Controller Area Network

Joaquim Ferreira, José Fonseca

1 Introduction

Controller Area Network (CAN) is a popular and very well-known bus system, both in academia and in industry. CAN protocol was introduced in the mid eighties by Robert Bosch GmbH [7] and it was internationally standardized in 1993 as ISO 11898-1 [27]. It was initially designed to distributed automotive control systems, as a single digital bus to replace traditional point-to-point cables that were growing in complexity, weight and cost with the introduction of new electrical and electronic systems. Nowadays CAN is still used extensively in automotive applications, with an excess of 400 million CAN enabled microcontrollers manufactured each year [14].

The widespread and successful use of CAN in the automotive industry, the low cost associated with high volume production of controllers and CAN's inherent technical merit, have driven to CAN adoption in other application domains such as: industrial communications, medical equipment, machine tool, robotics and in distributed embedded systems in general.

CAN provides two layers of the stack of the Open Systems Interconnection (OSI) reference model: the physical layer the data link layer and optionally an additional application layer (not standardized). Notice that CAN physical layer was not defined in Bosch original specification, only the data link layer was defined. However, the CAN ISO specification filled this gap and the physical layer was then fully specified. CAN is a message-oriented transmission protocol, i.e., it defines message contents rather than nodes and node addresses. Every message has an associated message identifier, which is unique within the whole network, defining both the content and the priority of the message. Transmission rates are defined up to 1 Mbps.

The large installed base of CAN nodes with low failure rates over almost two decades, led to the use of CAN in some critical applications such as Anti-locking Brake Systems (ABS) and Electronic Stability Program (ESP) in cars. In parallel with the wide dissemination of CAN in industry, the academia also devoted a large effort to CAN analysis and research, making CAN one of the must studied fieldbuses. That is why a large number of books or book chapters describing CAN were published. The first CAN book, written in French by D. Paret, was published in 1997 and presents the CAN basics [36]. More implementation oriented approaches, including CAN node implementation and application examples, can be found in Lorenz

[32] and in Etschberger [16], while more compact descriptions of CAN can be found in [11] and in some chapters of [35].

Despite its success story, CAN application designers would be happier if CAN could be made faster, cover longer distances, be more deterministic and more dependable [41][39]. Over the years, several protocols based in CAN were presented, taking advantage of some CAN properties and trying to improve some known CAN drawbacks. This chapter, besides presenting an overview of CAN, describes also some other relevant higher level protocols based on CAN, such as CANOpen [13], DeviceNet [6], FTT-CAN [1] and TTCAN [28].

2 CAN Technology Basics

The original specification [7] contains few more than a MAC protocol based on decentralized medium random access (CSMA type) that uses a particular physical characteristic of the medium to support a non-destructive bit-wise arbitration. The communication is message-oriented since each message receives an identifier which must be unique in the system and which establishes the message priority for the arbitration process. Any transmitted CAN message is broadcast to every node in the network.

Whenever two or more nodes start transmitting at the same time, only the one transmitting the message with the highest priority will proceed with the transmission. All others quit and try again after the current transmission ceases. This deterministic mechanism allows to compute the worst-case response time of all CAN messages [50][14]. So, from the real-time point of view, CAN is a serial data bus that supports priority based message arbitration and non-preemptive message transmission. In 1994 Tindell and Burns [50] and Tindell et al. [51] adapted previous research on fixed priority preemptive scheduling for single processor systems to the scheduling of messages on CAN, providing a method of calculating the worst-case response times of all CAN messages. Prior to Tindel's results, the bus utilization in automotive applications was typically around 30 or 40%, but still requiring extensive testing to obtain confidence that CAN messages would meet their deadlines [14], since then there were no analysis tools that could guarantee real-time behavior. With the advent of new design techniques, based on schedulability analysis, CAN bus utilization could be increased to around 80% [15]. Tindel's results were transferred to industry in the form of commercial CAN schedulability analysis tools, e.g., Volcano Network Architect, that have been used by a large number of major automotive manufacturers. Unfortunately, the initial CAN schedulability analysis [50] [51] was flawed [14]. It may provide guarantees for messages that, in the worst-case scenario, will in fact miss their deadlines. Davis et al [14] correct previous flawed analysis and discuss the impact on commercial CAN systems designed and developed using flawed schedulability analysis, while making recommendations for the revision of CAN schedulability analysis tools.

2.1 Physical Layer

The CAN ISO standard incorporates the original Bosch specifications [7] as well as part of the physical layer, the physical signaling, the bit timing and synchronization. There are a small number of other CAN physical layer specifications including:

- **ISO 11898-2 High Speed** – ISO 11898-2 is the most used physical layer standard for CAN networks. In this standard the data rate is defined up to 1 Mbit/s with a theoretically possible bus length of 40 m at 1 Mbit/s.
- **ISO 11898-3 Fault-Tolerant** – This standard defines data rates up to 125 kbit/s with the maximum bus length depending on the data rate used and the bus load. Up to 32 nodes per network are specified. The fault-tolerant transceivers support the complete error management including the detection of bus errors and automatic switching to asymmetrical signal transmission.
- **SAE J2411 Single Wire** – An unshielded single wire is defined as the bus medium and the communication takes place with a nominal data rate of 33,3 kbit/s The standard defines up to 32 nodes per network. The main application area of this standard is in comfort electronics networks in vehicles.
- **ISO 11992 Point-to-Point** –This standard defines a point-to-point connection for use mainly in vehicles with trailers. The nominal data rate is 125 kbit/s with a maximum bus line length of 40 m.

The most popular CAN physical layer protocol, available in most of the CAN transceivers, is the one defined in the ISO 11898-2 standard [27]. The maximum achievable bus line length in a CAN network, represented in Table 1, depends on:

- The loop delays of the connected bus nodes and the delay of the bus lines.
- The differences of the relative oscillator tolerance between nodes.
- The signal amplitude drop due to the series resistance of the bus cable and the input resistance of bus nodes.

Notice, however that it is allowed to use bridge-devices or repeaters to increase the maximum bus line length (not considered in Table 1).

The CAN physical layer provides a 2-state medium: the *dominant* and the *recessive*. Whenever two nodes simultaneously transmit bits of opposite value, then all nodes should read dominant. Usually the dominant state is associated with the binary value 0 and recessive with the binary value 1.

The physical layer has a number of built-in fault-tolerant features. CAN provides resilience against a variety of physical faults such as one open wire, the short-circuit of the two signal wires, or even one of the

Bit Rate	Bus Length
1 Mbit/s	30 m
800 kbit/s	50 m
500 kbit/s	100 m
250 kbit/s	250 m
125 kbit/s	500 m
62,5 kbit/s	1000 m
20 kbit/s	2500 m
10 kbit/s	5000 m

Table 1: Practical CAN bus length for ISO 11898 compliant transceivers and standard bus line cables (adapted from [12]).

signal wires shorted to ground or power. Notice that not all CAN controllers are able to implement these features, by switching from differential signaling to single line signalling, at higher bus speeds. The CAN differential electrical signaling mode is very resistant to electromagnetic interference (EMI) since interference will tend to affect each side of a differential signal almost equally. However, the differential electrical signalling does not fully prevent electromagnetic interference to affect the signal on the bus in such a way that one or more nodes on the bus will simultaneously read a different bit value from that which was transmitted. A node detecting the error (possibly the transmitter) will invalidate the message by transmitting an error frame. The number and the nature of EMI induced transmission faults and the ability of the physical layer to prevent them depends on several factors as the cables type and length, the number of nodes, the transceiver type, the EMI shielding, etc.

2.2 Data Link Layer

The data link layer of CAN includes the services and protocols required to assure a correct transfer of information from one node to another.

There are four types of message on CAN: data frames, error frames, remote transmission request frames and overload frames. The latter two types of messages are rarely used in real application and, thus, will not be described further.

The CAN protocol supports two message frame formats, the only essential difference being in the length of the identifier. The CAN *base frame* format supports a length of 11 bits for the identifier (formerly known as CAN 2.0 A), and the CAN *extended frame* format supports a length of 29 bits for the identifier (formerly known as CAN 2.0 B).

Data on the bus is sent in data frames which consist of up to 8 bytes of data plus a header and a footer.

The frame is structured as a number of fields, as depicted in Figure 1.

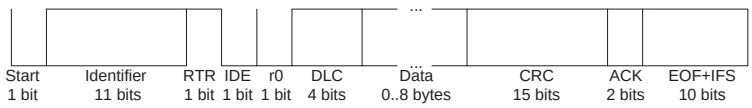

Figure 1: CAN base frame format.

A CAN base frame message begins with the start bit called *Start Of Frame* (SOF). This bit is followed by the *arbitration field* which consist of the identifier and the *Remote Transmission Request* (RTR) bit used to distinguish between the data frame and the data request frame called remote frame. The following *control field* contains the *IDentifier Extension* (IDE) bit to distinguish between the CAN base frame and the CAN extended frame, as well as the Data Length Code (DLC) used to indicate the number of following data bytes in the *data field*. If the message is used as a remote frame, the DLC contains the number of requested data bytes. The data field that follows is able to hold up to 8 bytes. The integrity of the frame is guaranteed by the following *Cyclic Redundant Check* (CRC) sum. The *ACKnowledge* (ACK) field comprises the ACK slot and the ACK delimiter. The bit in the ACK slot is sent as a recessive bit and is overwritten as a dominant bit by those receivers which have at this time received the data correctly. Correct messages are acknowledged by the receivers regardless of the result of the acceptance test. The end of the message is indicated by *End Of Frame* (EOF). The *Intermission Frame Space* (IFS) is the minimum time in equivalent number of bits separating consecutive messages. Unless another station starts transmitting, the bus remains idle after this.

Associated with every CAN message there is a unique message identifier that defines its content and also the priority of the message. Bus access conflicts are resolved by a non-destructive bitwise arbitration scheme where the identifiers of the involved messages are observed bit-by-bit by all nodes, in accordance with the wired-AND mechanism, by which the dominant state overwrites the recessive state. All those nodes with recessive transmission and dominant observation lose the competition for bus access. The nodes that lost the arbitration automatically become receivers of the message with the highest priority and do not re-attempt transmission until the bus is available again. In this way, transmission requests are handled in order of their importance for the system as a whole.

2.3 Detecting and signaling errors

A CAN node does not acknowledge message reception, instead it signals errors immediately as they occur. For error detection the CAN protocol implements three mechanisms at the message level:

- **Cyclic Redundancy Check** – This mechanism accounts for message corruption. The transmitter node computes the CRC and transmits it within the message. The receiver decodes the message and

recomputes the CRC and if they do not match, there has been a CRC error.

- **Frame check** – This mechanism detects message format violations, i.e., it checks each field against the fixed format and the frame size.
- **Acknowledge errors** – Since the receivers of a message must issue an acknowledgement bit in the ACK field, if the transmitter does not receive an acknowledgement an ACK error is indicated, thus allowing a node to detect isolation from the network.

Besides error detection at the message level, the CAN protocol also implements mechanisms for error detection at the bit level:

- **Transmission monitoring** – Each node transmitting a message also monitors the bus level to be able to detect differences between the bit sent and the bit received. This mechanism allows distinguishing global errors from errors local to the transmitter only.
- **Bit stuffing** – CAN uses a non return to zero codification to prevent nodes from loosing synchronization by receiving long sequences of recessive or dominant bits. A supplementary bit is inserted by the transmitter into the bitstream after five consecutive equal bits. The stuff bits are removed by the receivers that also detect violations of the bit stuffing rule.

If at least one station detects any error, it will start transmitting an error frame in the next bit aborting the current message transmission. This prevents other stations from accepting the message and thus ensures the consistency of data throughout the network. After transmission of an erroneous message that has been aborted, the sender automatically re-attempts transmission (automatic retransmission). During the new arbitration process all nodes compete for bus access and the one with the higher priority message will win arbitration. Thus the message affected by the error could be delayed. There is, however, a special case where consistency throughout the network is compromised, as it will be discussed later.

To prevent a faulty CAN controller to abort all transmissions, including the correct ones, the CAN protocol provides a mechanism to distinguish sporadic errors from permanent errors and local failures at the station. This is done by statistical assessment of station error situations with the aim of recognizing a station's own defects and possibly entering an operation mode in which the rest of the CAN network is not negatively affected. This may go as far as the station switching itself off (bus-off state).

In CAN each node that detects an error sends an error flag composed of six consecutive dominant bits, i.e., a violation of the bit stuffing rule, enabling all nodes on the bus to be aware of a transmission error. The frame affected by the error automatically re-enters into the next arbitration phase. The error recovery time (the time from detecting an error until the possible start of a new frame) varies from 17 to 31 bit times [34].

To prevent an erroneous node from disrupting the functioning of the whole system, e.g. by repetitively sending error frames, the CAN protocol includes fault confinement mechanisms that are able to detect permanent hardware malfunctioning and to remove defective nodes from the network. To do this a CAN controller has two error counters; the transmit error (TEC) and the receive error (REC) counters which are incremented/decremented according to a set of rules [7][27]. Each time a frame is correctly received or transmitted by a node, the value of the corresponding counter is decreased. Conversely each time a transmission error is detected the value of the corresponding counter is increased.

Depending on the value of both counters, the station will be in one of the three states defined by the protocol: error active, error passive and bus-off. In the error active state (REC<128 and TEC<128) the node can send and receive frames without restrictions. In the error passive state ((REC>127 or TEC>127) and TEC<=255) the node can transmit but it must wait 8 supplementary bits after the end of the last transmitted frame and it is not allowed to send active error frames upon the detection of a transmission error, it will send passive error frames instead. Furthermore, an error-passive node can only signal errors while transmitting.

After behaving well again for a certain time, a node is allowed to re-assume the error-active status. When the TEC is greater than 255 the node CAN controller goes to the bus-off state. In this state, the node can neither send nor receive frames and can only leave this state after a hardware or software reset and after having successfully monitored 128 occurrences of 11 consecutive recessive bits (a sequence of 11 consecutive recessive bits corresponding to the ACK, EOF and the intermission field of a correct data frame).

When in the error passive state, the node signals the errors in a way that cannot force the transmitter to retransmit the incorrectly received frame. This behavior is a possible source of inconsistency that must be controlled. As an example of the consequences this can have, consider the case of an error-passive node being the only one to detect an error in a received frame. The transmitter will not be forced to retransmit and the error-passive node will be the only one not to receive the message. Several authors proposed avoiding the error passive [47][26][23] state to eliminate this problem. This is easily achieved [47][41] using a signal available in most CAN circuits, the error warning notification signal. This signal is generated when any error counter reaches the value 96. This is a good point to switch off the node before it goes into the error-passive state, assuring that every node is either helping to achieve data consistency or disconnected.

2.4 Network topology

CAN network topology is bus based. Replicated busses are not referred in CAN standard, however it is possible to implement them [45] [48].

Over the years, some star topologies for CAN have been proposed [44][29][4]. The solution presented in [29], is based in a passive star network topology and relies on the use of a central element, the star coupler,

to concentrate all the incoming signals. The result of this coupling is then broadcast to the nodes. The solution presented by Rucks [44] is based on an active star coupler capable of receiving the incoming signals from the nodes bit by bit, implementing a logical AND, and retransmitting the result to all nodes. None of these solutions is capable of disconnecting a defecting branch and so, from the dependability point of view they only tolerate spatial proximity faults.

Barranco *et al.* [4] proposed a promising solution based in an active hub that is able to isolate defective nodes from the network. This active star is compliant with CAN standard and allows detecting a variety of faults (stuck-at node fault, shorted medium fault, medium partition fault and bit-flipping fault) that will cause the faulty node to be isolated. The replication of the active star has also been considered [5]. Recently, Valter *et al.* [48] proposed a set of components, an architecture and protocols to enable the dynamic management of the topology of CAN networks made of several replicated buses, both to increase the total bandwidth and to reconfigure the network upon bus permanent error. In many operational scenarios, the proposed solution could be plugged into existing systems to improve its resilience to bus permanent error, without changing the code running in the nodes. In [49] these proposals were adapted to FTT-CAN with multiple masters and multiple buses.

3 CAN Based Upper Layer Protocols

Several CAN based higher layer protocols have emerged over the years. Some of them are widely used in industry, e.g., CANopen[13] and DeviceNet[3], while others such as TCAN[8], FlexCAN[40] and FTT-CAN[2] have emerged in academia to overcome some well known CAN limitations namely large and variable jitter, lack of clock synchronization, limited speed-distance product, flexibility limitations, data consistency issues, limited error containment and limited support for fault tolerance [39]. In 2001 an extension of CAN, the Time-Triggered CAN (TTCAN) [28] was presented to support explicit time-triggered operation on top of CAN. Although TTCAN is not strictly an upper layer protocol based on CAN, it is considered in this section since it offers some services that can be found in some CAN based upper layer protocols.

Due to space limitations, this section presents only an overview of a subset of these CAN based protocols, namely: CANOpen, DeviceNet, FTT-CAN and TTCAN.

3.1 CANOpen

CANopen, internationally standardized as CENLEC EN 50325-4, is a CAN-based higher-layer protocol and its specifications covers application layer and communication profile, a framework for programmable devices, recommendations for cables and connectors and SI units and prefix representations. In terms of configuration capabilities, CANopen is quite flexible and CANopen networks are used in a very broad range of application

fields such as machine control, building automation, medical devices, maritime electronics, etc [13]. CANopen was initially proposed in an EU-Esprit research project and, in 1995, its specification was handed over to the CAN in Automation (CiA) an international group of users and manufacturers.

As depicted in Figure 2, the CANopen device model can be divided into three parts: the communication interface and protocol software, the object dictionary and the process interface and application program. The communication interface and protocol software provide services to transmit and to receive communication objects over the CAN bus. The object dictionary describes all data types, communication objects and application objects used in the device, acting as the interface to the application software. The application program provides the internal control functionality as well as the interface to the process hardware interfaces.

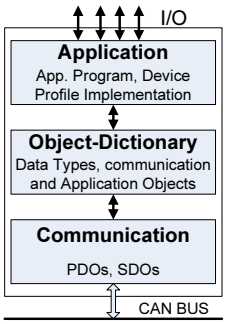

Figure 2: CANopen device model.

A device profile is the complete description of the device application with respect to the data items in the object dictionary. The object directory is the central element of every CANopen device describing the device's functionality. The object directory contains all the parameters of a device that can be accessed via the network, for example, device identifiers, manufacturer name, communications parameters and device monitoring. The device-specific area contains the connection to the process, i.e., the I/O functionality and drivers' parameters. The behavior in the event of an error can also be configured in the object directory. Accordingly, the behavior of a device can be adapted to the respective utilization requirements using the object directory [13].

CANopen uses standardized profiles and off-the-shelf devices, tools, and protocol stacks are commercially available. In an effort to promote reuse of application software and to potentiate communication compatibility, interoperability and interchangeability of devices, CANopen provides pre-defined application objects. Also, manufacturer-specific functionality in devices, can be added to the generic functionality described in the profiles. CANopen provides standardized communication objects for real-time data (Process Data Objects, PDO), configuration data (Service Data Objects, SDO), and special functions (Time Stamp, Sync

message, and Emergency message) as well as network management data (Boot-up message, NMT message, and Error Control).

CANopen differentiates between two data transfer mechanisms: fast exchange of short process data, using process data objects (PDOs) and access to the entries of the object directory, that is done via SDO (Service Data Object). PDOs can be asynchronous, synchronous or on-demand and the transfer is done without protocol overhead. The synchronous transmission of messages is supported by pre-defined communication objects (Sync message and time stamp message). Synchronous messages are transmitted with respect to a pre-defined synchronization message, asynchronous message may be transmitted at any time. Synchronous transmission of a PDO message means that the time elapsed form the transmission of the Sync message to the transmission of the PDO message is constant, i.e., the synchronous message is transmitted within a fixed time window with respect to the Sync transmission, and at most once for every period of the Sync. SDOs are confirmed data transfers that establish point-to-point communication between two devices, implementing services of handshaking, fragmentation and reassembly. SDOs are used for parameter passing, configuration, etc.

CANopen supports three types of communication models: master/slave, client/cerver and producer/consumer. CANopen networks also provide redundant transfer of safety-oriented information in defined time windows.

3.2 DeviceNet

DeviceNet, international standard IEC 62026-3, is other CAN-based higher layer protocol, initially proposed by Allen-Bradley (now owned by Rockwell Automation), and is mostly dedicated to industrial automation. DeviceNet specifications include the application layer and device profiles. The management of the DeviceNet specification was transferred to the Open DeviceNet Vendor Association (ODVA), a non-profit organization that develops and markets DeviceNet [3].

The DeviceNet physical layer specifies a terminated trunk and drop line configuration. Communication and power are provided in the trunk and drop cables, enabling devices to be directly powered from the bus.

Up to 64 logical nodes can be connected to a single DeviceNet network, using both sealed and open-style connections. There is support for strobed, polled, cyclic, change-of-state, and application-triggered data transfer. The user can choose master/slave, multi-master and peer-to-peer or a combination configuration, depending on device capability and the application requirements. DeviceNet is based on the Part A of the CAN standard, therefore it uses the eleven bit identifier. DeviceNet uses abstract object models to describe communication services, data and behavior. A DeviceNet node is built from a collection of objects describing the system behavior and grouping data using virtual objects, as any object oriented programming language would [6].

DeviceNet communication model defines an addressing scheme that provides access to objects within a

device. The object model addressing information includes:

- **Device Address** – Referred to as the Media Access Control Identifier (MAC ID), is an integer identification value assigned to each node on the network. A test, executed at power-up, guarantees the uniqueness of the value on the network.
- **Class Identifier** – Refers to a set of objects that represent the same type of system component. The class identifier is an integer identification value assigned to each object accessible from the network.
- **Instance Identifier** – Refers to the actual representation of an object of a class. The instance identifier is an integer identification assigned to an object instance that identifies it among all instances of the same class within a particular device, with an unique identifier value.
- **Attribute Identifier** – Attributes are parameters associated with an object. Attributes typically provide some type of status information, represent a characteristic with a configurable value, or control the behavior of an object. The attribute identifier is an integer identification value assigned to an object attribute.

The DeviceNet application layer specifies how CAN identifiers are assigned and how the data field is used to specify services, move data, and determine its meaning. DeviceNet adopts the producer-consumer model, so the source device broadcast the data to the network with the proper identifier. All devices who need data listen for messages and when devices recognize the appropriate identifier, they consume the data.

There are two types of messages usually required by most automation devices: I/0 messages and explicit messages. I/O messages are for time critical control-oriented data and provide dedicated communication paths between a producing application and one or more consuming applications and typically use the higher priority identifiers [6]. Explicit messages, on the other hand, provide multi-purpose point-to-point communication paths between two devices. They provide the typical request/response-oriented network communications used to perform node configuration and diagnosis and typically use lower priority identifiers. Fragmentation services are also provided for messages that are longer than eight bytes.

3.3 TTCAN

Time-Triggered Communication on CAN (TTCAN) is an extension of CAN, introducing time-triggered operation based on a high precision network-wide time base. There are two possible levels in TTCAN, level-1 and level-2. Level-1 only provides time triggered operation using local time (Cycle_Time). Level-2 requires a hardware implementation and provides increased synchronization quality, global time and external clock synchronization. As native CAN, TTCAN is limited to a maximum data rate of 1 Mbit/s, with typical data efficiency below 50%. TTCAN network topology is bus based. Replicated busses are not referred in TTCAN standard, however it is possible to implement them [33].

TTCAN adopts a Time-division Multiple Access (TDMA) bus access scheme. The TDMA bandwidth allocation scheme divides the timeline into time slots, or time windows. Network nodes are assigned different slots to access the bus. The sequence of time slots allocated to nodes repeats according to a basic cycle. Several basic cycles are grouped together in a matrix cycle. All basic cycles have the same length, but can differ in their structure. When a matrix cycle finishes the transmission scheme starts over by repeating the matrix cycle (Figure 3). The matrix cycle defines a message transmission schedule. However, a TTCAN node does not need to know the whole system matrix, it only needs information of the messages it will send and receive.

Since TTCAN is built on top of native CAN, some time windows may be reserved for several event messages. In such windows, it is possible that more than one transmitter may compete for the bus access right. During these slots the arbitration mechanism of the CAN protocol is used to prioritize the competing messages and to grant access to the higher priority one. In this sense, the medium access mechanism in TTCAN can be described as TDMA with Carrier Sense Multiple Access with Bitwise Arbitration (CSMA-BA) in some pre-defined time slots. This feature makes the TTCAN protocol as flexile as CAN during the arbitration windows, without compromising the overall system timeliness, i.e. the event-triggered messages do not interfere with the time-triggered ones.

The TDMA cycle starts with the transmission of a reference message from a time master. The reference messages are regular CAN messages with a special and known *a priori* identifier and are used to synchronize and calibrate the time bases of all nodes according to the time master's time base, providing a global time for the network.

TTCAN level-1 guarantees the time-triggered operation of CAN based on the reference message of a time master. Fault-tolerance of that functionality is established by redundant time masters, the so called potential time masters. Level-2 establishes a globally synchronized time base and a continuous drift correction among the CAN controllers.

There are three types of time windows (Figure 3):

- **Exclusive time windows** – for periodic messages that are transmitted without competition for the CAN bus. No other message can be scheduled in the same window. The automatic retransmission, upon error, is not allowed.

- **Arbitrating time windows** – for event triggered messages, where several event-triggered messages may share the same time window and bus conflicts are resolved by the native CAN arbitration. Two or more consecutive arbitrating time windows can be merged. Message transmission can only be started if there is sufficient time remaining for the message to fit in. Automatic retransmission of CAN messages is disabled (except for merged arbitrating windows).

- **Free time windows** – reserved for future extensions of the network. A transmission schedule could reserve time windows for future use, either for new nodes or to assign existing nodes more bandwidth. Notice that a free time window can not be assigned to a message unless it is previously converted to either an exclusive or an arbitrating time window.

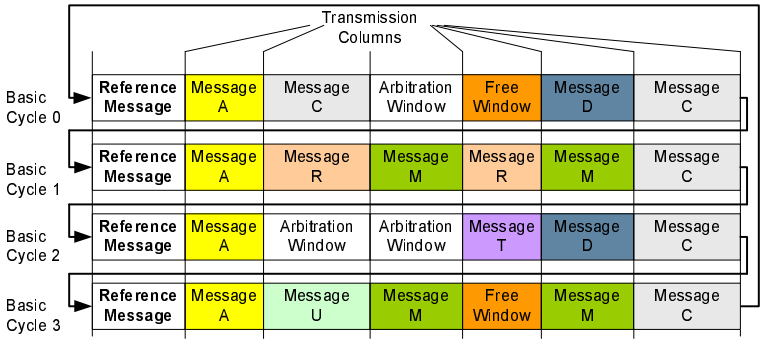

Figure 3: TTCAN system matrix, where several basic cycles build the matrix cycle (adapted from [24]).

Clock synchronization in a TTCAN network is provided by a time master. The time master establishes its own local time as global time by transmitting the reference message. To compensate for slightly different clock drifts in the TTCAN nodes and, to provide a consistent view of the global time, the nodes perform a drift compensation operation. A unique time master would be a single point of failure, thus TTCAN provides a mechanism for time masters redundancy and replacement whenever the current time master fails to send a reference message. In this case, the CAN bus remains idle and any of the potential time masters will try to transmit a reference message after a certain amount of time. In case two potential time masters try to send a reference message at the same time, the native CAN bit arbitration mechanism ensures that only the one with the highest priority wins. When a failed time master reconnects to the system with active time triggered communication, it waits until it is synchronized to the network before it may try to become time master again.

The strategy adopted, concerning fault confinement, is very similar to the one followed by native CAN, i.e., error passive and bus-off. Since CAN failures are already considered in ISO 11898-1 standard (data link layer), TTCAN considers mainly scheduling errors (e.g. absence of a message) and each TTCAN controller only provides error detection. Active fault confinement is left to a higher layer or to the application.

TTCAN may work under several operational modes: configuration mode, CAN communication, time-triggered communication or event synchronized time-triggered communication. However, operating modes may only change via configuration mode. The system matrix configuration may be read and written by the application during initialization, but it is locked during time-triggered communication, i.e., scheduling tables are locally implemented in every node and must be configured before system start-up.

3.4 FTT-CAN

The basis for the FTT-CAN protocol (Flexible Time-Triggered communication on CAN) has been first presented in [1]. Basically, the protocol makes use of the dual-phase elementary cycle concept in order to combine time and event-triggered communication with temporal isolation. Moreover, the time-triggered traffic is scheduled on-line and centrally in a particular node called master. This feature facilitates the on-line admission control of dynamic requests for periodic communication because the respective requirements are held centrally in just one local database. With on-line admission control, the protocol supports the time-triggered traffic in a flexible way, under guaranteed timeliness. Furthermore, there is yet another feature that clearly distinguishes this protocol from other proposals concerning time-triggered communication on CAN [38][28] that is the exploitation of its native distributed arbitration mechanism. In fact, the protocol relies on a relaxed master-slave medium access control in which the same master message triggers the transmission of messages in several slaves simultaneously (master/multi-slave). The eventual collisions between slave's messages are handled by the native distributed arbitration of CAN.

The protocol also takes advantage of the native arbitration to handle event-triggered traffic in the same way as the original CAN protocol does. Particularly, there is no need for the master to poll the slaves for pending event-triggered requests. Slaves with pending requests may try to transmit immediately, as in normal CAN, but just within the respective phase of each elementary cycle. This scheme, similar to the arbitration windows in TTCAN, allows a very efficient combination of time and event-triggered traffic, particularly resulting in low communication overhead and shorter response times.

In FTT-CAN the bus time is slotted in consecutive Elementary Cycles (ECs) with fixed duration. All nodes are synchronized at the start of each EC by the reception of a particular message known as EC trigger message, which is sent by the master node.

Within each EC the protocol defines two consecutive windows, asynchronous and synchronous, that correspond to two separate phases (see figure 4). The former one is used to convey event-triggered traffic, herein called asynchronous because the respective transmission requests can be issued at any instant. The latter one is used to convey time-triggered traffic, herein called synchronous because its transmission occurs synchronously with the ECs. The synchronous window of the n^{th} EC has a duration that is set according to the traffic that is scheduled for it. The schedule for each EC is conveyed by the respective EC trigger message (see figure 5). Since this window is placed at the end of the EC, its starting instant is variable and it is also encoded in the respective EC trigger message. The protocol allows establishing a maximum duration for the synchronous windows and correspondingly a maximum bandwidth for that type of traffic. Consequently, a minimum bandwidth can be guaranteed for the asynchronous traffic.

In order to maintain the temporal properties of the synchronous traffic, such as composability with respect to the temporal behavior, it must be protected from the potential interference of asynchronous

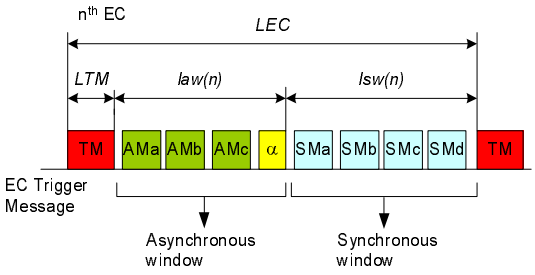

Figure 4: The Elementary Cycle (EC) in FTT-CAN.

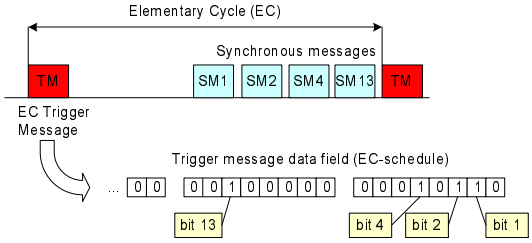

Figure 5: Master/multislave access control. Slaves produce synchronous messages according to an elementary-cycle schedule conveyed by the trigger message. If the x data bit is 1, then message x is produced in this EC; if it is 0, then message x is not produced.

requests. Thus, a strict temporal isolation between both phases is enforced by preventing the start of transmissions that could not complete within the respective window. This is achieved by removing from the network controller transmission buffer any pending request that cannot be served up to completion within that interval, keeping it in the transmission queue.

The communication requirements are held in a database located in the master node [37], the System Requirements Database (SRDB). The SRDB holds the properties of each of the message streams to be conveyed by the system, both real-time and non-real-time, as well as a set of operational parameters related to system configuration and status. This information is stored in a set of three tables: the Synchronous Requirements Table (SRT), the Asynchronous Requirements Table (ART) and the Configuration and Status Record (SCSR)

Based on the SRT, an on-line scheduler builds the synchronous schedules for each EC. These schedules are then inserted in the data area of the respective EC trigger message (see figure 5) and broadcast with it. Due to the on-line nature of the scheduling function, changes performed in the SRT at run-time will be reflected in the bus traffic within a bounded delay, resulting in a flexible behavior.

4 CAN Limitations

Judging by the large installed base of CAN controllers in significative application domains, system designers are content with CAN features, however they would be much happier if CAN could be made faster, cover longer distances, be more deterministic and more dependable [41][39]. In fact, because of its limited dependability features, there is an ongoing debate on whether the CAN protocol, with proper enhancements, can support safety-critical applications [21][31]. CAN limitations and recent development and research carried out to overcome them are discussed in detail in [39]. These include large and variable jitter, lack of clock synchronization, limited speed-distance product, flexibility limitations, data consistency issues, limited error containment, and limited support for fault tolerance. Over the years, several proposals contributed to overcome some of CAN limitations [5][47][22][18][8][43].

A specially important limitation is related with inconsistent communication scenarios [42][47], that is one of the strongest impairments to achieve high dependability over CAN. These scenarios are due to the protocol specification and may reveal themselves both as inconsistent message omissions, i.e., some nodes receive a given message while others do not, and as inconsistent message duplicates, i.e., some nodes receive the same message several times while others receive only once. Inconsistent communication scenarios make distributed consensus in its different forms, e.g., membership, clock synchronization, consistent commitment of configuration changes, or simply the consistent perception of asynchronous events, more difficult to attain.

In principle, and according to the error detection and signaling capabilities of CAN, any frame which suffers an error would be consistently rejected by all the nodes of the network. However, some failure scenarios have been identified [47][41] that can lead to undesirable symptoms such as inconsistent omission failures and duplicate message reception. The probability of those error scenarios depends on an important factor, the CAN bit error rate. Rufino, et al., presented [47] an analysis based on the assumption that the *bit error rate* varies from 10^{-4}, in case of an aggressive environment, to 10^{-6} in the case of a benign environment. The results obtained, based in these assumptions, for IMO/h and IMD/h are rather high and would became a serious impairment for using CAN (or CAN based protocols) in safety-critical applications.

Over the years several studies have been conducted [34][25][10] to assess the worst case response time of CAN messages under channel errors. These studies used generic error models, that take into account the nature of the errors, either single bit errors or burst of errors, and their minimum inter-arrival time. However, no error statistics were provided to support the error models.

In CAN, as well as in CANopen and DeviceNet, the automatic message retransmission, upon transmission error is not disabled. Conversely, in TTCAN the automatic message retransmission of CAN is disabled, while in FTT-CAN [2] it is restricted. In TTCAN and FTT-CAN, an error in a given message does not affect the response time of others, and the error detection and signalling of CAN would normally ensure that the error would be consistently detected by all network nodes except in the cases where inconsistent message delivery

	CAN		TTCAN	FTT-CAN	
bit error rate	$\Delta t = 5ms$			$\rho = 0.5$	
	IMD/h	IMO/h	IMO/h	IMD/h	IMO/h
2.6×10^{-7}	7.59	1.05×10^{-9}	7.59	3.79	3.79
3.1×10^{-9}	8.93×10^{-2}	1.24×10^{-11}	8.93×10^{-2}	4.46×10^{-2}	4.46×10^{-2}
3.0×10^{-11}	8.75×10^{-4}	1.22×10^{-13}	8.75×10^{-4}	4.37×10^{-4}	4.37×10^{-4}

Table 2: Estimated rates of IMO per hour in CAN, TTCAN and FTT-CAN.

may occur.

Inconsistent message reception scenarios are much more frequent in the case of TTCAN [42] than in native CAN and their frequency depends on the ***bit error rate*** **(BER)** of the CAN bus. The assumptions concerning the BER made by Rufino et al., although realistic in other networks, seemed somewhat pessimistic considering the specific case of CAN and specially the characteristics of the CAN physical layer. This lead to the design of an experimental setup to measure real values of CAN bit error rate in aggressive ($BER = 2.6 \times 10^{-7}$), industrial ($BER = 3.1 \times 10^{-9}$) and benign ($BER = 3.0 \times 10^{-11}$) environments [20]. Experimental results from a set of experiments showed that BER was 2.6×10^{-7} in an aggressive environment (CAN network placed near a high-frequency arc-welding machine) , 3.1×10^{-9} in a normal environment (CAN network at an industrial production line) and 3.0×10^{-11} at a benign environment (CAN network at the university laboratory).

Based on these results one can compute the probability of inconsistency scenarios in CAN networks. In [47], the probability of inconsistency scenarios in CAN is calculated as a function of the channel bit error rate ($\boldsymbol{B}$). In [42] the analysis is adapted to include the cases with no retransmissions upon error (TTCAN). Please refer to [22] for a detailed analysis and discussion of these issues.

Table 2 presents the probability of inconsistencies both for CAN (including CANopen and DeviceNet), TTCAN and FTT-CAN. Results from Table 2 are based in the same assumptions of Rufino's work considering a node failure rate $\lambda = 10^{-4}$ failures per hour.

Observing Table 2, it seems that in native CAN the occurrence of inconsistent message omissions has a lower probability than previously assumed. At least in the environments considered in the experiments. In fact it is below the 10^{-9} threshold usually accepted for safety-critical applications [30]. However, the probability of inconsistent message duplicates (messages are eventually delivered but they could be out of order) is still high enough to be taken into account.

Concerning both TTCAN and FTT-CAN, one cannot neglect inconsistent message omissions because they are rather frequent. However, it should be stressed that this not means that native CAN is more dependable than TTCAN or FTT-CAN, since native CAN does not bound automatic retransmissions that

may origin deadline losses. That is, the probability of inconsistent message omissions is lower in native CAN, but the probability of a message miss its deadline is higher.

4.1 Consequences of physical faults of the nodes

CAN protocol does not restrict the *failure semantics* of the nodes, so that they may fail in arbitrary ways. Some of these failures are automatically handled by CAN's native implementation of error detection capabilities and automatic fail-silence enforcement, described previously, leading the erroneous node to a state, the bus-off state, where it is unable to interfere with other nodes. However, in some specific situations, these mechanisms do not fully contain the errors within the nodes. Specifically, a CAN node only reaches the bus-off state (fail silence) after a relatively long period of time (when the TEC reaches 255). For example, in the case of an erratic transmitter in a 32 node CAN network at 1 Mbps, the worst-case time to bus off is 2.48 ms [46]. Moreover, a CAN node running an erroneous application can also compromise most of the legitimate traffic scheduled according to a higher layer protocol implemented in software in a standard CAN controller, simply by accessing the network at arbitrary points in time (*babbling idiot* failure mode). Notice that a faulty application running in a node with a CAN controller may transmit high priority messages at any time without causing any network errors, and consequently the CAN controller will never reach the bus-off state. An uncontrolled application transmitting at arbitrary points in time via a non-faulty CAN controller is a much severe situation than a faulty CAN controller also transmitting at arbitrary points in time because, in the first case, a non faulty CAN controller has no means to detect an erroneous application transmitting legitimate traffic. In the second case the CAN controller would enter bus-off state after a while and the error would be eventually confined. To overcome the *babbling idiot* failure mode special components located between the node and the bus, the bus guardians, have been proposed for CAN [52][9][19][17]. Bus guardians are usually adopted to enforce fail silence behavior in nodes of a distributed system because they are simpler than using node replication and some sort of voting mechanism. Apart from the cost issue, simplicity is also important because a simpler component is often more dependable than a complex one.

References

[1] L. Almeida, J. A. Fonseca, and P. Fonseca. Flexible Time-Triggered Communication on a Controller Area Network. *Proceedings of Work-In-Progress Session of RTSS'98 (*19^{th} *IEEE Real-Time Systems Symposium)*, 1998.

[2] L. Almeida, P. Pedreiras, and J. A. Fonseca. The FTT-CAN Protocol: Why and How. *IEEE Transactions on Industrial Electronics*, 49(6), 2002.

[3] Open DeviceNet Vendor Association. DeviceNet Technical Overview. `http://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00026R1.pdf`; accessed February 12, 2009.

[4] M. Barranco, G. Rodríguez-Navas, J. Proenza, and L. Almeida. CANcentrate: An active star topology for CAN networks. *Proceedings of the 5^{th} IEEE International Workshop on Factory Communication Systems (WFCS 2004)*, 2004.

[5] Manuel Barranco, Luís Almeida, and Julián Proenza. Experimental assessment of ReCANcentrate, a replicated star topology for CAN. *SAE 2006 Transactions Journal of Passenger Cars: Electronic and Electrical Systems*.

[6] S. Biegacki and D. VanGompel. The application of DeviceNet in process control. *ISA Transactions*, 35:169–176, 1996.

[7] Robert BOSCH. *CAN Specification Version 2.0*. Postfach 300240, D-7000 Stuttgart 30, 1991.

[8] I Broster. *Flexibility in Dependable Communication*. PhD thesis, Department of Computer Science, University of York, York, YO10 5DD, UK, Aug 2003.

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

[10] I. Broster, A. Burns, and G. Rodríguez-Navas. Probabilistic analysis of CAN with faults. In *Proceedings of the 23rd Real-time Systems Symposium*, pages 269–278, Austin, Texas, 2002.

[11] Gianluca Cena and Adriano Valenzano. Operating Principles and Features of CAN Networks. In *The Industrial Information Technology Handbook*, pages 1–16. 2005.

[12] CiA. *CAN Physical Layer*. CiA (CAN in Automation), 2001.

[13] CiA. *CANopen Application Layer and Communication Profile, version 4.02*. CiA (CAN in Automation), 2002. EN 50325-4 Standard.

[14] R.I. Davis, A. Burns, R.J. Bril, and J.J. Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[15] R. DeMeis. Cars sag under weighty wiring. *Electronic Times*, 10/24/2005.

[16] Konrad Etschberger. *Controller Area Network - Basics, Protocols, Chips and Applications*. IXXAT Automation, 2001.

[17] J. Ferreira, L. Almeida, and J. Fonseca. Bus guardians for CAN: a taxonomy and a comparative study. *Proceedings of the Latin-American Workshop on Dependable Automation System, satellite workshop of the Second Latin-America Symposium on Dependable Computing (LADC 2005)*, pages 3–10, 2005.

[18] J. Ferreira, L. Almeida, J. Fonseca, P Pedreiras, E. Martins, G. Rodriguez-Navas, J. Rigo, and J. Proenza. Combining Operational Flexibility and Dependability in FTT-CAN. *IEEE Transactions on Industrial Informatics*, 2(2):95–102, 2006.

[19] J. Ferreira, L. Almeida, Ernesto Martins, P. Pedreiras, and J. Fonseca. Components to Enforce Fail-Silent Behavior in Dynamic Master-Slave Systems. *Proceedings of SICICA'2003 5^{th} IFAC International Symposium on Intelligent Components and Instruments for Control Applications*, July 2003.

[20] J. Ferreira, A. Oliveira, P. Fonseca, and J. A. Fonseca. An experiment to assess bit error rate in CAN. *RTN 2004 - 3^{rd} Int. Workshop on Real-Time Networks satellite held in conjunction with the 16^{th} Euromicro Intl Conference on Real-Time Systems*, June 2004.

[21] J. Ferreira, P. Pedreiras, L. Almeida, and J. Fonseca. The FTT-CAN protocol: improving flexibility in safety-critical systems. *IEEE Micro (special issue on Critical Embedded Automotive Networks)*, 22(4):46–55, 2002.

[22] Joaquim Ferreira. *Fault-Tolerance in Flexible Real-Time Communication Systems*. PhD thesis, University of Aveiro, Portugal, May 2005.

[23] P. Ferriol, F. Navio, J. Pons, J. Proenza, and J. Miro-Julia. A double CAN architecture for fault-tolerant control systems. In *5th International CAN Conference, ICC'98*, San Jose CA, Nov 1998.

[24] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther. Time triggered communication on CAN. In *Proceedings of 7^{th} International CAN Conference*. CAN in Automation GmbH, Oct 2000.

[25] H. Hansson, C. Norström, and S. Punnekkatt. Integrating reliability and timing analysis of CAN-based systems. *Proceedings of IEEE Workshop on Factory Communications Systems (WFCS-2000)*, 2000.

[26] H. Hilmer, H.-D. Kochs, and E. Dittmar. A fault-tolerant communication architecture for real-time control systems. In *Proc. IEEE Int. Workshop on Factory Communication Systems*, Barcelona, Spain, Oct 1997.

[27] International Standards Organisation. *ISO 11898. Road Vehicles—Interchange of digital information—Controller area network (CAN) for high speed communication*, 1993.

[28] ISO. Road vehicles - Controller area network (CAN) - Part 4: Time triggered communication, 2001.

[29] IXXAT. FO-Star-Coupler. *http://www.ixxat.de/fo-star-coupler_en,7460,5873.html; accessed February 21, 2005.*, 2005.

[30] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer Academic Press, 1997.

[31] H. Kopetz. A comparison of CAN and TTP. Technical Report 1998, Technishe Universitat Wien, Austria, 1998.

[32] Wolfhard Lawrenz. *CAN System Engineering : From Theory to Practical Applications.* Springer, 1997.

[33] B. Müller, T. Führer, F. Hartwich, R. Hugel, and H. Weiler. Fault tolerant ttcan networks. In *Proceedings of* 8^{th} *International CAN Conference.* CAN in Automation GmbH, Oct 2002.

[34] N. Navet, Y.-Q. Song, and F. Simonot. Worst-case deadline failure probability in real-time applications distributed over controller area network. *J. Syst. Archit.*, 46(7):607–617, 2000.

[35] Nicolas Navet and Francoise Simonot-Lion (Editors). *Automotive Embedded Systems Handbook.* CRC Press, 2009.

[36] Dominique Paret. *Le Bus CAN.* DUNOD, 1997.

[37] Paulo Pedreiras. *Supporting Flexible Real-Time Communication on Distributed Systems.* PhD thesis, University of Aveiro, Portugal, July 2003.

[38] M. Peraldi and J. Decotignie. Combining real-time features of local area networks FIP and CAN. *Proc. of ICC'95 (*2^{nd} *International CAN Conference)*, 1995.

[39] Juan Pimentel, Julian Proenza, Luís Almeida, Guillermo Rodríguez-Navas, Manuel Barranco, and Joaquim Ferreira. Dependable Automotive CAN Networks. In *Automotive Embedded Systems Handbook, N. Navet and F. Simonot-Lion (ed).* CRC Press, 2009.

[40] Juan R. Pimentel and José Alberto Fonseca. FlexCAN: A Flexible Architecture for Highly Dependable Embedded Applications. *RTN 2004 -* 3^{rd} *Int. Workshop on Real-Time Networks sattelite held in conjunction with the* 16^{th} *Euromicro Intl Conference on Real-Time Systems*, June 2004.

[41] J. Proenza and J. Miro-Julia. MajorCAN: A modification to the Controller Area Network to achieve Atomic Broadcast. *IEEE Int. Workshop on Group Communication and Computations. Taipei, Taiwan*, 2000.

[42] Guillermo Rodríguez-Navas and Julián Proenza. Analyzing Atomic Broadcast in TTCAN Networks. In *Proceedings of the* 5^{th} *IFAC International Conference on Fieldbus Systems and their Applications (FET 2003), Aveiro, Portugal*, pages 153–156, 2003.

[43] G. Rodríguez-Navas, J. Proenza, and H. Hansson. An UPPAAL Model for Formal Verification of Master/slave Clock Synchronization over the Controller Area Network. In *Proceedings of the 6th IEEE International Workshop on Factory Communication Systems, Torino, Italia*, 2000.

[44] M. Rucks. Optical layer for CAN. In *Proceeding of the* 1^{st} *International CAN Conference*, volume 2, pages 11–18, Mainz, Germany, 1994. CiA.

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

[46] J. Rufino and P. Veríssimo. Hard real-time operation of CAN. *CSTC Technical Report RT-97-02*, 1997.

[47] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodigues. Fault-tolerant broacast in CAN. *Digest of Papers,* 28_{th} *International Symposium on Fault Tolerant Computer Systems*, pages 150–159, 1998.

[48] Valter Silva, Joaquim Ferreira, and José Fonseca. Dynamic Topology Management in CAN. In *Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation*. IEEE, 2006.

[49] Valter Silva, José Fonseca, and Joaquim Ferreira. Master Replication and Bus Error Detection in FTT-CAN with Multiple Buses. In *Proceedings of the 12th IEEE Conference on Emerging Technologies and Factory Automation*. IEEE, 2007.

[50] K. Tindell and A. Burns. Guaranteeing message latencies on controller area network (CAN). In *Proceedings of the 1st International CAN Conference*, pages 1–11, 1994.

[51] K. Tindell, A. Burns, and A. J. Wellings. Calculating controller area network (CAN) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

[52] K. Tindell and H. Hansson. Babbling idiots, the dual-priority protocol, and smart CAN controllers. In *Proceedings of the 2nd International CAN Conference*, pages 7.22–28, 1995.