Guaranteeing Fault Tolerance through Scheduling on a CAN bus

M. P. Oliveira, A. O. Fernandes, S. V. A. Campos, A. L. A. P. Zuquim

Departamento de Ciência da Computação, Universidade Federal de Minas Gerais

Caixa Postal 702 - 30123-970 - Belo Horizonte - MG - Brasil
{marcospo, otavio, scampos, ana@dcc.ufmg.br}

Abstract. *Prioritizing tasks in Hard-Real-Time Systems is a problem belonging to NP-hard class. Scheduling and resource allocation in real-time systems are difficult problems due to the timing constraints of the tasks involved. Scheduling policies in hard real-time systems need to ensure that tasks will meet their deadlines under all circumstances, even in the presence of faults.*

This work presents techniques to enhance the fault tolerance capability of multiprocessor hard real-time systems, in the presence of transient and permanent faults. As a special case, in the present paper we propose a new method to obtain a high level of fault-tolerance in the CAN bus by incorporating time redundancy and task schedulability tests, which may be used concurrently with processor redundancy and any other hardware redundancy.

1 Introduction

Real-time systems are systems that depend on the result of computation as well on the deadline by which this result is reached. Real-time systems include sensors and actuators; task deadlines are typically derived from the required responsiveness of the sensors and actuators, which are monitored and controlled by the system. Examples of such systems include signal processing, process control, flight control, telecommunication, automotive and life-support medical systems. Hard real-time systems (HRTSs) have stringent timing constraints, and the consequence of missing task deadlines may be catastrophic.

Scheduling and resource allocation in real-time systems are difficult problems due to the timing constraints of the tasks involved. Fault tolerance is an especially vital requirement for HRTS development. Scheduling policies in hard real-time systems need to ensure that tasks will meet their deadlines under all circumstances, even in the presence of transient or permanent faults. The time requirements and fault model depend on a high knowledge of the application and its environment.

This work presents an overview, extension and application of techniques to enhance the fault tolerance capability of multiprocessor hard real-time systems based on the CAN protocol. We introduce extensions to the CAN (Controller Area Network) bus communication protocol and apply a set of techniques to improve its reliability. A well-known problem of the CAN bus is on delivering low priority messages, which may be compromised if the bus is flooded with higher priority messages. In this work, we use task schedulability and time redundancy to optimize fault-tolerance requisites for multiprocessor hard real-time systems. In this scenario, a new technique is proposed to enhance the fault-tolerance capability of the CAN protocol by incorporating time redundancy, which can be used in conjunction with hardware and software redundancy to tolerate faults in hard real-time systems.

This work was developed over the CAN protocol mainly because of it is present in more than 90% of microcontrollers and DSPs (digital signal processors) incorporating real-time protocols. A comparison of real-time fault-tolerant communication protocols is also presented, highlighting the advantages of the proposed method.

This paper is organized in the following form. In Section 2 related work including real-time protocols with fault tolerance requisites are presented. In Section 3 the new approach to improve reliability in the CAN bus is shown. Finally the results are briefly analyzed and some conclusions are presented.

2 Related Work

One of the essential services provided by real-time fault-tolerant distributed architecture is communication of information from one distributed component to another; a communication bus is one of its principal components, and the protocols used for control and communication on the bus are among its principal mechanisms. In truth, these architectures are the safety-critical core of the applications built above them, and the choice of services to provide to those applications, and the mechanisms of their implementation, are issues of major importance in the construction and certification of safety critical embedded systems [12][13]. In a distributed hard real-time system, communication between tasks on different processors must occur in bounded time.

Redundant busses are often used in safety-critical environments to handle device faults. Besides fault tolerance, many applications require real-time guarantees such as deterministic message latency. There are various protocols for such purposes, with different complexities, which are used by the avionics industry, such as Airbus and Boeing, and automobile industry, such BMW and Audi. Some of the more representative real-time communication protocols today are TTP/C (Time-Triggered Protocol) [17], FlexRay [6], CAN (Controller Area Network) [4] and TTCAN (Time-Triggered CAN) [1][3][11].

Some of the busses considered here are primarily *time triggered* which means that all activities involving the bus, and often those involving components attached to the bus, are driven by the passage of time. In *event-triggered* busses, the activities are driven by the occurrence of events. A time-triggered system interacts with the world according to an internal schedule, whereas an event-triggered system responds to stimuli.

The Time Triggered Architecture (TTA) was developed by Hermann Kopetz and colleagues at the Technical University of Vienna [7]. Commercial development of the architecture is being deployed for safety-critical applications in cars and for flight-critical functions in aircrafts. In a Time-Triggered Architecture, the communication system decides autonomously and according to a static schedule when to transmit a message.

FlexRay [6] is a new real-time protocol, not yet released to the public, being developed by a consortium of companies (BMW, Motorola, etc.) aiming to be more flexible than TTP/C. Although used primarily for automotive applications, it is representative of state-of-the-art safety critical real-time protocols. FlexRay introduces some dynamism through the combination of time-triggered and event-triggered operation.

The controller area network (CAN) protocol [4] uses a serial multimaster communication where prioritized messages with up to eight bytes data length can be sent using an arbitration protocol and an error-detection mechanism for a high level of data integrity. The CAN bus is ideal for applications operating in noisy and harsh environments, such as in the automotive and other industrial fields that require reliable communication.

A new development in CAN technology is the TTCAN protocol [1][3][11], a higher-layer protocol above the unchanged standard CAN protocol that synchronizes the communication schedules of all CAN nodes in a network and that provides a global system time, avoiding the transmission collisions commonly found in standard CAN networks. In TTCAN, all the message instances are transmitted only on previously allocated time-slots, just like the TTP/C protocol, without competing with other messages for the bus.

2.1 Rate Monotonic Scheduling applied to the CAN bus

The dynamic scheduling algorithm used by the CAN protocol is virtually identical to scheduling algorithms commonly used in real-time systems to schedule computation on processors [14]. In fact, the analysis of the timing behavior of such systems can be applied almost without change to the problem of determining the worst-case latency of a given message queued for transmission on CAN.

Tindell *et. al.* [15] developed a CAN analysis based on RMA, showing how to find the response time for messages being transmitted in a CAN bus.

As defined by Tindell, the worst-case response time is composed of two delays: the queuing delay and the transmission delay. The queuing delay is the longest time that a message can be queued in a station and be delayed because other higher and lower priority messages are being sent on the bus, which are known as interference and blocking time respectively. The transmission delay is the time taken to actually send the message on the bus.

A model for error handling must also be included, once it is important in a fault tolerant scenario. In a CAN bus, an error detected by either the sender of a message or a receiver station is signaled to the sender station, which must re-transmit that message. The costs of error handling are given as the most probable bound on the overheads due to errors in an interval of duration t, and it includes the cost of re-transmission. An extended analysis can also be found in [15] which includes also remote transmission request messages.

3 Improving Reliability in a CAN bus

A perceived problem with the CAN protocol for use in distributed real-time control applications is the inability to bind the response times of messages. While CAN is very good at transmitting the most urgent data, it is unable to provide guarantees that deadlines are met for less urgent data [7] [8] [9], once the most urgent data may flood the bus avoiding the transmission of the less urgent data.

In a CAN bus, the delivery of low priority messages may be compromised if the bus is flooded with higher priority messages. In this sense, we need to guarantee that an overload in the bus will not occur and a deterministic package response time can be achieved.

3.1 Applying Fault tolerance requisites to a CAN bus

Due to the critical nature of the tasks in hard real-time systems, it is essential that faults be tolerated. Transient faults in real-time systems are generally tolerated using time redundancy, which involves the re-execution of any task running during the occurrence of a transient fault [5].

Ghosh [2] showed a recovery scheme for single and multiple faults that ensures the re-execution of any task after a fault has been detected. Once the dynamic scheduling algorithm used by the CAN protocol is virtually identical to scheduling algorithms commonly used in real-time systems to schedule tasks, this recovery scheme may be also used to ensure the re-transmission of any message in a CAN bus

The general approach to fault tolerance is to maintain enough slack (backup time) in the schedule so that any message instance can be re-transmitted if a fault occurs during its transmission. If no faults occur, messages are transmitted just following the usual RMS scheme and the slack is not used. If a fault occurs in the transmission process of a message, a recovery scheme is used to re-transmit that message. The ratio of slack S available over an interval of time L is thus constant and can be imagined to be the utilization of a backup message B. If the backup utilization is U_B, then the slack available during an interval L, denoted by B_L, is $B_L=U_BL$.

In order to apply this approach the following conditions must be satisfied:

[Sl]: There should be sufficient slack for every instance of each message to be re-transmitted. That is, the slack between kT_i and $(k + 1) T_i$ should be at least C_i for any value of k and i, what ensures the availability of sufficient slack for a message to be re-transmitted.

[S2]: When any instance of τ_i finishes executing, all the slack available within its period (at least C_i if [Sl] holds) should be available for the re-transmission of τ_i. This slack can be used after a message finishes transmitting to re-transmit that message before its deadline, if a fault is detected.

[S3]: When a message re-transmits, it should not cause any other message transmission to miss its deadline, allowing all tasks to meet their deadlines even when a high priority task needs to re-execute.

If these three conditions are met, then it is possible to re-transmit a faulty message and meet its deadline. However, a recovery scheme must define also how the slack should be used and a very straightforward scheme consists on the faulty message simply being re-transmitted at its own priority.

This scheme is a general approach to distribute slack in the schedule, and it can be applied to any non-fault-tolerant scheduling scheme for preemptive, periodic tasks where the RMS assumptions hold. Any transmission time C_i in the non-fault-tolerant scheme can be split into two parts for the fault-tolerant scheme: a new transmission time $C_i' = C_i(l - U_B)$ (where U_B is the backup utilization) and a slack equal to C_iU_B. To guarantee the re-transmission of a message before its deadline, its *critical instance* is considered, which is defined as the time at which the message's transmission is maximized - it happens when the message starts its transmission process simultaneously with all higher priority messages [10]. The total slack available for any message at its critical instance is equal to the total slack available within a period boundary, which is defined as the beginning of a period.

Ghosh showed also that, by splitting up each transmission time C_i into a new transmission time C_i' and slack, as described above, the utilization of each task τ_i is reduced to $U_i(l- U_B)$, and thus the following general fault tolerance boundary for an RMS ($U_{G\text{-}FT\text{-}RMS}$) is obtained:

$$U_{G\text{-}FT\text{-}RMS} = n(2^{1/n} - 1)(1 - U_B) = U_{LL\text{-}RMS}(l- U_B) \qquad (1)$$

The above equation is a general one applicable to an RMS for any value of U_B. If $U_B=\max\{Ui\}$, $i = 1,..., n$, then any message transmission in the system can tolerate a single fault. Any number of faults can be tolerated if [S1] holds.

Multiple faults within two consecutive period boundaries are also guaranteed to be tolerated using the scheme described above. If several backups are provided in the system, and the total backup utilization is U_{BT}, then a general boundary for the message set can be derived by replacing U_B with U_{BT} in $U_{G\text{-}FT\text{-}RMS}$; that is, the new boundary is $U_{LL\text{-}RMS}(l- U_{BT})$.

3.2 Limiting the maximum transfer rate of the CAN bus

Considering the CAN protocol, the absence of a message is identified by the CPUs connected to the bus as a fault. In this case, a failure model is applied, implying in the re-transmission of the message that failed in its transmission or even in the execution of an alternative action such as the reconfiguration of the bus.

The RMS scheme can be applied to the CAN bus since the following premises can be assumed:

- The messages are independent, which means in other words that they are asynchronous to each other.
- The messages will have their priority ascertained by RMA.
- Each message has a maximum transmission time and deterministic period.

The independency of the messages is guaranteed by the fact that we are dealing with periodic control messages. We must also guarantee that [S1], [S2] and [S3] are satisfied, as presented in Section 3.1. The maximum transmission time is, in the worst-case, the worst-case transmission time defined by Tindell *ET. al* in [14], and will help us define the backup slack size and ensure the availability of sufficient slack for a message to be re-transmitted. This slack can be used after a message finishes transmitting to re-transmit that message before its deadline, if a fault is detected. This means that [S1] and [S2] hold for the CAN protocol.

The third condition [S3] may not hold for a CAN bus, once a higher priority message re-transmission may prevent the transmission of a lower priority message, causing the latest to miss its deadline. Thus, to guarantee that a CAN bus can tolerate faults, we define a maximum transmission rate for each message instance; instead of pre-defining a time-driven slot as is the main idea of TTP/C. With this idea in mind, we will not limit a node transmission to its slot, but allow it to transmit at any time if its transmission frequency allows. We will call this extension of the CAN bus as RMCAN, which means *Rate Monotonic CAN.*

From the protocol point of view, the retransmission of a finite number of messages in the case of a transmission failure may not compromise the bus bandwidth and the RMS may be applied deterministically.

This way, any message corruption or further errors indicate that a fault occurred, and the message must be re-transmitted and also meet its deadline. Moreover, if the fault persists, a failure is detected and an alternative process must be executed in another processor to prevent a global failure.

The following example shows how RMCAN can be used to determine whether a message can be re-transmitted before its deadline with guarantees in a CAN bus. Consider a set of 10 processors - N = 10 - sending 5 periodic messages with utilizations U_i = 1% for each message i. Each processor is also limited to a maximum data transmission volume of 10% of the bus bandwidth. Once this limit is reached, the processor that is sending messages stops sending messages and higher layers of the processor that should be receiving the messages will tolerate the fault taking the appropriate action in the context of the specific application.

If we assume that a message fault needs to be tolerated and re-transmitted up to 5 times, then U_B = 5%. Equation (2) gives us a bound of 66% while the sum of utilizations of the task is 50%.

$$100\,(2^{1/100} - 1) * (1 - 0{,}05) = 0{,}66 \qquad (2)$$

Since $\Sigma U_i < U_{G\text{-}FT\text{-}RMS}$, the messages are schedulable.

4 Comparison of Real-Time Fault-Tolerant Communication Protocols – advantages of the proposed method

This comparison does not reiterate the common design decisions, but focuses on the differences between TTP/C, CAN, TTCAN and FlexRay protocols and shows the advantages of the adaptation proposed over the CAN protocol in this work (RMCAN).

From the buses considered just above, only TTP/C is solely time-triggered while the CAN bus is event-triggered. TTCAN and FlexRay combine time-triggered and event-triggered operation aiming to be more flexible than TTP/C and safer than CAN protocol. This time-triggered versus event-triggered decision is a fundamental design choice that influences many aspects of their architectures and mechanisms. The mechanism adopted by each protocol to resolve transmission concurrency between nodes is decisive to indicate if collisions or concurrency occur during runtime.

Analyzing the performance of these protocols, we may see that latency is constant and known at design time for TTP/C, while it may increase with load in a CAN bus. The main problem with the CAN bus is that it cannot prevent an overload of the communication system, which may cause a disastrous result when the delivery of low priority messages is prejudiced by higher priority messages re-transmission. RMCAN solves this problem by limiting the transmission frequency of a node to a maximum value. In this case, the worst-case and latency are precisely known.

In this sense, little is known about the FlexRay protocol, which has not been released yet. All that is known is that FlexRay provides no services to its applications beyond best efforts message delivery. A never give up strategy inside FlexRay leaves the control of the communication system with the application, and latency will be constant and precisely known at design time for the TDMA window.

Resuming, we may say that TTP/C provides an off-line communication design yielding guaranteed latency for all messages in the system, but presents low flexibility once bandwidth is distributed at design time by assigning frames of specific length to each node. In a CAN bus, otherwise, priorities are distributed at design time by assigning unique identifiers and a full control by the application over the bandwidth distribution. The CAN protocol is highly flexible and widely available, although some extensions must be done to guarantee a reliable mechanism to build fault-tolerant safe-critical systems. TTCAN is a compromise and represents the necessary evolution of CAN for dealing with higher loads on the bus. However, synchronizing nodes is not a simple task, and brings a new complexity to the CAN bus. RMCAN was developed to be as efficient as TTCAN without incorporating extra hardware or difficulties. Both protocols can be implemented using a regular CAN microcontroller, although for TTCAN it is also necessary an extra hardware for the time-triggered portion of the protocol. Synchronizing nodes is not a simple task, and brings a new complexity to the CAN bus in TTCAN. Implementing RMCAN is much easier, and does not require any extra hardware. The transmission frequency of each message set can be controlled through software.

FlexRay can be considered the state-of-art in the real-time fault-tolerant communication protocols area, although it has not been released yet. It promises a higher bit rate than TTCAN an increase in flexibility when compared to TTP/C.

The use of the CAN protocol in the development of applications is favored by the high availability of microcontrollers incorporating the bus. Today, the biggest advantages of CAN compared to other networks are the costs and the price/performance ratio. The enhancements proposed by TTCAN and RMCAN are examples of how CAN problems can be circumvented and its spread presence in the market can be explored.

	TTP/C	CAN	TTCAN	RMCAN	FlexRay
Media access strategy	Time-triggered	Event-triggered	Time and Event-triggered	Event-triggered and transmission frequency directed	Time and Event-triggered
Dynamic bandwidth sharing among nodes	No	Yes	Yes	Yes	Yes
Market presence	< 1%	> 99%	May explore the CAN protocol market presence		Not available
Data Efficiency vs. Latency	Constant and known at design time	Increases with load	Constant	Constant	Constant and known at design time
Response Time	Deterministic	Non-deterministic	Deterministic	Deterministic	Deterministic

Table 1 – Comparison between TTP/C, CAN, TTCAN, RMCAN and FlexRay protocols

5 Conclusions

Tasks in real-time systems must meet their deadlines under all circumstances, even in the presence of transient or permanent faults. This work has shown that time redundancy through scheduling is a powerful tool to deal with faults in real-time systems. The harmonious integration of the available techniques enhance the fault tolerance capability of multiprocessor hard real-time systems.

Relating to real-time communication protocols, the time-triggered and event-triggered approaches find favor in different application areas, and each has strong advocates [13]. The CAN protocol may have a non-deterministic response time for an arbitrary low priority message. Researchers sometimes say that the CAN protocol is more appropriate for soft real-time systems (flexible requirements), while appropriate protocols for hard real-time systems include TTP/C. RMCAN, the extension proposed to the CAN protocol, shows that it is possible to bound the message transmission time, thus making possible its use on HRTSs.

In RMCAN there is a limit on the node transmission rate, making the transmission time deterministic, even for low priority messages. We do not limit a node transmission to its slot, but allow it to transmit at any time if its transmission frequency allows. This way one can guarantee that a message will arrive at its deadline or it will not arrive anymore, in which case a backup action is taken.

One advantage of the CAN protocol over time-triggered protocols is the extensibility aspect. New nodes can be added to the bus, while in the TTP/C protocol, for example, a slot for a new node has to be reserved at design time. Other advantages are the high availability of microcontrollers incorporating the CAN bus, and the price/performance ratio. The enhancements proposed to the CAN protocol show that its reliability can be increased and its spread presence in the market can be further explored.

Another aspect in multiprocessor real-time systems relates to the interdependence of task execution time and message transmission time. The release jitter of a receiver task depends on the arrival time of a message, which in turn depends on the interference from higher priority messages, which in turn depends on the release jitter of the sender tasks. A future work may be the analysis of the relationship of these times, considering worst-case situations.

6 References

[1] Fuhrer T., Muller B., Dieterle W., Hatwitch F., Hugel R., Weiler H., Walther M., GmbH R. B.; Time Triggered Communication on CAN; Proceedings 7th International CAN Conference; 2000.
[2] Ghosh Sunondo, "Guaranteeing Fault Tolerance Throgh Scheduling in Real-Time Systems", Ph.D. Thesis, University of Pittsburgh 1996.
[3] Hartwich F., Futhrer T., Hugel R., Muller B., GmbH R. B.; Timing in the TTCAN Network; Proceeding 8th Intenational CAN Conference; 2002, Las Vegas.
[4] ISO 11898:1993 Road vehicles -- Interchange of digital information -- Controller area network (CAN) for high-speed communication.
[5] Kopetz H., Kantz H., Grunsteidl G., Puschener P., Reisinger J.; Tolerating Transient Faults in MARS. In Symp. On Fault Tolerant Computing(FTCS-20), pages 466-473. IEEE, 1990.
[6] Kopetz H., A Comparation of TTP/C and FlexRay. Research Report. Institut fur Tevhnische Informatik. Technische Universitat Wien, Austria. 2001.
[7] Kopetz, H., Griinsteidl, G., TTP- A Protocol for Fault-Tolerant Real-Time Systems, IEEE Computer, January 1994, pp. 14-23
[8] Kopetz H.. Communication Protocols for Fault-Tolerant Distributed Real-Time Systems. 1994. Nortic Seminar on Dependable Computing, Technical University of Denmark, Lyngby, Denmark.
[9] Kopetz, H., *"A Solution to an Automotive Control System Benchmark",* Institut fur Technische Informatik, Technische Universitat Wien, research report 4/1994 (April 1994)
[10]Liu C.L., Layland J.W., " Scheduling Algorithms for multiprogramming in a Hard-Real-Time Environment," J.ACM, vol. 20, pp.46-61, 1973.
[11]Muller B., Fuhrer T., Hatwitch F., Hugel R., Weiler H., GmbH R. B.; Fault Tolerant TTCAN Networks; Proceedings 8th International CAN Conference; 2002; Las Vegas.
[12]Rushby J., Bus Architectures for Safety-Critical Embedded Systems. SRI International Computer Science Laboratory. 2001 Menlo Park USA.
[13]Rushby J., A Comparison of Bus Architectures for Safety-Critical Embedded Systems. SRI International Computer Science Laboratory. CSL Technical Report. 2001 Menlo Park USA.
[14]Tindell K., Burns A., Guaranteeing Message Latencies on Control Area Network (CAN). University of York, Department of Computre Science, York, England, 1994.
[15]Tindell K., Burns A., Wellings A.,. Calculating Controller Area Network (CAN) Message Response Times. University of York, Department of Computre Science, York, England, 1994.
[16]Tindell K., Fixed Priority Scheduling of hard real-time systems. Ph. D. Thesis. University of York, Department of Computer Science, York, England. 1994.
[17]TTTech Computertechnik, TTP/C Protocol. Specification of TTP/C Protocol. http://www.tttech.com AG 1999.