The FTT-CAN Protocol: Why and How

Luís Almeida, *Member, IEEE*, Paulo Pedreiras, and José Alberto G. Fonseca, *Member, IEEE*

***Abstract*—The requirement for flexible operation is becoming increasingly important in modern industrial systems. This requirement has to be supported at all system levels, including the field level in process industry, as well as the cell and machine control levels in manufacturing industry, where fieldbus-based communication systems are commonly found. Furthermore, typical applications at these levels require both time- and event-triggered communication services, in most cases under stringent timing constraints, to convey state data in the former case and alarms and management data in the latter. However, neither the requirement for flexible operation under guaranteed timeliness nor for joint support of time and event-triggered traffic are efficiently fulfilled by most of existing fieldbus systems.**

This paper presents a new protocol, Flexible Time-Triggered communication on Controller Area Network, which fulfills both requirements: it supports time-triggered communication in a flexible way as well as being an efficient combination of both time- and event-triggered traffic with temporal isolation. These types of traffic are handled by two complementary subsystems, the Synchronous and the Asynchronous Messaging Systems, respectively. The paper includes a justification for the new protocol as well as its description and worst case temporal analysis for both subsystems. This analysis shows the capability of the protocol to convey real-time traffic of either type.

***Index Terms*—Distributed computer control systems, fieldbus systems, flexible real-time communication, real-time distributed systems, real-time scheduling.**

I. Introduction

THE requirement for flexibility is becoming increasingly important in industrial systems motivated by the need to reduce the costs of setup, configuration changes, and maintenance [22], [24]. This requirement naturally extends to all system levels including the field level in process industries and the cell and machine control levels in manufacturing industries, where fieldbus-based distributed computer control systems can be found. Particularly concerning the fieldbus system, flexibility implies dynamic communication requirements meaning that the online addition, removal, and adaptation of message streams must be supported. On the other hand, most of the data exchanges handled by the fieldbus are also subject to stringent timing constraints arising from control and monitoring requirements. Unfortunately, flexibility and timeliness have typically been considered separately and most of the fieldbuses available today favor either one aspect or the other [24], i.e., either time-constrained services are guaranteed sacrificing flexibility or such guarantees are sacrificed in exchange for higher flexibility.

Manuscript received March 27, 2001; revised October 3, 2001. Abstract published on the Internet September 13, 2002. This work was supported in part by the Portuguese Government under Grant PRAXIS XXI/BD/21679/99.

The authors are with DET-IEETA, Universidade de Aveiro, 3810-193 Aveiro, Portugal (e-mail: lda@det.ua.pt; pedreiras@alunos.det.ua.pt; jaf@det.ua.pt).

Digital Object Identifier 10.1109/TIE.2002.804967

Another requirement typically found in fieldbus systems is the capacity to deliver both time- and event-triggered communication services under timing constraints. The former ones are well suited to convey periodic updates of state data whilst the latter ones are more adapted to convey alarms and management data. Again, existing fieldbus systems privilege either one or the other type of services. In systems eminently time-triggered, event-triggered services are either nonexisting or handled inefficiently in terms of either response time or network utilization. On the other hand, in systems eminently event-triggered, interesting properties of time-triggered services such as composability with respect to the temporal behavior are normally lost [13].

Therefore, adequate choices of communication paradigms and protocols are required to achieve the desired combination of both time and event-triggered services in an efficient, flexible, and timely way. This paper will start by discussing related communication paradigms to show that existing fieldbus systems do not generally support such combination in an efficient and flexible way. This fact is used to justify the development of a new protocol, Flexible Time-Triggered communication on Controller Area Network (FTT-CAN), which is presented in the remainder of the paper. A worst case response time analysis for communication requests is also carried out, showing the protocol ability to deliver real-time communication services.

II. Communication Paradigms

During the past several years, the fieldbus research community has known several debates which opposed different concepts and paradigms [25], e.g., static versus dynamic, synchronous versus asynchronous, deterministic versus non-deterministic, time-triggered versus event-triggered, etc. This section will revisit two particular debates, which clearly relate to the requirements of flexibility, timeliness and efficiency.

A. Static versus Dynamic Traffic Scheduling

The underlying traffic scheduling paradigm used in a fieldbus system has a direct impact both on the guarantees for timely behavior as well as on the fieldbus operational flexibility. Two main paradigms can be identified: static scheduling, where the communication requirements are fixed throughout all system operation, and release and transmission times are known at pre-run time; and dynamic scheduling in which case the communication requirements may change at run time. While the former paradigm is particularly adapted to support timeliness, because it allows complex offline schedulability analysis to be carried out, its level of flexibility is very low (at most, only changes between a limited number of predefined operational modes are allowed). On the other side, dynamic scheduling supports the desired level of flexibility but, to support timeliness guarantees, an online admission control based on an adequate schedulability analysis must be used. Otherwise, the system may do its best to meet the timing constraints associated to the communication requirements but with no timeliness guarantees.

Concerning task scheduling, each of these two paradigms can be further divided in two categories [20]: static table-based and static priorities preemptive scheduling on one hand, dynamic best-effort and dynamic planning-based scheduling on the other. These paradigms can also be found concerning message scheduling in fieldbuses, except that preemption is normally not considered. Examples of fieldbuses relying on static table-based scheduling are: WorldFIP [7], [8] concerning periodic exchanges of identified variables, as well as TTP [12] and TT-CAN [9] that use distributed tables. In these cases, the communication requirements are fixed at pre-run time and an explicit schedule is built that is used at run time to timely initiate the data exchanges. Notice that the communication requirements cannot be changed by the application at run time. However, in the majority of the existing fieldbus systems, the communication requirements can be modified by the application at run time without any admission control and thus without timeliness guarantees, i.e., dynamic best-effort scheduling. As examples, consider ProfiBus, P-Net [7], [8], WorldFIP concerning aperiodic communication services, and most CAN-based systems. Nevertheless, it is still possible to obtain timeliness guarantees for the traffic in these fieldbus systems by using adequate analyzes, e.g., [23] for CAN, [26] for ProfiBus, and [27] for P-Net. Notice, however, that those analyzes are normally executed offline, only. At run time the fieldbus handles the data exchanges in a highest priority first fashion, i.e., static priorities online scheduling. Therefore, timeliness guarantees remain valid as long as the communication requirements are kept unchanged by the application at run time.

Finally, the dynamic planning-based scheduling paradigm allows combining flexibility and timeliness guarantees by the use of online admission control. In the case of a fieldbus, any submitted change to the current communication requirements is subject to the admission control before it is accepted. Such control consists on verifying, online, whether the timeliness of the resulting traffic can be guaranteed, for example by using the analyzes referred above. The change is accepted, only, if such guarantee is given, otherwise it is rejected. One example is the proposal done by Rössler and Geppert [21] for inclusion in CAL [5], a CAN-based communication system. Their proposal was to modify the DBT (distributor) protocol, through which identifiers are allocated to messages, so that new message streams can be added online if the timeliness of the communication system is not jeopardized.

One fieldbus specification that already considers the dynamic planning-based scheduling paradigm is the Foundation Fieldbus-H1 [7], [8]. In this case, a particular node called the Link Active Scheduler (LAS), controls the communication in each link by making use of a schedule table and tokens. When the LAS uses one of the specified scheduling profiles known as *dynamic*, then it can accept change requests to the scheduling table, which are accepted only if the resulting schedule is feasible. However, the standard does not specify how to implement such dynamic profile.

B. Event- versus Time-Triggered Communication

Another debate concerns the paradigm used for application architectures with event-triggered ones being opposed to those based on time triggering [11]. One of the main aspects of this debate concerns the communication infrastructure in distributed applications. This discussion has been fostered by the appearance of the Time-Triggered Protocol—TTP [12] that highlighted the advantages of that paradigm in real-time communication systems. More recently, such paradigm has also been addressed by the ISO Technical Committee TC22/SC3/WG1 that, in 1999, set up a task force (TF6) to work on the definition of a new CAN-based standard, TT-CAN, which is a time-triggered profile for CAN.

Event-triggered communication does seem more ergonomic and even more resource efficient. However, when worst case requirements are considered, that efficiency is not verified. Since events are asynchronous by nature, a typical worst case assumption is that all events that must be handled by the system will occur simultaneously. In order to cope with such situation in a timely fashion the required amount of resources (e.g., network bandwidth) is very high. On the contrary, the time-triggered approach forces the communication activity to occur at predefined instants in time at a rate determined by the dynamics of the environment under control. One of the features of this approach is that it allows relative phase control among the streams of messages to be transmitted over the communication system. By using this feature, messages of different streams can be set out of phase allowing a reduction on the number of messages that become ready for transmission simultaneously. This feature is responsible for one of the most important properties of time-triggered communication as stressed by Kopetz [13], i.e., the support for composability with respect to the temporal behavior. This property assures that, when two subsystems are integrated to form a new system, the temporal behavior of each of them will not be affected. This does not hold true for event-triggered communication. In this case, the level of contention at the network access that each subsystem *feels* before integration is always increased upon integration due to the traffic generated by the other subsystems.

Furthermore, the relative phase control allowed by the time-triggered approach may lead to two other positive effects. Firstly, it improves the control over the transmission jitter felt by periodic message streams. Secondly, it supports higher network utilization with timeliness guarantees.

Therefore, when considering worst-case requirements the time-triggered approach is more resource efficient than the event-triggered one. However, when considering average-case requirements, time-triggered communication is considerably greedy when compared to event-triggered one. Consequently, by dimensioning a system according to its worst case requirements, as typical in hard real-time systems, the time-triggered approach tends to be less expensive than the event-triggered one. Nevertheless, since the average network utilization of event-triggered systems is normally lower, such systems can easily support other types of communication with less stringent or no timing constraints (e.g., traffic associated with the management of either remote nodes or network) without any additional cost. This fact can have a positive impact on the overall efficiency of the communication system utilization, reducing its exploitation costs.

Apart from the above considerations on network utilization, it is commonly accepted [25], [18] that time-triggered communication is well adapted to control applications that typically require regular transmission of state data with low, or bounded, jitter (e.g., motion control, engine control, temperature control, position control). On the other hand, event-triggered communication is well adapted to the monitoring of alarm conditions that are supposed to occur sporadically and seldom, and also to support asynchronous nonreal-time traffic, e.g., for global system management.

C. Combining Event- and Time-Triggered Traffic

Despite their different characteristics, many applications do require joint support for both event- and time-triggered traffic and, thus, a combination of both paradigms in order to share their advantages is desirable. An important aspect is that temporal isolation of both types of traffic must be enforced or, otherwise, the asynchrony of event-triggered traffic can spoil the properties of the time-triggered one. This isolation is achieved by allocating bandwidth exclusively to each type of traffic. A typical implementation makes use of bus-time slots called elementary cycles, or microcycles (e.g., [19]), containing two consecutive phases dedicated to one type of traffic each. The bus time becomes, then, an alternate sequence of time-triggered and event-triggered phases. The maximum duration of each phase can be tailored to suit the needs of a particular application. If each type of traffic is forced to remain within the respective phase then temporal isolation is guaranteed. This concept is used, for example, in the WorldFIP fieldbus. However, since this fieldbus uses a centralized MAC protocol (master–slave), the handling of event-triggered (aperiodic) traffic is relatively inefficient requiring a considerable amount of bandwidth to allow the master node (arbitrator) to become aware of, and process, aperiodic requests. First, the master has to poll the nodes for the existence of aperiodic requests to be served, which is normally carried out using the periodic traffic coming from each node. Then, when a node signals that it has pending aperiodic requests, the master has to poll the node for the identification of the individual requests and finally process them one at a time.

In the Foundation Fieldbus-H1, a somewhat similar scheme is used. The LAS contains the schedule for the time-triggered traffic but not necessarily organized in elementary cycles. This node grants the other nodes, Link Masters (LMs), the permission to control the bus and transmit event-triggered messages during precise time windows, only, that do not overlap with the time used by the time-triggered messages. The LAS implements a virtual token ring to control the order by which LMs access the network. The sequence in the ring can be any, in order to control the distribution among the several LMs of the bandwidth available for aperiodic communication. This token-based method is also relatively inefficient for two reasons. Firstly, the tokens still consume bandwidth, and secondly, nodes with pending aperiodic communication requests have to wait for the token even when the remaining nodes in the ring list have no requests.

In the case of TT-CAN, a time-division multiple-access (TDMA)-based technique is followed, similar to the one proposed in [28]. In this case, a static cyclic table organized as a matrix is used, containing a sequence of well-determined windows. These can either be exclusive or arbitration windows and their sequence in the cycle (TDMA round) can be any. However, there are several practical constraints that must be observed when building the table. For example, all the windows in the same column must be of equal width and type, and the number of lines must be a power of 2. The exclusive windows are dedicated to the transmission of a single time-triggered message, each. There is no bus contention in these windows. On the other hand, arbitration windows can be shared by several event-triggered messages and potential collisions are sorted out by the original MAC protocol of CAN, based on the carrier-sense multiple-access (CSMA) technique with bit-wise nondestructive collision resolution. The network controllers execute a further access control to prevent the transmission of event-triggered messages to extend beyond the respective window thus assuring temporal isolation. The fact that there is a CSMA-based MAC protocol that resolves collisions at bus access during the arbitration windows greatly simplifies the handling of event-triggered traffic, resulting in a higher efficiency. Notice that there is no need for token-passing or master requests as in the previous cases.

On the other hand, a pure TDMA approach is used in TTP/C, with exclusive slots, only, to transmit each message within the TDMA round. The schedule is static and each message transmission is guaranteed to fit within the respective slot. The support of time-triggered traffic is obvious. On the contrary, event-triggered traffic can only be supported by pre-allocating a number of slots for the transmission of eventually pending event-triggered messages. However, these slots are also dedicated and thus, at a given instance, if no transmission request for the respective message is pending the slot is wasted, i.e., unused. This time-based polling mechanism for each event-triggered message causes these ones to be undifferentiated from the time-triggered traffic inheriting the properties referred in the previous section, particularly high efficiency under worst case requirements and low efficiency under average-case requirements whenever these are substantially lower than the former ones. Therefore, for the purpose of this paper, it will be considered that TTP/C supports time-triggered traffic, only.

In many other fieldbus systems, it is possible to specify cyclic time-triggered data exchanges but with no temporal isolation from the event-triggered traffic, e.g., ProfiBus, P-Net, DeviceNet [6]. This means that the properties of time-triggered traffic are lost, particularly the relative phase control among periodic data streams and, consequently, jitter control as well as composability with respect to the temporal behavior. In fact, from the network point of view, in such systems all the traffic is handled as event-triggered. Nevertheless, this does not mean that these systems handle such traffic efficiently. This strongly depends on the MAC protocol used by the fieldbus system. For example, CSMA-based protocols are efficient with respect to

TABLE I
PROPERTIES OF SOME FIELDBUS SYSTEMS

Fieldbus	Scheduling Paradigm[1]	Dynamic comm. req.	ET traffic	TT traffic	TT/ET isolation	Efficient ET handl.
WorldFIP	ST+(DBE/SP)	N	Y	Y	Y	-
FF-H1	DP+(DBE/SP)	Y[2]	Y	Y	Y	-/+
TTP/C	ST	N	N	Y	------	------
TT-CAN	ST+(DBE/SP)	N	Y	Y	Y	+
ProfiBus	DBE/SP	Y	Y	Y[3]	N	-/+
P-Net	DBE/SP	Y	Y	Y[3]	N	-/+
DeviceNet	DBE/SP	Y	Y	Y[3]	N	+
FTT-CAN	DP+(DBE/SP)	Y	Y	Y	Y	+

[1]ST– Static Table-Driven, SP– Static Priorites-Driven, DBE– Dynamic Best Effort, DP– Dynamic Planning-Based
XX+YY → XX for TT traffic and YY for ET, (XX/YY) → XX or YY for ET traffic depending on pre-analysis.

[2]"Y" assuming a *dynamic scheduling profile*, only ("N" for all other profiles)

[3]Automatic Cyclic Transmissions

handling event-triggered traffic since nodes try to initiate transmission as soon as the respective request is received from the application (e.g., DeviceNet and other CAN-based systems). On the other hand, token-based MAC protocols are not so efficient because a node always has to wait for the token despite having pending transmission requests and also because of the bandwidth used by the tokens (e.g., ProfiBus and P-Net, among masters). The situation is worse with master–slave-based MAC protocols since all slaves have to be polled by the master so that it becomes aware of slaves transmission requests and grants them the necessary right to transmit (e.g., ProfiBus and P-Net, between each master and slave node).

D. Why a New Protocol?

From the above discussions, it can be seen that the joint support for both time- and event-triggered traffic is advantageous for many applications. However, existing fieldbus protocols either do not support both types of traffic (e.g., TTP/C), or both types are supported but without temporal isolation (e.g., ProfiBus, P-Net, DeviceNet). In the cases where temporal isolation is enforced, the event-triggered traffic is handled inefficiently (e.g., WorldFIP, Foundation Fieldbus-H1) and/or the time-triggered traffic is specified statically, thus not supporting operational flexibility (e.g., TT-CAN).

The FTT-CAN protocol herein presented addresses these issues and fulfills the requirements for flexibility, timeliness and efficient combination of time and event-triggered traffic. Recently, another communication system meant for distributed embedded systems has been proposed, FlexRay [10], that aims at fulfilling similar requirements. However, as the current specification states, its time-triggered traffic must still be defined statically. Due to lack of complete knowledge about this protocol at the time of writing this paper, it has not been further considered. Table I summarizes the properties of several fieldbus systems as discussed above, along this section. It already includes the FTT-CAN protocol in order to allow a fast comparison with existing fieldbuses. In the remainder of the paper, this protocol will be presented, supporting the properties claimed in Table I.

III. INTRODUCTION TO FTT-CAN

The basis for the FTT-CAN protocol was 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 online and centrally in a particular node called master. This feature facilitates the online admission control of dynamic requests for periodic communication because the respective requirements are held centrally in just one local table. With online admission control, the protocol supports the time-triggered traffic in a flexible way, under guaranteed timeliness (dynamic planning-based scheduling paradigm).

Furthermore, there is another feature that clearly distinguishes this protocol from other proposals concerning time-triggered communication on CAN [18], [9] that is the exploitation of its native distributed arbitration mechanism. In those proposals, there are specific mechanisms to avoid collisions in the time-triggered traffic, either through master–slave transmission control [18] or through control of transmission instants [9] using a strictly periodic reference message (TT-CAN level 1) or with clock synchronization (TT-CAN level 2). In both cases, the original MAC of CAN is made useless, contributing to a low efficiency in the former case, due to the bandwidth taken by the master messages, and to a low flexibility in the latter case, due to the static nature of the *a priori* knowledge of all transmission instants. On the contrary, FTT-CAN takes advantage of the native MAC of CAN to reduce communication overhead and support a high efficiency and flexibility in the time-triggered traffic. The protocol relies on a relaxed master–slave transmission control in which the same master message triggers the transmission of messages in several slaves simultaneously (master/multislave). The eventual

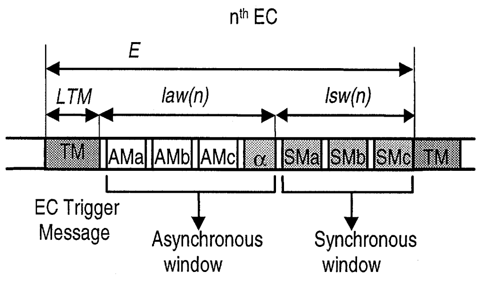

Fig. 1. Elementary cycle in FTT-CAN.

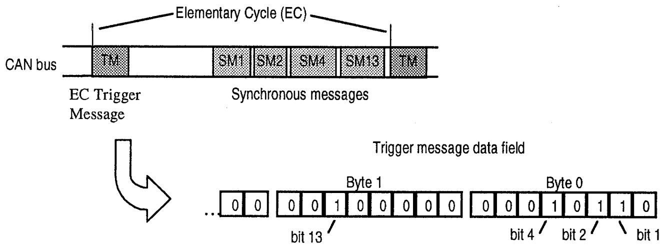

Fig. 2. EC trigger message data contents.

collisions between slaves' messages are handled by the native distributed arbitration of CAN. Moreover, the protocol also takes advantage of the CAN arbitration to handle event-triggered traffic in the same way as the original 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 TT-CAN, allows a very efficient combination of time and event-triggered traffic, resulting in low communication overhead and shorter response times.

The nomenclature used in the protocol follows. In FTT-CAN the bus time is slotted in consecutive *Elementary Cycles* (ECs) with fixed duration (E time units). All nodes are synchronized at the start of each EC by the reception of a particular message known as *EC trigger message* (TM), which is sent by a particular node called *master*. The transmission of this message, including stuff bits, takes *LTM* (constant) time units.

Within each EC the protocol defines two consecutive windows, asynchronous and synchronous, that correspond to two separate phases (Fig. 1). 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 it is transmitted synchronously with the ECs. The synchronous window of the nth EC has a duration $lsw(n)$ that is set according to the traffic scheduled for it. Such schedule is conveyed in the respective EC trigger message. Moreover, since this window is placed at the end of the EC [16], the trigger message also conveys its relative starting instant. The asynchronous window has a duration $law(n)$ equal to the remaining time between the EC trigger message and the synchronous window. The protocol allows establishing a maximum duration for the synchronous windows (*LSW*) and correspondingly a maximum bandwidth for that type of traffic. Consequently, a minimum bandwidth can be guaranteed for the asynchronous traffic.

The reason why the asynchronous window precedes the synchronous one is related with the need to decode the EC trigger message in each node [16] since it specifies which synchronous messages must be transmitted in the respective EC. This decoding takes an amount of time that strongly depends on the node processor capacity, being as large as the transmission time of one or more messages when simple 8-b microcontrollers are used, or just an insignificant fraction of time with 32-b microprocessors. Thus, if the synchronous window was defined right after the EC trigger message, the gap between this message and the first synchronous message would be hardware dependent and the corresponding bus time would be wasted. On the other hand, by defining the asynchronous window before the synchronous one, the decoding of the EC trigger message can be carried out in parallel with the transmission of asynchronous traffic, resulting in a more efficient bus utilization. Moreover, as long as an adequate minimum duration is guaranteed for the asynchronous windows, the hardware dependency is substantially reduced.

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 interference of asynchronous 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. Consequently, a short amount of idle time may appear at the end of the asynchronous window (α in Fig. 1). At the end of the synchronous window, another short amount of idle time may appear but due to variations in the stuff bits used in the physical encoding of CAN messages. However, in the remainder of the paper, the maximum number of stuff bits will always be considered.

The communication services of FTT-CAN are delivered to the application by means of two subsystems, the Synchronous Messaging System (SMS) and the Asynchronous Messaging System (AMS), that manage the respective type of traffic. The SMS offers services based on the producer–consumer model [25] while the AMS offers send and receive basic services, only. A more detailed description of both subsystems follows.

IV. SMS

The SMS conveys the time-triggered traffic herein called synchronous because it is synchronized with the ECs. In fact, the EC duration is the basic time unit used to describe the temporal attributes of the time-triggered traffic. Moreover, it is the EC trigger message that sets the pace of time progression, in a sparse time base with unit increments, specifying in its data field the synchronous messages scheduled for that EC (Fig. 2). All nodes that produce synchronous messages have to decode the EC trigger message and check whether they are producers of any of the specified messages. This checking is carried out by

TABLE II
COMMUNICATION OVERHEAD IMPOSED BY THE EC TRIGGER MESSAGE

Tx rate (Mbit/s)	**# Data Bytes** (# sync msg)	**LTM** (μs)	**E** (ms)	**Overhead** (%)
0.125	4 (32)	736	10	7.4
0.125	8 (64)	1040	10	10
1	4 (32)	92	5	1.8
1	8 (64)	130	5	2.6

scanning a local table containing the identification of the messages to be produced/consumed by this node. Upon transmission of synchronous messages, eventual collisions on bus access within the synchronous window are resolved by the native distributed MAC protocol of CAN. This mechanism allows realizing centralized scheduling with low communication overhead, i.e., one additional message per EC, only. Table II indicates the bandwidth used by the EC trigger message in four typical scenarios. For each transmission rate, the overhead can be further reduced by increasing the EC duration (E), or by reducing the data length of the EC trigger message whenever the application needs fewer synchronous messages (1 bit per message is required).

A. Synchronous Requirements Table

The temporal attributes of the synchronous messages are expressed in the Synchronous Requirements Table (SRT) that resides in the master node. Each entry describes one synchronous message stream, i.e., a sequence of messages carrying successive instances of the same entity such as readings of a temperature sensor or actuating values for an actuator (unless noted otherwise, a message stream will be referred to simply as a message). The SRT is organized as follows:

$$SRT \equiv \{SM_i(DLC_i\, C_i\, Ph_i\, P_i\, D_i\, Pr_i),\ i = 1 \cdots N_s\}. \quad (1)$$

DLC is data length in bytes (from 0 to 8), C is the respective maximum transmission time (including stuff bits), Ph stands for the relative phasing, P for period, D for deadline, and Pr for fixed priority. Both Ph, P and D are expressed as integer multiples of E, the EC duration. N_s is the number of synchronous messages (SRT entries). The CAN identifier of each message is formed by adding the index i to a pre-configured offset. The relationship between identifier and the priority Pr can be any. This relationship has an impact at the intra EC level, only, e.g., it influences the transmission order of the synchronous messages scheduled for the same EC. In a larger timescale, that relationship has no impact on the temporal behavior of the synchronous traffic, which is controlled essentially by the scheduling policy and specified priorities.

B. Flexible Scheduling of Synchronous Messages

Based on the SRT, an online scheduler builds the synchronous schedules for each EC. These schedules are then inserted in the data area of the respective EC trigger message and broadcast with it. Due to the online 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.

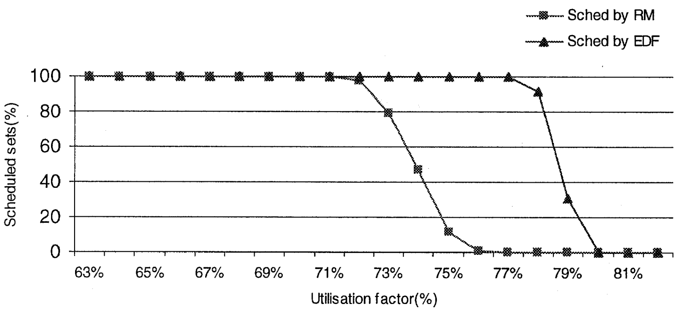

Fig. 3. Schedulability versus bus utilization under RM and EDF.

From an operational point of view, two different solutions have been used to implement the scheduler. One is the planning scheduler [2], a software-based implementation that allows reducing the processing overhead of online scheduling. This technique consists on building a static schedule table for a given period of time into the future called *plan* and rebuilding that table online at the end of each plan. The plan duration is not correlated with the messages periods and thus the memory requirements to hold a plan table are bounded and known *a priori*. The planning scheduler is particularly well suited to systems with low computational capacity nodes (e.g., based on simple 8-b microcontrollers). A negative feature of this technique is its lower responsiveness to changes in the communication requirements, when compared to normal online scheduling, arising from the static nature of each plan table. Notice that changes in the SRT, which holds those requirements, are taken into account from plan to plan, only. However, when the planning scheduler is used in the scope of FTT-CAN, the limitation on system responsiveness can be substantially reduced by using asynchronous messages to enforce the changes in communication requirements, temporarily, until they are handled by the planning scheduler [17].

The second solution that has been developed to implement the scheduling function in FTT-CAN makes use of FPGA-based scheduling co-processors. This solution provides, at a higher hardware cost, the extra computational capacity required to execute both the scheduling policy online as well as an adequate schedulability analysis. For example, the co-processor described in [14] scans the SRT and creates a new EC schedule every EC. Moreover, it is also capable of executing several schedulability tests in that interval. The result of this solution is a high degree of flexibility and responsiveness, plus a residual computational overhead, only, in the master processor.

Apart from the flexibility inherent to the use of online scheduling, as referred to above, the FTT-CAN protocol exhibits another level of flexibility related with the scheduling policy. In fact, the scheduling is carried out based on the SRT independently of the message identifiers. Thus, any scheduling policy can be easily implemented, e.g., Rate-Monotonic (RM), Deadline-Monotonic (DM), Earliest-Deadline First (EDF), Least-Laxity First (LLF), overriding the identifier-based traffic scheduling embedded in the MAC of CAN. Fig. 3 illustrates the use of FTT-CAN with RM and EDF, using 80% of the bus bandwidth allocated to the SMS ($LSW = 0.8 * E$). In particular, it shows the percentage of random message sets schedulable by both policies as a function of bus utilization and it illustrates the superior schedulability capacity of EDF over RM, as expected. With EDF, practically the whole bandwidth allocated to the SMS could be used with guaranteed schedulability. The fact that EDF does not achieve 100% utilization of the allocated bandwidth is explained by the interference of nonpreemptive message transmission.

The flexibility of using any scheduling policy is a valuable feature of FTT-CAN. For example, in [29] and [30] two techniques to implement EDF over CAN are presented, based on dynamic manipulation of message identifiers in order to obtain the desired dynamic priority scheduling at the bus access level. Furthermore, both require explicit clock synchronization among nodes. The respective implementations are fully distributed but are also relatively inefficient and computationally demanding in all nodes. The main drawbacks are a reduced number of bits to encode the dynamic priority and the need to cyclically de-queue messages to update their identifiers. Both aspects lead to a degradation of EDF performance. On the other hand, the EDF implementation based on FTT-CAN is straightforward. It requires no explicit clock synchronization, there is no need to update message identifiers and there is no extra computational demand in any node with one exception, the master, where the EDF scheduler is executed.

C. *Schedulability of Synchronous Traffic*

The scheduling model used for the synchronous traffic does not allow the transmission of messages to cross the boundary of the synchronous window. This is avoided by using inserted idle time, i.e., whenever a message does not fit completely within the synchronous window of a given EC it is delayed to the next. Consequently, the EC trigger message is always transmitted without any blocking. However, the use of inserted idle time has also a negative impact on the traffic schedulability.

In [4] a scheduling model is presented, based on fixed priorities, in which a set of periodic nonpreemptive tasks is scheduled with inserted idle time. The model, named blocking-free nonpreemptive scheduling, is very similar to the one used to schedule the synchronous traffic in FTT-CAN. Tasks periods and deadlines are integer multiples of a basic cycle duration (call it E), the execution times are always shorter than E and task activations are always synchronous with the start of a cycle. The only difference is that, in [4], the whole cycle is available to execute tasks, while in FTT-CAN the synchronous traffic is restricted to the synchronous window within each EC, with maximum length LSW.

In order to transform the FTT-CAN model into the one used in [4], so that the analysis therein presented can be used, it suffices to inflate all execution times by a factor equal to E/LSW. This is equivalent to expanding the synchronous window up to the whole EC (Fig. 4) and carries no consequence in terms of schedulability since messages scheduled for a given synchronous window will remain within the same cycle. Applying this transformation to the original set of messages SRT (1) results in a new virtual set that can be expressed as SRT^0 (2) in which all the remaining parameters but the execution times are kept unchanged

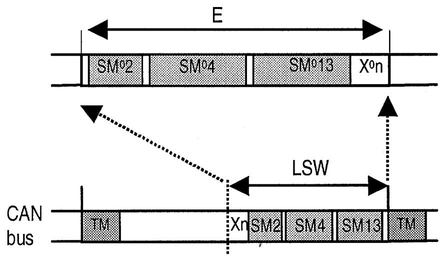

Fig. 4. Expanding the synchronous window to allow using the blocking-free nonpreemptive model.

$$SRT^0 \equiv \{SM_i^0(C_i^0\, Ph_i\, P_i\, D_i\, Pr_i),\quad C_i^0 = E/LSW * C_i,\ i = 1 \cdots N_s\}. \tag{2}$$

The results in [4] are now directly applicable over SRT^0, particularly the theorem stating that any existing analysis for fixed priorities preemptive scheduling can be used in this model if the execution times C_i^0 are replaced by C_i' as in (3), where E is the cycle duration and X^0 the maximum inserted idle time $(\max_n(X_n^0))$

$$C_i' = C_i^0 * E/(E - X^0). \tag{3}$$

Expanding (3) with the transformation in (2) and noting that $X^0 = E/LSW * X$, yields the final transformation (4) that has to be carried out over the original message transmission times, i.e., those in the SRT, so that any existing analysis for fixed priorities preemptive scheduling can be used

$$C_i' = C_i * E/(LSW - X). \tag{4}$$

However, any schedulability assessment obtained via that theorem is just sufficient, only. The reason is the pessimism introduced when using an upper bound for X. Except for a few particular situations, the exact value $X = \max_n(X_n)$ cannot be determined. Nevertheless, an upper bound is easy to obtain, e.g., the transmission time of the longest message among those that can cause inserted idle time. This can be obtained as $\max_{j=k\cdots Ns}(C_j)$ through expression (5), considering that the message set is ordered by decreasing priorities

$$\max_{j=k\cdots N_s}(C_j) \geq X = \max_n(X_n)$$
$$k\colon \sum_{i=1}^{k-1} C_i \leq LSW \wedge \sum_{i=1}^{k} C_i > LSW. \tag{5}$$

An important corollary of the theorem referred above is that Liu and Layland's utilization bound for RM [31] can be used with just a small adaptation as part of a simple online admission control for changes in the SRT incurring in very low run-time overhead. This is expressed in condition (C1)

$$U = \sum_{i=1}^{Ns}\left(\frac{C_i}{P_i}\right) < N_s\left(2^{1/N_s} - 1\right)\left(\frac{LSW - X}{E}\right)$$

$$\Longrightarrow \begin{array}{c}\text{SRT is schedulable}\\ \text{with RM under any}\\ \text{phasing.}\end{array} \tag{C1}$$

A similar line of reasoning can be followed to adapt the Liu and Layland's utilization bound for EDF. In this case, the maximum inserted idle time (X) plus the remaining amount of time in the EC outside the synchronous window ($E - LSW$) can be considered as the worst case transmission time of a virtual message ($C_v = E - LSW + X$) that is added to the original set and transmitted every EC ($P_v = E$). This virtual message will be the highest priority one in every EC and will fill in the part of the EC that cannot be used by the synchronous messages. Assume, now, that the resulting extended set, i.e., the original SRT plus the virtual message, can be scheduled preemptively. In this situation, the Liu and Layland's bound can be used (6)

$$U_v = \frac{E - LSW + X}{E} + \sum_{i=1}^{Ns}\left(\frac{C_i}{P_i}\right) \leq 1. \tag{6}$$

However, due to the extra load imposed by the virtual message, all other messages will finish transmission either in the same EC or later in this schedule than in the original one with the traffic confined to the synchronous window and with inserted idle time. Thus, if the extended set is schedulable the SRT will also be. This results in the sufficient schedulability condition (C2)

$$U = \sum_{i=1}^{Ns}\left(\frac{C_i}{P_i}\right) < \left(\frac{LSW - X}{E}\right) \Longrightarrow \begin{array}{l}\text{SRT is schedulable with}\\ \text{EDF under any phasing.}\end{array} \tag{C2}$$

Another important result presented in [4] is a new analysis based on a traffic timeline, which allows obtaining a more accurate schedulability assessment for fixed priorities scheduling, e.g., RM, DM, or other. This assessment is necessary and sufficient if both of the following conditions are verified.

1) All messages must be considered in phase, i.e., ready for transmission at a hypothetical instant $t = 0$ called critical instant (worst case phasing).
2) No lower priority message can be scheduled before a higher priority one. Otherwise, one could not guarantee that the first message instance after the critical instant suffers the worst case response time.

This analysis does not have a closed formula but instead requires the execution of a simple algorithm (Fig. 5) to obtain the worst case response times to transmission requests (Rwc_i, $i = 1 \cdots N_s$), considered as the maximum time lapse from message exact periodic activation to complete transmission. The algorithm consists in determining, for all messages, the EC where they are first transmitted after the critical instant (line 9). This is carried out EC by EC (line 2), taking into account the effective message sequence in the schedule imposed by the respective priorities (line 4). This way, the inserted idle time in each EC is accounted for with exactitude (lines 6 and 7), consequently resulting in exact worst case response times. The algorithm herein presented differs from the one in [4] in that it accumulates the load of each EC ($lsw(n)$) up to the maximum length of the synchronous window (LSW), only, and calculates the worst case response time with a resolution of one EC. At the end of each complete run of the inner for loop in line 4, $lsw(n)$ contains the effective duration of the synchronous window in the nth EC. The vector $r_{k=1\cdots Ns}(n)$ indicates the messages with transmission requests pending in the nth EC.

```
1.  for (k=1;k≤$N_s$;k++){$Rwc_k$=0; $r_k$(1)=1;}
2.  for (n=1;(n≤$D_{Ns}$/E and $Rwc_{Ns}$=0);n++){
3.          lsw(n)=0;
4.   for (k=1;k≤$N_s$;k++){
5.          $r_k$(n+1)=$r_k$(n);
6.            if (lsw(n) + $r_k$(n)*$C_k$ <= LSW) {
7.                lsw(n) = lsw(n) + $r_k$(n)*$C_k$;
8.                $r_k$(n+1)=0;
9.                if ($Rwc_k$=0) $Rwc_k$=n;
10.                   }
11.               if (n mod $P_k$/E = 0) $r_k$(n+1)=1;
12.               }
13.           }
```

Fig. 5. Pseudocode for the timeline analysis.

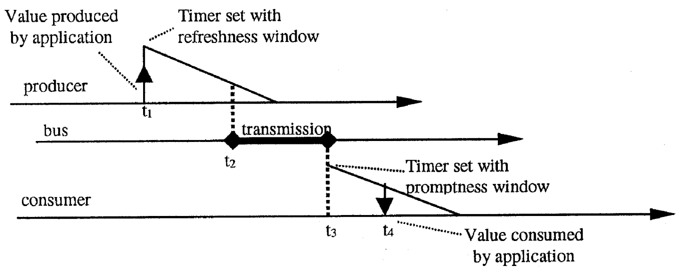
Value produced by application
Timer set with refreshness window
producer
t1
bus
transmission
t2
Timer set with promptness window
consumer
t3
t4
Value consumed by application

Fig. 6. Support for temporal accuracy.

After having determined the worst case response times for all messages, a trivial schedulability test can be carried out by comparing this time with the respective deadline. As long as both conditions referred above hold, the test supports a necessary and sufficient condition (C3)

$$Rwc_i \leq D_i \qquad \forall_{i=1\cdots N_s} \Longleftrightarrow \begin{array}{c}\text{SRT is schedulable with}\\ \text{worst case phasing.}\end{array} \tag{C3}$$

In case either condition 1) or 2) do not hold, the values of Rwc_i obtained from the algorithm in Fig. 6 may not be exact but upper bounds to the effective worst case values and, thus, the schedulability test results in a sufficient but not necessary condition.

D. Further Comments on Temporal Behavior

Apart from the scheduling-related issues there are also other aspects that have impact on the temporal behavior of the synchronous traffic in FTT-CAN. Firstly, the SMS handles the synchronous traffic with autonomous control, i.e., the transmission and reception of messages is carried out exclusively by the network interface without any intervention from the application software. The message data is passed to and from the network by means of shared buffers. This means that the network interface, in what concerns the SMS, behaves as a temporal firewall between the application and the network, since it isolates the temporal behavior of both parts, increasing the system robustness.

Secondly, the protocol supports information on the temporal accuracy of the data that is conveyed in synchronous messages. Two Boolean status variables are delivered to the application whenever the data is read from the network interface (Fig. 6). The refreshness status indicates that the delay between the data being written in the network interface and the respective message transmission is less than the refreshness window. The refreshness bit is generated at the producer side and it is coded in the message identifier so that it is transmitted together with it. On the receiver side, the promptness status indicates that the delay between the message reception and the respective data being read by the application is less than the promptness window. Whenever one of these status variables is false, it means that the respective data has either waited too long to be transmitted (false refreshness) or to be read (false promptness).

Thirdly, the SMS services available to the application software follow the producer–consumer model and are, basically, the *SMS_produce* service, i.e., writing a message in the appropriate buffer in the network interface, and the *SMS_consume* service, i.e., reading from a message buffer in the network interface. For both services there is an option to synchronize with the network traffic. This option allows controlling the cyclic execution of application software within remote nodes simply by adjusting the periodicity of the respective messages. This is the basis for a particular global system management policy named network-centric [3]. Moreover, the SMS also delivers the services required to manage the *SRT* such as *SRT_add*, *SRT_remove* and *SRT_change* message. These services automatically invoke an on-line admission control to assure a continued timely communication. However, for particular applications where such feature is not required, e.g., when changes in the *SRT* at run time are not required, then the online admission control can be disabled, saving unnecessary overhead.

V. AMS

The FTT-CAN protocol also supports asynchronous traffic for event-triggered communication, which is handled by the AMS. This subsystem works very similarly to the original CAN protocol using its native priority-based distributed arbitration mechanism and inheriting its efficiency in handling event-triggered traffic. However, on top of the CAN arbitration, the AMS contains another level of access control that allows confining this type of traffic to the asynchronous window in each EC. This is required to prevent asynchronous messages from interfering with the SMS or the EC trigger message, enforcing a strict temporal isolation between the two subsystems. The access control that establishes the beginning and end of each asynchronous window is based on time, relative to the EC trigger message. It does not require any form of control based on message exchanges, e.g., tokens, being, thus, bandwidth efficient.

Furthermore, nodes with pending asynchronous transmission requests try to transmit immediately during the asynchronous window. Outside this window such requests are kept on hold until the next window, then reentering arbitration. On average, this technique results in short response times to asynchronous requests. In the worst case, it is necessary to account for the potential blocking caused by the periods of bus exclusion, i.e., the periods of time outside the asynchronous windows.

A. AMS Communication Services

The communication activity in the AMS follows the external control paradigm, i.e., the transmission of messages takes place upon explicit requests from the application software. Such requests are issued by means of a basic service called *AMS_send*, which is a nonblocking send function with queuing. The queue is ordered first by priority, according to the message identifiers, and second by request instant (FCFS). The length of the queue within each node is set at configuration time according to the number of asynchronous message streams it may transmit and to the number of messages of the same stream that can be queued at the same time [32]. This is particularly relevant when the minimum inter-arrival time of transmission requests in a given stream is shorter that the worst case time to process a single request of that stream.

The delivery of messages to the application software is accomplished by means of a complementary basic service called *AMS_receive*, a blocking receive function that allows waiting for a specified, or unspecified message. At the receiving node, the AMS also queues the messages arriving from the network until they are retrieved with the *AMS_receive* service. The length of the queue is also set up at configuration time, similarly to the queue in the sender side. In this case, the important aspects are the number of asynchronous message streams that a node may receive as well as the number of messages of the same stream that may arrive between two consecutive retrieves. More complex and reliable exchanges, e.g., requiring acknowledge or requesting data, must be implemented at the application level, using the two basic services referred to above.

B. Asynchronous Traffic Scheduling

From a traffic-scheduling point of view, the AMS follows a dynamic best-effort paradigm. In fact, its current version does not include an embedded mechanism to perform online admission control of this type of traffic. However, for a given set of communication requirements, it can be shown that the worst case response time to asynchronous requests is upper bounded [16], [32], thus supporting the use of asynchronous messages to convey real-time data, e.g., alarms. The basic scheduling policy is directly inherited from the original CAN protocol, i.e., priority driven, with fixed priorities expressed as message identifiers. Furthermore, the scheduling uses inserted idle time (α in Fig. 1) to enforce a strict temporal isolation between the two types of traffic, and exclusions to represent the periods of the ECs outside the asynchronous windows, where asynchronous messages cannot be transmitted.

The bandwidth available to the AMS is the one left unused by the SMS (synchronous messages) and the EC trigger messages. Thus, the heavier the synchronous load is, the shorter becomes the AMS communication capacity. There is, however, as referred to in Section III, the possibility to guarantee a minimum bandwidth available to the AMS by establishing a maximum duration for the synchronous windows (*LSW*). The duration of the asynchronous window in the nth EC ($law(n)$) can be computed

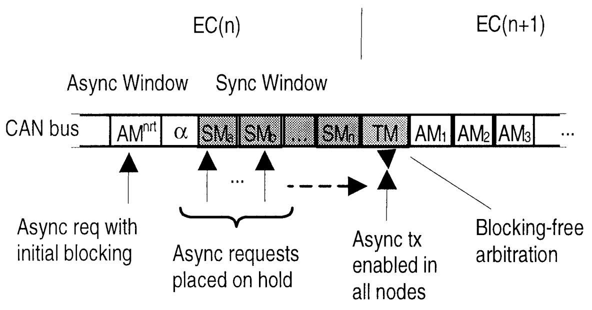
EC(n)
EC(n+1)
Async Window
Sync Window
CAN bus
AM^nrt
α
SMa
SMb
...
SMn
TM
AM1
AM2
AM3
...
Async req with initial blocking
Async requests placed on hold
Async tx enabled in all nodes
Blocking-free arbitration

Fig. 7. Avoiding chained blocking in the start of asynchronous windows.

within the algorithm in Fig. 5 by inserting expression (7) in between lines 11 and 12

$$law(n) = \begin{cases} E - LTM - lsw(n), & lsw(n) + C_s < LSW \\ E - LTM - LSW, & lsw(n) + C_s \geq LSW. \end{cases} \tag{7}$$

The lower expression in (7) allows accounting for possible idle time insertion (upper bounded by C_s) in the construction of the synchronous schedule for the respective EC. The value of C_s is the one given by expression (5). This correction is important for the schedulability analysis of the asynchronous traffic because it allows considering the critical instant for the response time to asynchronous requests as the EC in which all synchronous messages are released simultaneously.

A particular aspect that has a considerable impact on the temporal behavior of the AMS is the synchronization among all nodes in the start of each asynchronous window. Without such synchronization, a possible blocking could occur in the start of every window degrading the response time to higher priority asynchronous requests. In order to avoid this type of chained blocking, the transmission of pending asynchronous messages is enabled during the transmission of the EC trigger message that immediately precedes the next asynchronous window (Fig. 7). This detail ensures that pending asynchronous messages from different nodes will enter arbitration simultaneously after the EC trigger message, respecting the messages priorities and avoiding blocking. Consequently, an asynchronous message may suffer blocking from another lower priority message once, at most, when the send command is issued. Then, the message enters arbitration and there will be no further blocking even if the arbitration process lasts for several asynchronous windows until the message is effectively transmitted. This aspect is not commonly dealt with in other protocols that use shared windows for event-triggered traffic, such as TT-CAN, in which there may exist one blocking within each arbitration window.

C. Schedulability of Asynchronous Traffic

The schedulability of the asynchronous traffic in FTT-CAN has been studied in [16] and further improved in [32]. The analysis therein presented is based on the determination of worst case response times. It follows closely the one in [23] for the original CAN protocol but introduces a few modifications to allow coping with inserted idle time and exclusions.

The set of real-time asynchronous communication requirements is held in a table named ART—Asynchronous Requirements Table (8)

$$ART \equiv \{AM_i(DLC_i\, C_i\, mit_i\, D_i\, Pr_i),\ i = 1 \cdots N_a^{RT}\}. \tag{8}$$

Each entry in this table describes one asynchronous message stream, which must always be of a sporadic nature, i.e. there is a minimum interarrival time (mit) that must elapse between consecutive messages of the same stream. The parameters DLC, C and D are equivalent to those of the synchronous messages (1) except that this deadline is not an integer multiple of E. The parameter Pr is the message priority, which is directly expressed as a CAN identifier. N_a^{RT} stands for the number of real-time asynchronous message streams.

Notice that there may exist more nonreal-time asynchronous messages, which, for the sake of flexibility, are not constrained except by the use of an adequate identifier with lower priority. These messages will generically be referred to as AM^{nrt}.

The following analysis does not consider message queuing, at the sender, neither de-queuing at the receiver. The response time to a transmission request for message AM_i is defined as the time lapse from the request instant until complete transmission and it is considered as composed of three parts (9). The parameter σ_i is called *dead interval* and corresponds to the first exclusion period, between the request and the instant in which the message effectively enters arbitration. The parameter w_i, known as *level-i busy window*, allows accounting for exclusions as well as for the interference caused by higher priority messages in the arbitration process until message AM_i starts transmission

$$R_i = \sigma_i + w_i + C_i. \tag{9}$$

An upper bound to the worst case response time for asynchronous message AM_i (Rwc_i^a) is obtained by using upper bounds for both σ_i and w_i (Fig. 8). The first one (σ^{ub}) can be obtained with expression (10) considering that Ca represents the transmission time of the longest asynchronous message. Notice that both the transmission time of AM^{NRT} and the inserted idle time a last for no longer than Ca

$$\sigma^{ub} = 2 * Ca + LSW + LTM. \tag{10}$$

The upper bound for the level-i busy window (w_i^{ub}) is obtained considering that, when message AM_i enters arbitration, it suffers the maximum interference from the synchronous and all higher priority asynchronous messages. It can be determined via an iterative process similar to the one in [23] using a cumulative bus demand function ($H_i(t)$) of the asynchronous traffic with higher priority than AM_i (hp_i) (11)

$$H_i(t) = \sum_{j \in hp_i} \left\lceil \frac{t + \sigma^{ub}}{mit_j} \right\rceil * C_j. \tag{11}$$

However, in this case [32], a bus availability function for the asynchronous traffic ($A(t)$) is also defined (12) to account for

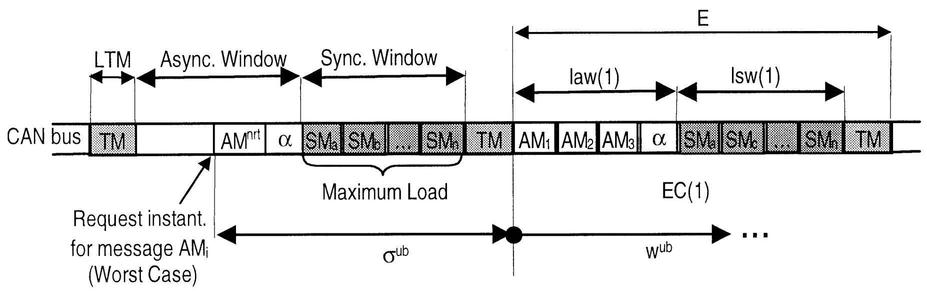

Fig. 8. Maximum *dead interval* (σ_i) and *level-i busy window* (w_i).

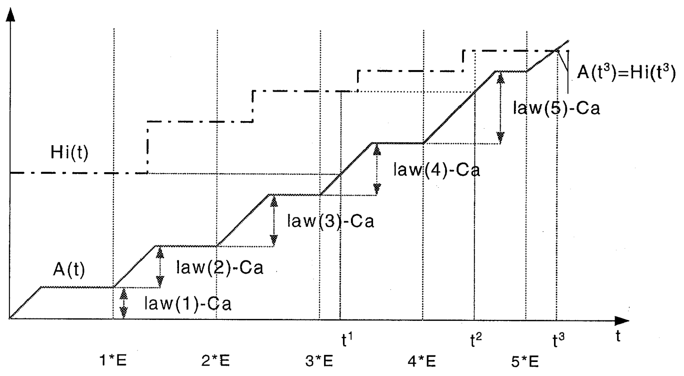

Fig. 9. Calculating the level-i busy window.

both inserted idle time, upper bounded by Ca, and exclusion periods

$$A(t) = \begin{cases} \sum_{j=1}^{n-1}(law(j) - Ca) + (t - (n-1)*E) \\ \quad (n-1)*E < t \leq (n-1)*E + (law(n) - Ca) \\ \sum_{j=1}^{n}(law(j) - Ca) \\ \quad (n-1)*E + (law(n) - Ca) < t \leq n*E \end{cases},$$
$$\text{with } n - 1 = \left\lfloor \frac{t}{E} \right\rfloor. \quad (12)$$

The upper bound for w_i is then obtained as the first instant in time, counted from the start of the arbitration process (end of the *dead interval*), that causes $H_i(t) = A(t)$ (13)

$$w_i^{ub} = t\colon H_i(t) = A(t). \quad (13)$$

This equation can be solved iteratively by using $t^1 = H_i(0)$ and $t^{m+1} = A^{inv}(H_i(t^m))$ (Fig. 9) where $A^{inv}(t)$ stands for the inverse of $A(t)$. The process stops when $t^{m+1} = t^m$ (and $w_i^{ub} = t^{m+1}$) or $t^{m+1} > D_i - C_i - \sigma^{ub}$ (deadline cannot be guaranteed).

An upper bound to the worst case response time for message AM_i (Rwc_i^a) can be obtained through expression (9), replacing w_i by w_i^{ub} obtained from (13), and σ_i by σ^{ub} obtained from (10). Knowing the worst case response time upper bounds for all asynchronous messages, a straightforward sufficient schedulability condition can be derived (C4), consisting of comparing these upper bounds with the respective deadlines

$$Rwc_i^a \leq D_i \qquad \forall_{i=1\cdot Na} \implies \text{ART is schedulable.} \quad (C4)$$

VI. Impact of Errors in FTT-CAN

In real-world distributed systems, the presence of communication errors is inevitable. Hence, it is important to understand how the communication protocol is affected by errors in order to determine the consequent degradation in the quality of its communication services. In what concerns errors, the CAN protocol includes a feature known as automatic retransmission. As soon as an error is detected all nodes transmit error frames in order to resynchronize. Then, the bus is again available to the transmission of messages and the node that was transmitting when the error occurred automatically retransmits the same message. This feature is desirable from the point of view of reliable communication. However, from a timeliness point of view, this feature is not so interesting because the automatic retransmission takes place independently of the temporal validity of the respective message. Furthermore, this feature is also incompatible with a distributed time-triggered approach (e.g., TT-CAN) since it may cause an extension of message transmission times beyond the duration of the respective pre-allocated time slots. Thus, TT-CAN requires the use of *single-shot transmission*, a feature available in most current CAN controllers that corresponds to disabling the automatic retransmission.

On the other hand, FTT-CAN does not require the disabling of the automatic retransmission since the protocol limits its maximum extent to the duration of the window where the error took place. This is achieved implicitly through the same mechanism used to enforce temporal isolation between SMS and AMS, i.e., all transmission activity is suspended at the end of each window. This characteristic of FTT-CAN leads to a desirable error confinement within both subsystems, i.e., any error in SMS does not affect the AMS and vice-versa. Within each subsystem, extra time can be allocated in order to cope with the delays caused by errors as forecasted by an appropriate error model (e.g., [15] and [33]).

Apart from the errors that may occur during the synchronous and asynchronous windows, the EC trigger message is also subject to errors. In this case, the protocol defines a window during which the trigger message can be retransmitted upon error. Since the whole distributed system is synchronized by the reception of the trigger message, any delay that affects this message is carried on to the whole system. When the trigger message is not successfully transmitted up to the end of that window a backup master takes control of the bus and tries to transmit it. This redundancy of masters is required in order to cope with possible master failures. However, the description of the mechanisms used to assure the necessary synchronization between active and backup masters is beyond the scope of this paper.

VII. Summary of Protocol Properties

Summarizing, the previous sections have shown that the FTT-CAN protocol includes an ensemble of features that grant it interesting properties for use in flexible distributed computer control systems. These are as follows.

Temporal Isolation Between Synchronous and Asynchronous Traffic:

- implies a global time-triggered model;
- allows exploiting the advantages of time-triggered communication;
- suitable to distributed applications that involve control, monitoring and management traffic;
- supports error confinement within each type of traffic.

Relative Phase Control for Synchronous Traffic:

- improves traffic schedulability;
- improves jitter control;
- supports composability with respect to the temporal behavior.

Centralized Scheduling for Synchronous Traffic:

- high flexibility in terms of scheduling (any policy can be easily implemented);
- facilitates online admission control (communication requirements are centralized).

Autonomous Communication Control for Synchronous Traffic:

- high robustness with respect to the temporal behavior (enforces specified temporal behavior for the application).

These properties confirm the claims stated in Table I, concerning FTT-CAN. In fact, it should now be clear that, among the fieldbus systems referred to in the table, FTT-CAN is the only one that combines the following features: dynamic planning-based scheduling paradigm, i.e., dynamic communication requirements with guaranteed timeliness; time- and event-triggered traffic with temporal isolation; and bandwidth-efficient handling of the event-triggered traffic.

VIII. Conclusion

This paper discussed the advantages and disadvantages of operational flexibility as well as event and time triggered paradigms in fieldbus communication systems. A brief comparison among several existing fieldbus systems was carried out which stressed the absence of systems capable of combining both paradigms in a flexible and efficient way. This fact led to the development of a new protocol, FTT-CAN, which fulfills those requirements. The distinguishing feature of this protocol is that it supports time-triggered communication in a flexible way as well as an efficient combination of event and time-triggered traffic with temporal isolation, maintaining the desired properties of both types of traffic.

The communication services are delivered by two subsystems, the SMS and the AMS that handle time-triggered and event-triggered communication, respectively. The paper described these subsystems and showed a temporal analysis of both, showing their ability to convey real-time traffic of the respective type.

Finally, a summary of properties has shown that the design goals for flexibility, timeliness, and efficiency have been achieved. Moreover, the protocol is light in terms of both computational and communication overhead. Experimental setups with nodes based on the simple 80C592 8-b microcontroller clocked at 11 MHz have been successfully built, using a transmission rate of 125 kbit/s.

References

[1] L. Almeida, J. Fonseca, and P. Fonseca, "Flexible time-triggered communication on a controller area network," in *Proc. Work-In-Progress Session RTSS'98 (19th IEEE Real-Time Systems Symp.)*, Madrid, Spain, Dec. 1998, pp. 23–26.

[2] L. Almeida, R. Pasadas, and J. A. Fonseca, "Using a planning scheduler to improve flexibility in real-time fieldbus networks," *IFAC Control Eng. Practice*, vol. 7, pp. 101–108, Feb. 1999.

[3] L. Almeida and J. Fonseca, "FTT-CAN: A network-centric approach for CAN-based distributed systems," in *Proc. SICICA 2000, IFAC 4th Symp. Intelligent Components and Instruments for Control Applications*, Buenos Aires, Argentina, Sept. 2000, pp. 279–284.

[4] ——, "Analysis of a simple model for nonpreemptive blocking-free scheduling," in *Proc. ECRTS'01 (EUROMICRO Conf. Real-Time Systems)*, Delft, The Netherlands, June 2001, pp. 233–240.

[5] "CAN application layer for industrial applications," CiA, CAN in Automation Int. Users and Manufacturers Group, Erlangen, Germany, CiA DS 201-207, 1994.

[6] "DeviceNet specification—Release 2.0," ODVA—Open DeviceNet Vendor Assoc., Coral Springs, FL, vol. I and II, 1997.

[7] *General Purpose Fieldbus: Vol. 1: P-Net; Vol. 2: PROFIBUS; Vol. 3: WorldFIP, Amend.1: Foundation Fieldbus'H1*, European Standard EN 50170, 2000.

[8] *Fieldbus Standard for Use in Industrial Control System—Type 1: Existing IEC TS61158 Parts 3–6 (Foundation Fieldbus H1); Type 2: ControlNet; Type 3: PROFIBUS; Type 4:P-Net; Type 5: Fieldbus Foundation HSE; Type 6: SwiftNet; Type 7: WorldFIP; Type 8: Interbus-S*, IEC International Standard 61158, 2000.

[9] *Road Vehicles—Controller Area Network (CAN)—Part 4: Time-Triggered Communication*, ISO/WD11898-4, Dec. 2000.

[10] F. Bogenberger, B. Müller, and T. Füher *et al.*, "Protocol overview," presented at the FlexRay Int. Workshop, Munich, Germany, Apr. 2000.

[11] H. Kopetz, "Should responsive systems be event-triggered or time-triggered?," *IEICE Trans. Inform. Syst.*, vol. E76-D, pp. 1325–1332, 1993.

[12] *TTP/C Protocol, Version 0.5*, TTTech Computertechnik GmbH, Vienna, Austria, 1999.

[13] H. Kopetz, *Real-Time Systems Design Principles for Distributed Embedded Applications*. Norwell, MA: Kluwer, 1997.

[14] E. Martins and J. Fonseca, "Improving flexibility and responsiveness in FTT-CAN with a scheduling coprocessor," in *Proc. FeT'01 (IFAC Conf. Fieldbus Technology)*, Nancy, France, Nov. 2001, pp. 61–67.

[15] N. Navet and Y.-Q. Song, "Design of reliable real-time applications distributed over CAN (Controller Area Network)," in *Proc. INCOM'98 (IFAC Symp. Information Control in Manufacturing)*, Nancy, France, June 1998, pp. 391–396.

[16] P. Pedreiras and L. Almeida, "Combining event-triggered and time-triggered traffic in FTT-CAN: Analysis of the asynchronous messaging system," in *Proc. WFCS 2000, 3rd IEEE Workshop Factory Communication Systems*, Porto, Portugal, Sept. 2000, pp. 67–75.

[17] P. Pedreiras, L. Almeida, and J. Fonseca, "Improving the responsiveness of FTT-CAN synchronous messaging system in FTT-CAN," in *Proc. DCCS 2000, IFAC Workshop Distributed Computer Control Systems*, Sidney, Australia, Nov./Dec. 2000, pp. 110–115.

[18] M. A. Peraldi and J. D. Decotignie, “Combining real-time features of local area networks FIP and CAN,” in *Proc. ICC’95 (2nd Int. CAN Conf.)*, 1995, pp. 8.11–8.21.
[19] P. Raja and G. Noubir, “Static and dynamic polling mechanisms for fieldbus networks,” *ACM Operating Syst. Rev.*, vol. 27, no. 3, pp. 34–45, 1993.
[20] K. Ramamritham and J. A. Stankovic, “Scheduling algorithms and operating system support for real-time systems,” *Proc. IEEE*, vol. 82, pp. 55–67, Jan. 1994.
[21] F. Rossler and B. Geppert, “Applying quality of service architectures to the field-bus domain,” in *Proc. WFCS’97 (IEEE Int. Workshop Factory Communication Systems)*, Barcelona, Spain, Oct. 1997, pp. 39–48.
[22] J. A. Stankovic *et al.*, “Strategic directions in real-time and embedded systems,” *ACM Comput. Surv.*, vol. 28, no. 4, pp. 751–763, 1996.
[23] K. Tindell, A. Burns, and J. Wellings, “Calculating controller area network message response times,” *IFAC Control Eng. Practice*, vol. 3, no. 8, pp. 1163–1168, 1995.
[24] J.-P. Thomesse, “The fieldbuses,” *Annu. Rev. Control*, vol. 22, pp. 35–45, 1998.
[25] J.-P. Thomesse and M. Leon Chavez, “Main paradigms as a basis for current fieldbus concepts,” in *Proc. FeT’99 (Int. Conf. Fieldbus Technology)*, Magdeburg, Germany, Sept. 1999, pp. 2–15.
[26] E. Tovar and F. Vasques, “Cycle time properties of the PROFIBUS timed token protocol,” *Comput. Commun.*, vol. 22, no. 13, pp. 1206–1216, Aug. 1999.
[27] E. Tovar, F. Vasques, and A. Burns, “Supporting real-time distributed computer controlled systems with multi-hop P-net networks,” *IFAC Control Eng. Practice*, vol. 7, no. 8, pp. 1015–1025, Aug. 1999.
[28] M. Mock and E. Nett, “Real-time communication in autonomous robot systems,” in *Proc. ISADS’99 (4th Int. Symp. Autonomous Decentralised Systems)*, Tokyo, Japan, Mar. 1999, pp. 34–41.
[29] M. Natale, “Scheduling the CAN bus with earliest deadline techniques,” in *Proc. RTSS 2000 (21st IEEE Real-Time Systems Symp.)*, Orlando, FL, Dec. 2000, pp. 259–268.
[30] K. Zuberi and K. Shin, “Scheduling messages on controller area network for real-time CIM applications,” *IEEE Trans. Robot. Automat.*, vol. 13, pp. 310–314, Apr. 1997.
[31] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard real-time environment,” *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[32] P. Pedreiras and L. Almeida, “Asynchronous communication in FTT-CAN: Experimental results,” in *Proc. FeT’01 (IFAC Conf. Fieldbus Technology)*, Nancy, France, Nov. 2001, pp. 113–120.
[33] H. Hanson, C. Norstrom, and S. Punnekkat, “Integrating reliability and timing analysis of CAN-based systems,” in *Proc. WFCS 2000, 3rd IEEE Workshop Factory Communication Systems*, Porto, Portugal, Sept. 2000, pp. 165–172.

Luís Almeida (S’86–M’89) received the degree in electronics and telecommunications engineering and the Ph.D. degree in electrical engineering from the University of Aveiro, Aveiro, Portugal, in 1988 and 1999, respectively.

He has been an Assistant Professor in the Department of Electronics and Telecommunications, University of Aveiro, since 1999. He is also a Senior Researcher with the IEETA research unit at the university. Formerly, he was a Design Engineer with a company producing digital telecommunications equipment. His research interests lie in the fields of real-time networks for distributed industrial/embedded systems and navigation control for mobile robots. He is a coauthor of a national patent concerning a fieldbus communication system, and he regularly participates in scientific events in both of his fields of interest.

Paulo Pedreiras received the degree in electronics and telecommunications engineering in 1997 from the University of Aveiro, Aveiro, Portugal, where he is currently working toward the Ph.D. degree.

Formerly, he was a Design Engineer in an industrial company, engaged in the development of test and measurement equipment for the automotive industry. His current research interests include real-time communication systems, scheduling, fieldbuses, and real-time operating systems.

José Alberto G. Fonseca (M’00) was born in Aveiro, Portugal, in 1957. He graduated in electronics and telecommunications engineering and received the Ph.D. degree in electrical engineering from the University of Aveiro, Aveiro, Portugal, in 1980 and 1992, respectively.

He has been an Associate Professor in the Electronics and Telecommunications Department of the University of Aveiro, Aveiro, Portugal, since 2000. His current research interests are embedded systems, distributed systems, and industrial communications. Presently, he is the Co-ordinator of two research projects funded by the Portuguese government in the fields of distributed embedded systems and environment monitoring systems. Since 1992, he has authored or coauthored more than 40 technical papers and has recently submitted two patents for fieldbus-based systems.