - the node is still able to transmit and receive frames, but: after transmitting a data or remote frame the node requires an extra eight bit bus idle period following the *intermission field*, before it can start a new transmission[15]; error signalling is performed through transmission of a *passive error flag*, made up of six recessive bits, instead the normal *active error flag*.

Bus-Off - a node in this state does not participate in any bus activity, being unable to send or receive frames.

In order to complete our analysis of CAN inaccessibility, we proceed with the study of two examples, where the occurrence of permanent failures is assumed.

[15] An action known in CAN terminology as *Suspend Transmission*. If during this period another node starts transmitting, the suspended CAN node will become receiver of that frame.

FAILED TRANSMITTER

Let us analyse in first place, a scenario where the transmitter of a given error-active CAN node fails in a way that errors only affect the data and remote frames it sends. Such selective error pattern may be due, for instance, to a malfunction in the node's CRC generator.

Accordingly with the aforementioned error confinement policies: the transmitter failure may only systematically affect bus activity until the node enters the error-passive state; the number of transmission attempts causing this transition is bounded by:

$$n_{txfail} = \left\lceil \frac{err_{a\_thd}}{\Delta_{txerr}} \right\rceil \tag{20}$$

where $\lceil \ \rceil$ represents the *ceiling* function[16]; err_{a_thd} is the error-active count threshold and Δ_{txerr} accounts the *Transmit-Error-Count* increase produced by each transmission error [3].

Consecutive transmissions issued by the failed transmitter appear in the network interleaved with error frames and occasionally with good transmissions from other nodes. The longest inaccessibility period occurs in the absence of these last events and is obtained applying expression (20) to equation (19):

$$t^{wc}_{ina \leftarrow txfail} = n_{txfail} \cdot (t^{wc}_{data} + t^{wc}_{error} + t_{IFS}) \tag{21}$$

FAILED RECEIVER

When an error-active CAN receiver is affected by a similar failure, i.e. a malfunction in the CRC checker, errors are also successive in the network because, once again, only data and remote frame transfers are affected; error signalling succeeds without errors. However, any data/remote transfer does not succeed until the defective node becomes error-passive, which occurs after the following attempts:

$$n_{rxfail} = \left\lceil \frac{err_{a\_thd}}{\Delta_{rxer1} + \Delta_{rxer2}} \right\rceil \tag{22}$$

where the terms under the fraction represent two different contributions for *Receive-Error-Count* increase. They derive from the application of more than one error counting rule to the same data transfer, as defined in [3]. The time required to recover the network is thus obtained using expression (22) in equation (19). Thus:

$$t^{wc}_{ina \leftarrow rxfail} = n_{rxfail} \cdot (t^{wc}_{data} + t^{wc}_{error} + t_{IFS}) \tag{23}$$

5 Analytic results and comparison with other LANs

For consolidation of our study regarding accessibility constraints we now evaluate inaccessibility time bounds, for a given CAN network, for example, a 32 node CAN field-bus, in an industrial environment, for real-time sensing and actuating. The data rate is $1Mbps$ and we assume a moderate value ($n = 3$) for the *omission degree* bound due to medium errors. The results of the evaluation, for each one of the studied scenarios, are summarised in Table 3. We see that, exception made to the rather serious transmitter or receiver failures, inaccessibility is limited to $500\mu s$. While the figure is not high, it may come as a surprise, if compared with performance

[16] The *ceiling* function $\lceil x \rceil$ is defined as the smallest integer not smaller than x.

studies not concerned with fault-tolerant operation of the CAN bus. To achieve reliable hard real-time communication in CAN, the worst-case figure of table 3 must be included in the timeliness equations of the mechanisms presented in Section 3.

Data Rate - $1Mbps$		
$Scenario$	t_{ina} (μs)	
	$min.$	$max.$
Bit Errors	18.0	155.0
Bit-Stuffing Errors	23.0	145.0
CRC Errors	54.0	148.0
Form Errors	52.0	154.0
Acknowledge Errors	53.0	147.0
Overload Errors	14.0	40.0
Reactive Overload Errors	14.0	23.0
Overload Form Errors	15.0	60.0
Inconsistent Overload Errors	23.0	194.0
Medium Consecutive Errors ($n = 3$)	19.0	195.0
Medium Successive Errors ($n = 3$)	-	465.0
Transmitter Failure	-	2480.0
Receiver Failure	-	2325.0

Table 3: CAN Inaccessibility Times

The inaccessibility analysis we have just presented provides a better understanding of CAN performance in the presence of failures, providing guidance on how to justifiably achieve better performability and thus, better respect of bounded delay requirements. The figures derived should be taken as corrective terms for computing transmission delays in various situations.

In this sense, it will be interesting to compare the inaccessibility figures obtained for CAN with the ones obtained in previous studies [9, 11, 12], regarding real-time LANs with the same number of nodes. These results are summarised in Table 4 where, for each LAN, we have listed, along with best and worst-cases, a relevant set of other scenarios.

The main conclusion is that CAN displays inaccessibility durations much lower than any other real-time LAN.

6 Conclusions

To achieve reliable real-time operation on a given network, a bounded delay requirement must be met. Usually this analysis is performed by computing worst-case access/transmission delays, for normal network operation. However, achieving the bounded delay requirement for hard real-time communication means, amongst other factors, ensuring continuity of service. Networks, CAN included, are subject to failures: if these are not controlled, the above mentioned requirement is not met.

While the common solution lies in fully-replicating the network, and thus ensuring that at least one of the replicas meets the normal no-fault bounds, field buses such as CAN are aimed at unexpensive solutions, which should avoid replication. In this paper, we presented an alternative solution based on simplex CAN networks.

The main problem with this approach is inaccessibility: partitions, of physical and virtual nature, impair the network timeliness. Recovery from these situations takes time, and that is often disregarded by designers when expecting hard real-time behaviour from a system.

$Analysed Scenario$	t_{ina} (ms)	
	$min.$	$max.$
ISO 8002/4 Token-Bus $(5Mbps)$		
Station Join	0.07	4.67
Multiple Joins	0.37	135.72
Station Leave	0.06	0.06
Token Loss	1.74	5.90
ISO 8002/5 Token-Ring $(4Mbps)$		
Stripping Errors	4.6	4.7
Token Loss	14.6	14.7
Active Monitor Loss	1044.9	15068.4
Station Join/Leave	14.6	14.7
Streaming Receiver	28255.2	28278.3
ISO 9314 FDDI $(100Mbps)$		
No Valid Transmissions	2.53	2.76
No Valid Tokens	15.03	15.26
Station Join	30.03	30.24
Station Leave	20.03	20.24
Streaming MAC Receiver	9457.18	9457.33

Table 4: LAN Inaccessibility Times

We have shown how to embed inaccessibility in the communication model of CAN, so as to tolerate such faults. Furthermore, we made an exhaustive study of CAN inaccessibility. Finally, we derived figures for the several CAN inaccessibility situations, showing what the protocol designer can count on, in absolute terms. On the other hand, when compared to several standard LANs, CAN exhibits superior performability, in what concerns inaccessibility.

References

[1] F. Cristian. Synchronous atomic broadcast for redundant broadcast channels. Technical Report RJ 7203, IBM Research Division, San Jose, California, April 1990.

[2] X3T9.5 FDDI. *FDDI documents: Media Access Layer, Physical and Medium Dependent Layer, Station Mgt.*, 1986.

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

[4] Raj Jain. Performance analysis of FDDI token ring networks: effect of parameters and guidelines for setting TTRT. In *Proceedings of the ACM-SIGCOM'90 Symposium*, Philadelphia-USA, September 1990.

[5] D. Janetzky and K. Watson. Token bus performance in MAP and PROWAY. In *Proceedings of the IFAC Workshop on Distributed Computer Protocol System*. IFAC, 1986.

[6] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: the Mars approach. *IEEE Micro*, 9(1):25–41, February 1989.

[7] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Prog. Lang. and Systems*, 4(3), July 1982.

[8] D. Powell, editor. *Delta-4 - A Generic Architecture for Dependable Distributed Computing*. ESPRIT Research Reports. Springer Verlag, November 1991.

[9] J. Rufino and P. Veríssimo. A study on the inaccessibility characteristics of ISO 8802/4 Token-Bus LANs. In *Proceedings of the IEEE INFOCOM'92 Conference on Computer Communications*, pages 958–966, Florence, Italy, May 1992. IEEE. (also INESC AR 16-92).

[10] J. Rufino and P. Veríssimo. A study on the inaccessibility characteristics of the Controller Area Network. In *Proceedings of the 2nd International CAN Conference*, pages 7.12–7.21, London, England, October 1995. CiA.

[11] José Rufino and Paulo Veríssimo. A study on the inaccessibility characteristics of ISO 8802/5 Token-Ring LANs. Technical Report RT/24-92, INESC, Lisboa, Portugal, February 1992.

[12] José Rufino and Paulo Veríssimo. A study on the inaccessibility characteristics of the FDDI LAN. Technical Report RT/25-92, INESC, Lisboa, Portugal, March 1992.

[13] 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, September 1994. CiA.

[14] P. Veríssimo. Redundant media mechanisms for dependable communication in Token-Bus LANs. In *Proceedings of the 13th Local Computer Network Conference*, Minneapolis-USA, October 1988. IEEE.

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

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