Hardware support for CAN fault-tolerant communication

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

Nuno Pedrosa
nffp@dione.ist.utl.pt
IST-UTL

José Monteiro
jctm@dione.ist.utl.pt
IST-UTL

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

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

Abstract

Fault-tolerant distributed applications based on field-buses may take advantage from the availability of highly-dependable communication systems. In this paper, we address this problem in the context of CAN, the Controller Area Network, to conclude that CAN native mechanisms alone are unable to fulfill all the attributes of fault-tolerant communication protocols. The paper discusses how existing CAN controllers can be complemented with some simple machinery and low-level protocol modules, handling the problem effectively. The result is an enhanced CAN infra-structure able to extremely reliable communication.

1 Introduction

The design and implementation of distributed computer control systems intended for real-world interfacing, i.e. integrating sensors and/or actuators, have increasingly been based on standard field-buses. The development of applications for such environments may greatly benefit from the availability of reliable communication services, such as those provided by group communication, membership and failure detection.

However, the migration of fault-tolerant communication systems to the realm of field-buses presents non-negligible problems, some of them addressed recently [10], in the context of CAN, a field-bus that has assumed increasing importance and widespread acceptance in application areas as diverse as shop-floor control, robotics or automotive.

A current belief considers that the CAN protocol guarantees a totally ordered message delivery either to all nodes or to none (atomic broadcast). However, this assumption does not hold, at least in systems with high reliability requirements [10]. In such cases, fault-tolerant communication services must be supported by a protocol layer built on top of CAN.

In this paper it is investigated how some simple machinery, in addition to a classic CAN controller chip, can be used to efficiently support some of those protocol functions, thus providing a CAN infra-structure with enhanced dependability characteristics.

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

[†]Faculdade de Ciências da Universidade de Lisboa.

2 Controller Area Network

The Controller Area Network is a multi-master bus using a twisted pair cable as transmission medium [5]. 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: while transmitting the frame identifier each node monitors the bus; if the transmitted bit is recessive and a dominant value is monitored, the node gives up transmitting and starts to receive incoming data; the node transmitting the frame with the lowest identifier goes through and gets the bus. Frames that have lost arbitration are automatically retransmitted.

A *frame* is a piece of encapsulated information that travels on the network. It may contain a *message*[1]: in CAN, a *data frame* is used for that purpose. However, it may consist of heading/trailing fields only, such as a *remote frame*, which has no data field.

The following discussion assumes the reader to be fairly familiar with CAN operation [5].

[1]A *message* is a user-level piece of information.

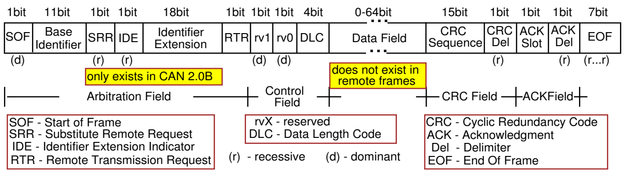


Figure 1: CAN frame structure

Impairments to dependability: the comprehensive set of CAN fault-confinement and error detection mechanisms ensures that most failures are perceived consistently by all nodes [2, 9]. Unfortunately, some subtle errors can lead to inconsistency and induce the failure of dependable communication protocols based on CAN operation alone. A thorough discussion of these problems can be found in [10].

One relevant aspect of that discussion concerns an important weakness of CAN error handling: in order to preserve consistency of frame dissemination, any recipient detecting an incorrect dominant value at the last bit of a frame[2] ($\times$ set in Figure 2-A) is obliged to accept the frame, because the sender may not have detected the error.

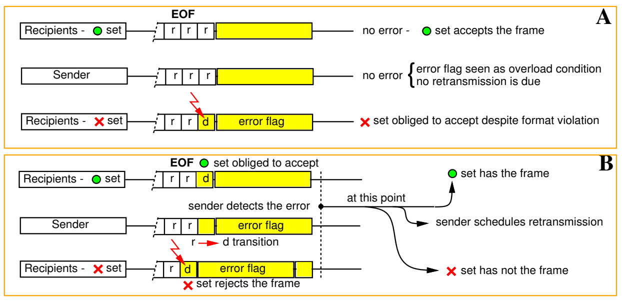


Figure 2: Inconsistency in CAN error handling

This opens room for inconsistent frame omissions, a problem that occurs if a disturbance corrupts the last but one bit of a frame in the recipients tagged $\times$ set in Figure 2-B: all recipients in this set[3] reject the frame while those in the $\bullet$ set must accept the frame; after error signaling, the sender schedules the frame for retransmission. Once the frame is retransmitted, an exact duplicate of the message will be accepted by the recipients in the $\bullet$ set of Figure 2-B. The problem gets worse if the sender fails before retransmission, which leads to an inconsistent message omission.

However infrequent it may be, the probability of occurrence of these scenarios is high enough to be taken into account, at least for highly fault-tolerant applications of CAN [10].

[2]Examples of causes for inconsistent detection are: electromagnetic interference or deficient receiver circuitry.

[3]The set may have only one element.

3 Fault-tolerant Communication

A software-based protocol suite intended to cope with inconsistent omission failures was presented in [10]. The occurrence of inconsistent omissions is formalized by the set of properties in Figure 3.

CAN1 - 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.

CAN2 - Weak Integrity: any message delivered to a correct node is delivered at least once.

CAN3 - Bounded Inconsistent Omission Degree: in a known time interval T_{rd}, inconsistent omission failures may occur in at most j transmissions.

Figure 3: Inconsistency-related CAN properties

The comparison of these properties with the relevant attributes of an atomic broadcast definition [3], clearly shows why the CAN protocol alone does not ensure an atomic broadcast service:

AB1 - Agreement: if a message is delivered to a correct node, then the message is eventually delivered to all correct nodes.

AB2 - Integrity: any message delivered to a correct node is delivered at most once.

Fault-tolerant broadcast protocols must then transform CAN1 and CAN2 into AB respective properties. In this paper, we address a hardware assisted solution, based on low-level dependability mechanisms, built on top of the CAN controller interface (Figure 4).

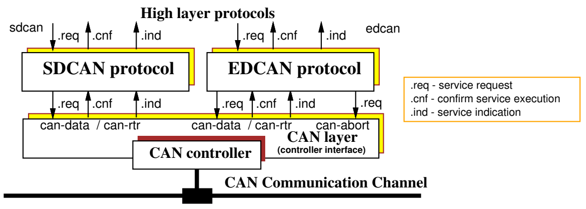


Figure 4: Low-level message diffusion protocols

The utilization of CAN 2.0B is assumed: the identifier fields are used to carry protocol control information, leaving the data field free to hold pure data. The grey shadowed area in Figure 5, signals the fields with greater relevance for low-level protocols.

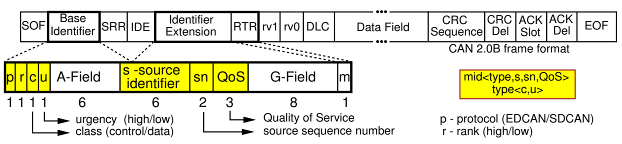


Figure 5: Protocol control information

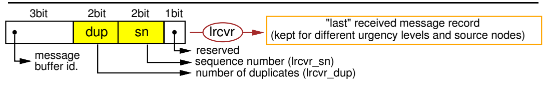
The SDCAN protocol (Figure 6) is intended to enhance CAN2 when no message ordering[4] is required. AB2 is guaranteed, by delivering the first copy of a message and discarding further duplicates. This procedure cannot be followed when messages need to be delivered accordingly with the order of the last retransmission (the successful one). In this case, all message copies are delivered to the higher layer, that will guarantee integrity, together with message ordering.

CAN Simple-Diffusion protocol

```
Sender
s00 when sdcan.req(mid⟨type⟨c,u⟩,s,sn,QoS⟩, mess) invoked at s do
s01     if mess = NULL then can-rtr.req(rank⟨HIGH⟩, mid);
s02     else can-data.req(rank⟨HIGH⟩, mid, mess); od;
s03 when can-rtr.cnf(mid) or can-data.cnf(mid, mess) confirmed do
s04     sdcan.cnf (mid,mess); od;
Recipient
r00 when can-data.ind(mid, mess) received at q
r01 or can-rtr.ind(mid, mess=NULL) received at q do
r02     lrcv_new(mid);                          // auxiliary function
r03     if lrcvr_dup[mid⟨type⟨u⟩,s⟩] = 1 then   // new message
r04         sdcan.ind (mid, mess);
r05     elif integrity(mid⟨QoS⟩) = WEAK then
r06         sdcan.ind (mid, mess); od;   // deliver all message copies
```

Figure 6: Specification of the SDCAN protocol

For each source node and urgency level, recipients locally maintain a record of relevant information, concerning the message with the "highest" sequence number, issued by that source (Figure 7). Each time a message is received, that record is updated, which allows to distinguish a new message from a duplicate.



3bit
2bit
2bit
1bit
dup
sn
lrcvr
"last" received message record (kept for different urgency levels and source nodes)
message buffer id.
reserved
sequence number (lrcvr_sn)
number of duplicates (lrcvr_dup)

```
lrcv_new(mid⟨type⟨c,u⟩,s,sn,QoS⟩) (auxiliary function)
a00 old_mid := lrcv_mid_get(mid⟨type⟨u⟩,s⟩);
a01 if mid⟨sn⟩ > lrcvr_sn[mid⟨type⟨u⟩,s⟩] then   // new message
a02     lrcvr_sn[mid⟨type⟨u⟩,s⟩] := mid⟨sn⟩;
a03     lrcvr_dup[mid⟨type⟨u⟩,s⟩] := 1;
a04     lrcv_mid_new(mid);
a05 else                                          // message duplicate
a06     lrcvr_dup[mid⟨type⟨u⟩,s⟩] := lrcvr_dup[mid⟨type⟨u⟩,s⟩] + 1;
a07 return old_mid;
```

Figure 7: Keeping message transaction information

The EDCAN protocol (Figure 8) is intended to enhance both integrity (CAN2) and agreement (CAN1) properties. AB1 is guaranteed by making recipients responsible for message retransmission. Retransmissions may stop once the number of duplicates exceeds the inconsistency degree bound (CAN3). Ordering of message delivery is not ensured.

CAN Eager-Diffusion protocol

```
Sender
s00 when edcan.req(mid⟨type⟨c,u⟩,s,sn,QoS⟩, mess) invoked at s do
s01     if mid⟨sn⟩ ≥ lrcvr_sn[mid⟨type⟨u⟩,s⟩] then
s02         if mess = NULL then can-rtr.req(rank⟨LOW⟩, mid);
s03         else can-data.req(rank⟨LOW⟩, mid, mess); od;
s04 when can-rtr.cnf(mid) or can-data.cnf(mid, mess) confirmed do
s05     edcan.cnf (mid,mess); od;
Recipient
r00 when can-data.ind(mid, mess) received at q
r01 or can-rtr.ind(mid, mess=NULL) received at q do
r02     old_mid := lrcv_new(mid);               // auxiliary function
r03     if lrcvr_dup[mid⟨type⟨u⟩,s⟩] = 1 then   // new message
r04         can-abort.req(old_mid);
r05         edcan.ind (mid, mess);
r06         if mess = NULL then can-rtr.req(rank⟨HIGH⟩, mid);
r07         else can-data.req(rank⟨HIGH⟩, mid, mess);
r08     elif lrcvr_dup[mid⟨type⟨u⟩,s⟩] > j then can-abort.req(mid); od;
```

Figure 8: Specification of the EDCAN protocol

In addition, it has to be guaranteed that no overrun incidents will ever occur in the management of CAN controller receive buffers, because this kind of omission failures will jeopardize the whole CAN protocol reliability guarantees.

4 CAN Dependability Engine

The short number of buffers available for the storage of incoming messages, in current CAN controllers, imposes the need of an appropriate hardware support in order to avoid overrun incidents. To cope with this problem we have devised a specialized hardware infrastructure, in complement to the CAN controller chip.

Since that circuitry is based on a micro-controller, other CAN dependability enforcement mechanisms, like those described in section 3, can be added with a little extra cost, provided that the overall system timeliness and storage requirements complies with existing speed and memory restrictions.

Dubbed CANDLE (CAN DependabiLity Engine), this component exhibits a modular architecture, in the sense that it makes no assumptions about upper and lower interfacing technologies. The CANDLE architecture, represented in Figure 9, is described next.

System interface - this unit comprises two different channels, implemented through FIFO[5] dual-port memories. To preserve latency, service *requests* are queued one at a time, on the *input channel*. Conversely, the *confirm* of service requests and service *indications* are queued, on the *output channel*, as soon as they become available. The output channel FIFO should be made as deep as required to support traffic bursts under the worst-case system latency.

[4]Implicitly defined by the QoS (Quality of Service).

[5]First-In First-Out.

CAN controller - in order to cope with different service *request* urgency levels and to efficiently support frame retransmissions by the EDCAN protocol, it is recommended to have a CAN controller with *object storage*[6]. The Intel 82527 controller was selected [4]. From the 15 message buffers available:

- one is exclusively dedicated to receive incoming traffic;
- two buffers, in a total of four, are assigned to SDCAN for transmission of control/data messages, at each urgency level;
- the EDCAN protocol may use the remaining buffers.

In order to prevent priority inversion [7], upon message retransmission, the EDCAN protocol should take into account that the 82527 internally uses buffer identifiers rather than message identifiers to schedule multiple on-chip requests.

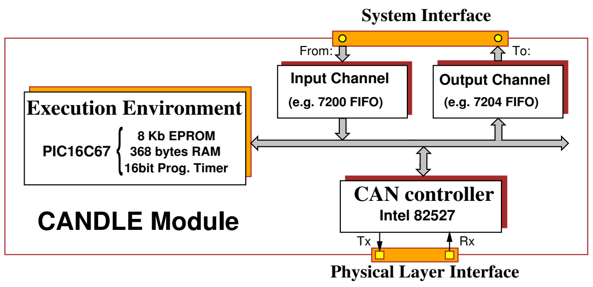


Figure 9: CAN Dependability Engine

Execution environment - entirely responsible for the command of the CAN controller, including message buffer management. Furthermore, it must support the execution of message diffusion protocols.

The PIC16C67 [6], a low-cost, high-performance, 8-bit micro-controller has been used in the design of the execution unit. Based on a RISC-architecture, the PIC16C67 can achieve a 200ns instruction cycle.

This is a fundamental factor to guarantee execution of all competing threads (receive, transmit and layer management) within the period corresponding to the minimum frame inter-arrival time. Additionally, this unit can be made responsible for multicast traffic filtering and message timestamping.

Physical layer interface - the physical layer circuitry is intentionally not included in the CANDLE architecture, in order to allow multiple design solutions. For example: classical wiring, low-cost fiber optics or the dual-media solution described in [8] for resilience against network partitions.

[6]Formerly called FullCAN.

5 Related Work

Commercially available circuits implement in silicon the standard CAN protocol. Different proposals to improve CAN operation [1, 11] require the modification of those chips. Our approach allows to enhance CAN dependability, using existing CAN controllers.

6 Conclusions

There is a need for fault-tolerant communications in distributed control systems based on field-buses. We have analyzed this problem, in the context of CAN. Dismissing the misconception that CAN native mechanisms support an atomic broadcast service we have looked for systemic solutions. The CANDLE architecture uses some simple machinery and low-level protocol entities, to complement the bare CAN operation, thus yielding an interface with enhanced dependability characteristics.

Acknowledgments - the authors wish to thank L. Rodrigues and C. Almeida for their valuable suggestions.

References

[1] G. Cena and A. Valenzano. An improved CAN fieldbus for industrial applications. *IEEE Transactions on Industrial Electronics*, 44(4):553–564, August 1997.

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

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

[4] Intel. *82527 - Serial Communications CAN Protocol Controller*, December 1995.

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

[6] Microchip Technology Inc., USA. *PIC 16C6X: 8-bit CMOS Microcontrollers*, 1997.

[7] J. Peden and A. Weaver. Performance of priorities on an 802.5 token ring. In *Proceedings SIGCOM'87 Symposium*, Stowe, VT, August 1987. ACM.

[8] J. Rufino. Dual-media redundancy mechanisms for CAN. Technical Report CTI RT-97-01, Instituto Superior Técnico, Lisboa, Portugal, January 1997.

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

[10] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in CAN. Technical Report CTI RT-97-04, Instituto Superior Técnico, Lisboa, Portugal, December 1997. (submitted for publication).

[11] K. Tindell and H. Hansson. Babbling idiots, the dual-priority protocol and smart CAN controllers. In *Proceedings of the 2nd International CAN Conference*, London, England, October 1995. CiA.