Fault Tolerant Scheduling on Controller Area Network (CAN)

Hüseyin Aysan, Radu Dobrin, and Sasikumar Punnekkat
Mälardalen Real-Time Research Centre, Mälardalen University
Västerås, Sweden
{*huseyin.aysan, radu.dobrin, sasikumar.punnekkat*}*@mdh.se*

***Abstract*—Dependable communications is becoming a critical factor due to the pervasive usage of networked embedded systems that increasingly interact with human lives in one way or the other in many real-time applications. Though many smaller systems are providing dependable services employing uniprocesssor solutions, stringent fault containment strategies etc., these practices are fast becoming inadequate due to the prominence of COTS in hardware and component based development (CBD) in software as well as the increased focus on building 'system of systems'. Hence the repertoire of design paradigms, methods and tools available to the developers of distributed real-time systems needs to be enhanced in multiple directions and dimensions.**

In future scenarios, potentially a network needs to cater to messages of multiple criticality levels (and hence varied redundancy requirements) and scheduling them in a fault-tolerant manner becomes an important research issue. We address this problem in the context of Controller Area Network (CAN), which is widely used in automotive and automation domains, and describe a methodology which enables the provision of appropriate scheduling guarantees. The proposed approach involves definition of fault-tolerant windows of execution for critical messages and the derivation of message priorities based on earliest deadline first (EDF).

***Keywords*-fault tolerance; time redundancy; real-time systems**

I. Introduction

Embedded systems are deployed ubiquitously in applications that interact and control our lives including in safety critical applications. These systems are increasingly interacting with each other and providing dependable communications is becoming a important research question. On the other hand, due to cost and time to market pressures, industry is turning towards commercial-off-the-shelf (COTS) solutions and component based software development (CBD). Though these approaches provide numerous novel possibilities in the product development and in the provision of enhanced services, they also bring-in hitherto unknown complex interactions between the individual components. Traditional approaches such as uniprocesssor solutions, clean separation between critical and non-critical tasks, stringent fault containment strategies, etc., are thus becoming inadequate at a fast pace and the system designers are in need of new methodologies that guarantee dependable behaviours of systems. In this paper, we address how to schedule a set of messages of mixed criticality in a fault-tolerant manner and ensure dependable communications in the context of Controller Area networks (CAN).

CAN is quite popular in the automotive and automation industries due to its ease in use, low cost and provided reduction in wiring complexity. CAN was designed in the 1980s at Robert Bosch GmbH [1] with a special focus on automotive real-time requirements. The most important feature of CAN from the real-time perspective is its predictable behaviour. CAN provides means for prioritized control of the transmission medium by using an arbitration mechanism which guarantees that the highest priority message that enters an arbitration will be transmitted first. This makes CAN amenable to response time analysis akin to those performed on fixed priority task sets. Volcano methodology at Volvo [2] is an example of the acceptance of such analysis by the industry.

The model underlying the basic CAN analysis assumes an error free communication bus, i.e. all messages sent are assumed to be correctly received, which may not always be true. For instance, in applications such as automobiles, the systems are often subjected to high degrees of Electro Magnetic Interference (EMI) from the operational environment which can potentially cause transmission errors. The common causes for such interference include cellular phones and other radio equipments inside the vehicle and electrical devices like switches and relays as well as as radars, radio transmissions from external sources and lightning in the environment. Electro Magnetic Compatibility (EMC) has been seriously considered by the automotive industry for more than 40 years, and several legislations and directives are in effect to tackle the interference problem [3]. However, even today it has not been possible to completely eliminate the effects of EMI since exact characterisation of all such interfences defy comprehension. Though usage of an all-optical network could greatly eliminate EMI problems, it is not favoured by the cost-conscious automotive industry.

These interferences cause errors in the transmitted data, which could indirectly lead to catastrophic results. To reduce the risk due to erroneous transmissions, CAN designers have provided elaborate error checking and error confinement features in the protocol. Basic philosophy of these features is to identify an error as fast as possible and then retransmit the affected message. This implies that in systems without spatial redundancy of communication medium/controllers, the fault-tolerance (FT) mechanism employed is time redundancy which could have an adverse impact on the latencies of message sets; potentially leading to violation of timing requirements. Moreover, mixed criticality messages imply different reliability requirements, e.g., non critical messages do not need to be retransmitted at all while critical ones require a specified level of fault-tolerance. Hence, the native message retransmission mechanism, that assumes that all messages are equally critical, can not handle the above scenarios and needs to be replaced. This is typically done by disabling the retransmission feature [4] and giving the responsibility to the local schedulers/nodes.

In this paper we propose a framework to provide selective fault-tolerance for messages with various fault-tolerance requirements scheduled on CAN. We analyze off-line the set of messages and provide scheduling attributes that ensures feasible transmission of messages as well as retransmissions upon error occurrences, that satisfy the fault-tolerance requirements.

II. Controller Area Network (CAN)

CAN is a broadcast bus designed to operate at speeds of up to 1 Mbps. Data is transmitted in messages containing between 0 and 8 bytes of data. An 11 bit identifier is associated with each message. There is also an extended CAN format with a 29 bit identifier, but since this format is identical in all other respects, it will not be considered here. The identifier is required to be unique, in the sense that two simultaneously active messages originating from different sources must have distinct identifiers. The identifier serves two purposes: (1) assigning a priority to the message, and (2) enabling receivers to filter messages. A station filters messages by only receiving messages with particular bit patterns. The use of the identifier as priority is the most important part of CAN with respect to real-time performance.

CAN is a collision-detect broadcast bus, which uses deterministic collision resolution to control access to the bus. The basis for the access mechanism is the electrical characteristics of CAN bus: if multiple stations are transmitting concurrently and one station transmits a '0' then all stations monitoring the bus will see a '0'. Conversely, only if all stations transmit a '1' will all processors monitoring the bus see a '1'. This behaviour is used to resolve collisions: each station waits until the bus is idle. When silence is detected, each station begins to transmit the highest priority message held in its output queue whilst monitoring the bus. The identifier is the first part of the message to be transmitted; the identifier is transmitted from the most-significant to the least-significant bit. If a station transmits a recessive bit ('1'), but monitors the bus and sees a dominant bit ('0'), then it stops transmitting since it knows that its message is not the highest priority message currently being transmitted in the system. Because identifiers are deemed unique within the system, a station transmitting the last bit of the identifier without detecting a collision must be transmitting the highest priority queued message, and hence can start transmitting the body of the message.

The CAN message format contains 47 bits of protocol control information (the identifier, CRC data, acknowledgement and synchronisation bits, etc.). The data transmission uses a bit stuffing protocol which inserts a stuff bit after five consecutive bits of the same value. The frame format is specified such that only 34 of the 47 control bits are subject to bit stuffing. Hence, the maximum number of stuff bits in a message m_i with n bytes of data is $\lfloor\frac{(n*8+34-1)}{4}\rfloor$ (since the worst case bit pattern is '1111100001111...'). This means that a message is transmitted with between 0 and 24 stuff bits. Hence, the size of a transmitted CAN message is in the range 47..135 bits. The transmission time, denoted C_i, of message M_i is given by the number of bits to be transmitted for the message multiplied by the time required to transmit one bit, denoted τ_{bit}. Hence, for a message with n bytes of data, the transmission time is:

$$C_i = (n*8+47+\lfloor\frac{(n*8+34-1)}{4}\rfloor)*\tau_{bit} \qquad (1)$$

A. Response Time Analysis of CAN

Tindell et. al [5] present analysis to calculate the worst-case latencies of CAN messages. This analysis is based on the standard fixed priority response time analysis for CPU scheduling [6], and later refined by Davis et. al. [7]. Calculating the response times requires a bounded worst case queuing pattern of messages. The standard way of expressing this is to assume a set of traffic streams, each generating messages with a fixed priority. The worst case behaviour of each stream is to periodically queue messages. In analogue with CPU scheduling, we obtain a model with a set $\mathcal{S}$ of messages (corresponding to CPU tasks). Each message $M_i \in \mathcal{S}$ has a priority P_i (defined by the message identifier), period T_i and worst case transmission time C_i of the message M_i.

For an ideal CAN controller (the non-ideal case is discussed in [8]) the worst-case latency R_i of a CAN message M_i is defined by

$$R_i = J_i + q_i + C_i \qquad (2)$$

where J_i is the queuing jitter of message M_i, inherited from the sender task which queues the message. We have assumed the minimum delay from the point in time p, relative which the response time is measured, to the time message M_i is queued to be 0 (p is typically the start of the period). In other cases we need to add a term $J_i^{smallest}$ to equation (1), since jitter is defined as the difference between the biggest and smallest delay from p. The worst-case queuing delay q_i

is given by,

$$q_i = max(B_i, C_i) + \sum_{j \in hp(i)} \left\lceil \frac{q_i + J_j + \tau_{bit}}{T_j} \right\rceil C_j \qquad (3)$$

where the term B_i is the worst-case blocking time of the longest possible message frame (i.e., the worst-case transmission time of a CAN message frame with 8 bytes of data and worst-case bit stuffing), $hp(i)$ is the set of messages with priority higher than M_i, J_j is the queuing jitter of message M_j and τ_{bit} caters for the difference in arbitration start times at the different nodes due to propagation delays and protocol tolerances. The reason for the blocking factor is that transmissions are non-preemptive, i.e., after a bus arbitration has started the message with highest priority among competing messages will be transmitted, even if a message with higher priority is queued before the transmission is completed. This means that, in the worst case, a message may have to wait for the transmission of at most an entire low priority message frame. Hence, B_i is defined by:

$$B_i = \max_{\forall k \in lp(i)} (C_k) \qquad (4)$$

where $lp(i)$ is the set of messages with priority lower than message M_i. Note that the lowest priority message has blocking factor zero, just as in the case of task scheduling. (But be aware! If there is background traffic with lower priority than the considered real-time message messages, the maximum background message size should also contribute to B_i.)

Punnekkat et al [9] extended the above analysis and presented an approach to schedule messages in a fault-tolerant manner using fixed priority scheduling (FPS). Broster [10] addressed the reliability of message transmission on CAN assuming probabilistic fault models. Bartolini et. al. [11] presented an approach to reduce the response time of multi-frame messages in CAN by using the Priority Inheritance Protocol.

B. Error Handling features in CAN

In CAN, errors may occur due to different sampling points or switching thresholds in different nodes or due to signal dispersion during propagation. To handle these, the CAN protocol provides elaborate error detection and self-checking mechanisms [12], specified in the data link layer of ISO 11898 [13]. The error detection is achieved by means of transmitter-based-monitoring, bit stuffing, Cyclic Redundancy Check and message frame check.

To make sure that all nodes have a consistent view, errors detected in one node must be globalized. This is achieved by letting the detecting node transmit an error flag containing 6 bits of same polarity. Upon reception of an error frame, each node will discard the erroneous message, which then will be automatically re-transmitted by the sender. Note that, the re-transmitted message could be subjected to arbitration during re-transmission. This implies that if any higher priority messages gets queued during the transmission and error signaling of the current message, then those messages will be transmitted before the erroneous message is re-transmitted. In our proposed approach we assume single-shot transmission, i.e., the automatic retransmission mechanism is disabled. This may require the use of controllers that has this particular feature built in, e.g., Atmel T89C51CCO2, Philips SJA1000 or Microchip MCP2515.

Specification documents of CAN [14] claim that the error detection mechanisms can detect and globalize all transmitter errors. Bursts are guaranteed to be detected on the receiver side up to a length of 15 (which is equal to the degree of f(x) in CRC sequence). Most longer error bursts are also detected. Even though there is a positive probability for undetected errors, we shall assume that all errors are detected. The probability for undetected errors is negligibly small, as indicated by the following quote from the CAN technical information [14]: “with an operating time of eight hours per day on 365 days per year and an error rate of 0.7 s, one undetected error occurs every thousand years (statistical average)”.

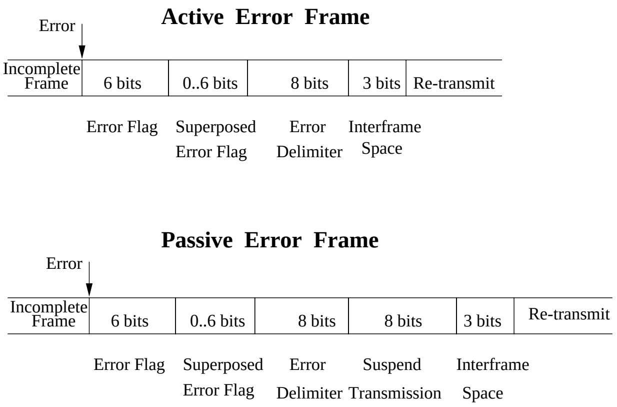

Figure 1. Error Frame Formats in CAN

Figure 1, shows formats of the CAN error frames (details are given in [14]). It can be seen that error signaling and recovery time is typically between 17 to 31 bit times. Since we are interested in the worst case behaviour, we shall use 31 bit times as the error signalling and recovery time in our model.

III. Real-time system model

We assume a distributed real-time architecture consisting of sensors, actuators and processing nodes communicating over CAN. The communication is performed via a set of periodic messages, $\Gamma = \{M_1, M_2, \ldots\}$, with various criticality levels. The criticality of a message indicates the severity of the consequences caused by its failure and corresponds to the amount of resources allocated for error recovery in terms of guaranteed re-transmissions. The basic assumption here is that the effects of a large variety of transient and intermittent hardware faults can effectively be tolerated by a simple re-transmission of the affected frames. We assume that an error can adversely affect only one message frame at a time and is detected by the nodes in the network. Γ_c represents the subset of critical messages out of the original message set and Γ_{nc} represents the subset of non-critical messages, so that $\Gamma = \Gamma_c \cup \Gamma_{nc}$.

A message consists of one or more frames. The network communication is non-preemptive during the frame transmissions. However, messages can preempt each other at frame boundaries. The non-preemptiveness of message frames may cause a higher priority message to be blocked by a lower priority message for at most one frame length, if the high priority message is released during the transmission of a lower priority frame. This phenomenon is called priority inversion, and it can affect all messages except the lowest priority one and only once per message period, before the transmission of the first message frame [15].

Each CAN message M_i is characterized by a 5-tuple $< P_i, T_i, D_i, N_i, R_i >$, where P_i is the priority (defined by the message identifier), T_i is the period, D_i is the relative deadline, which is assumed to be equal to the period, N_i is the number of frames that forms this message and R_i is the re-transmission requirement represented as the percentage of this message size. The number of frames needs to be guaranteed for re-transmission r_i is calculated by

$$r_i = \lceil N_i * R_i \rceil \tag{5}$$

Note that for non-critical messages $R_i = 0$.

In a no-error scenario, the worst case transmission time C_i of message M_i is

$$C_i = N_i * f * \tau_{bit} \tag{6}$$

where f is the worst case packet size after worst case bit stuffing.

For each *message instance* M_i^j we define an *original feasibility window* delimited by its original earliest start time $est(M_i^j)$ and deadline D_i^j relative to the start of the hyperperiod (LCM).

Obviously, the maximum utilization of the original critical messages together with the re-transmissions can never exceed 100%. This will imply that, during error recovery, transmission of non-critical messages cannot be permitted as it may result in overload conditions, except in situations where a non-critical frame is blocking a higher priority critical message due to priority inversion. We assume that, upon receiving an error frame, the nodes transmitting non-critical messages suspend their transmissions until the end of the failed message's period. This will require that all nodes transmitting non-critical messages have knowledge about critical messages' periods.

IV. Methodology

In this paper we propose an approach to enable fault-tolerant scheduling of messages with mixed criticalities in CAN.

A. Overview

As the original feasibility windows and original priority assignment (if any, e.g., in case of a legacy system) may not express the various FT requirements, our goal is to, first, derive new feasibility windows for each message instance $M_i^j \in \Gamma$ to reflect the FT requirements. Then, we assign message identifiers (priorities) that ensure the message transmissions within their new feasibility windows, thus, fulfilling the FT requirements.

While transmitting non-critical messages using a background priority band can be a safe and straightforward solution, in our approach we aim to provide non-critical messages a better service than background scheduling. Hence, depending on the criticality of the original set of messages, the new feasibility windows we are looking for differ as:

1) *Fault-Tolerant* (FT) feasibility windows for critical messages
2) *Fault-Aware* (FA) feasibility windows for non-critical messages

While critical messages need to be entirely transmitted within their FT feasibility windows to be able to be feasibly retransmitted upon an error, according to the reliability requirements, the derivation of FA feasibility windows has two purposes: 1) to prevent non-critical messages from interfering with critical ones, by causing a critical message to miss its deadline, while 2) enabling the transmission of the non-critical messages at high priority levels in error free situations. Since the size of the FA feasibility windows depends on the size of the FT feasibility windows, in our approach we first derive FT-feasibility windows and then FA feasibility windows. Then, we assign message priorities to ensure the message transmissions within their newly derived feasibility windows.

A high priority message can be blocked by a low priority message at most one frame at the beginning of its transmission. Since the priority assignment is one of the outputs of our method, we need to account for the blocking frame in every message transmission by adding one additional frame during the analysis.

In some cases, however, a fixed priority scheme cannot express all our assumed FT requirements and error assumptions. General FT requirements may require that instances of a given set of periodic messages needs to be transmitted in different order on different occasions. Obviously, there exists no valid fixed priority assignment that can achieve these different orders. Our approach proposes a priority

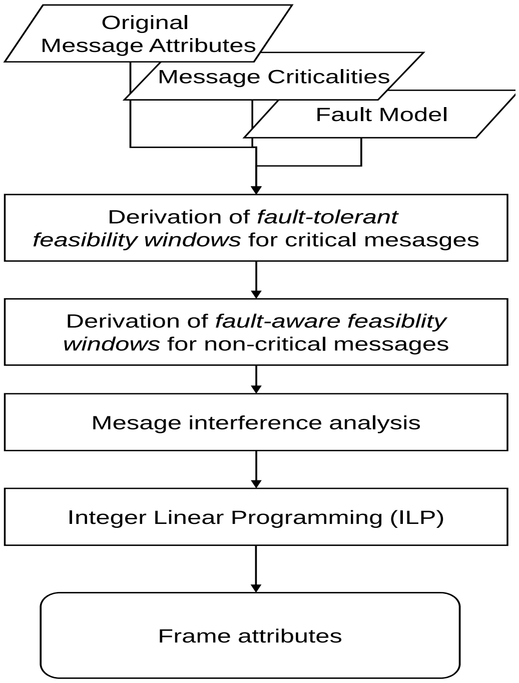
Original
Message Attributes
Message Criticalities
Fault Model
Derivation of *fault-tolerant feasibility windows* for critical mesasges
Derivation of *fault-aware feasiblity windows* for non-critical messages
Mesage interference analysis
Integer Linear Programming (ILP)
Frame attributes

Figure 2. Methodology overview

allocation scheme based on EDF at message instance level that efficiently utilizes the resources while minimizing the priority levels. We use Integer Linear Programming (ILP) to off-line analyze the interference between the message frames and to derive the minimum number of priorities that guarantees the message transmissions within their FT/FA Feasibility Windows.

The major steps of the proposed methodology are shown in Figure 2.

B. Proposed approach

We describe our proposed approach by using a simple example. Let our set of messages consist of 2 messages, A and B, where $T(A) = 8,\ T(B) = 16,\ N(A) = 3$ and $N(B) = 6$, i.e., message A is allocated to 3 frames that need to be transmitted during one period and message B is allocated to 6 frames. Moreover, let us assume B is the only critical message and has a re-transmission requirement $R_B = 80\%$, i.e., $N_{max}(B) = 6 + 5 = 11$ frames. If the messages are scheduled on CAN according to Rate Monotonic (RM) scheduling policy, the transmission scenario is illustrated in Figure 3). Note that message A may be blocked by the lower priority message B by at most one frame at the beginning of its transmission. The earliest start times and the deadlines are represented by up- and down arrows respectively.

To be able to re-transmit 80% of its frames before its deadline, B must complete before $D(B) - C(B)$. In this case, B's new deadline will be 11. One possibility is to assign B a higher priority than A. However, by doing so, the first instance of A will always miss its deadline, even in error-free scenarios (Figure 4). Moreover, this approach is not useful if a larger number of critical messages need to be feasibly transmitted on the bus.

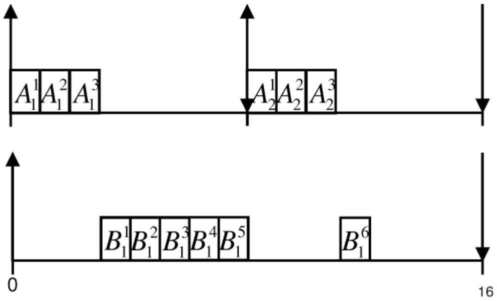

Figure 3. Original message set

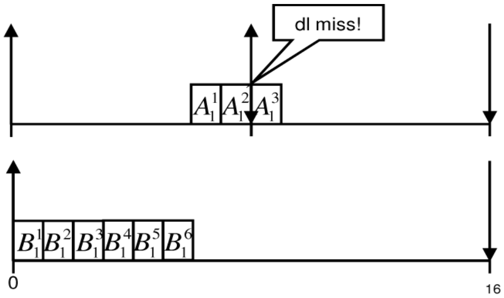

Figure 4. 'B' fault-tolerant - 'A' always misses its deadline

1) Derivation of FT- and FA feasibility windows: The first part of our approach is the derivation of FT and FA feasibility windows for critical and non-critical message instances respectively. Our approach first derives FT deadlines for the critical message instances so that, in case of an error, the erroneous frames can be re-transmitted before the original message deadline. Then, FA deadlines for the non-critical message instances are derived so that the provided fault-tolerance for the critical ones is not jeopardized. During these steps the goal is to keep the FT and FA deadlines as late as possible in order to maximize the flexibility for the second part of our approach, which is the priority assignment using an ILP solver.

Derivation of FT deadlines:: The aim of this step is to reserve sufficient resources for the re-transmission of erroneous frames in the schedule. Our goal is to provide guarantees while maximizing the bus utilization. Thus, we choose the approach proposed by Chetto and Chetto [16] to calculate the latest possible start of re-transmission of the erroneous frames. Specifically, we calculate FT-deadlines for each critical message instance, $D_{FT}(M_i^j)$, equal to the latest start time of the first re-transmitted frame. In this way we reserve sufficient resources for each critical message instance alternate, assuming that the cumulative resource utilization of the critical messages and re-transmissions, including blockings does not exceed 100% over LCM. In our example, the FT deadline of B is 11.

Derivation of FA deadlines:: We aim to provide FA deadlines to non-critical message instances to protect critical ones from being adversely affected. However, as a part of recovery action upon errors, the sending node should check if there is enough time left for the non-critical messages to be sent before their new deadlines. If not, the message is not transmitted.

To derive the FA deadlines, we repeat the process as in FT deadline derivation, on the set of non-critical messages but *in the remaining slack* after the critical messages (without re-transmissions) are scheduled to be transmitted as late as possible. We do so due to two reasons: we want to prevent non-critical messages from delaying the transmission of critical messages beyond their FT deadlines in case of critical frame failures, as well as to alow non-critical messages to be transmitted at high priority levels in error free scenarios. In our example the derived FT and FA deadlines are illustrated in Figure 5, where the FA deadlines for the instances of A are 4 and 16 respectively.

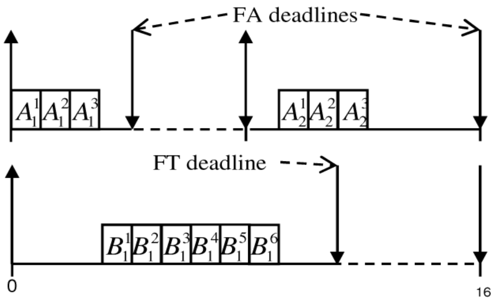

Figure 5. FT and FA deadlines

In some cases, we may fail finding valid FA deadlines for some non-critical messages instances. We say that a FA deadline, $D_{FA}(M_i^j)$, is *not valid* if $D_{FA}(M_i^j) - est(M_i^j) < C(M_i^j)$. This scenario could occur since the messages now may have deadlines less than periods after the derivation of FT deadlines. In these cases, we keep the original deadline, and we make sure that the priority assignment mechanism will assign the non-critical message a background priority, i.e., lower than any other critical message, and any other non-critical message with a *valid FA deadline*.

2) FPS attribute assignment: We analyze the set of messages with new deadlines and identify priority relations for each point in time t_k at which at least one message instance (i.e., the first frame of the message) is released on the bus. We derive priority inequalities between messages to ensure their transmission within their derived FT- and FA feasibility windows. By solving the inequalities, our method generates scheduling attributes for the message set Γ_{FPS}.

Our model consists now of four types of message frames: critical messages consisting of primary frames Γ_c and re-transmitted frames $\bar{\Gamma}_c$, and non-critical messages, consisting of non-critical frames, with and/or without valid FA deadlines, $\Gamma_{nc} = \Gamma_{nc}^{FA} \cup \Gamma_{nc}^{non\_FA}$. Every $t_k \in [0, LCM)$ such that t_k equals the release time of at least one message instance, we consider a subset $\Gamma_{t_k} \subseteq \Gamma$ consisting of:

1) $\{current\_instances\}_{t_k}$ - instances M_i^j of message M_i, released *at* the time t_k: $est(M_i^j) = t_k$
2) $\{interfering\_instances\}_{t_k}$ - instances M_s^q of message M_s released *before* t_k but potentially executing *after* t_k: $est(M_s^q) < t_k < D(M_s^q)$, where

$$D(M_s^q) = \begin{cases} D_{FT}(M_s^q), & if\ M_s^q \in \Gamma_c \\ \overline{D}_{FT}(M_s^q), & if\ M_s^q \in \bar{\Gamma}_c \\ D_{FA}(M_s^q), & if\ M_s^q \in \Gamma_{nc}^{FA} \\ D(M_s^q), & if\ M_s^q \in \Gamma_{nc}^{non\_FA} \end{cases}$$

We derive priority relations within each subset Γ_{t_k} based on the derived FT and FA deadlines, i.e., the message with the shortest relative deadline will get the highest priority in each inequality:

$\forall t_k, \forall M_i^j, M_s^q \in \Gamma_{t_k}$, where $i \neq s$:

1) if $M_i^j, M_s^q \in \Gamma_c \cup \Gamma_{nc}^{FA}$, or if $M_i^j, M_s^q \in \Gamma_{nc}^{non\_FA}$

$$P(M_i^j) > P(M_s^q),\ where\ D(M_i^j) < D(M_s^q)$$

2) if $M_i^j \in \Gamma_c \cup \Gamma_{nc}^{FA}$ and $M_s^q \in \Gamma_{nc}^{non\_FA}$

$$P(M_i^j) > P(M_s^q)$$

In tie situations, e.g., when the message instances M_i^j and M_s^q have same deadlines, we prioritize the one with the earliest start time. In cases where even the earliest start times are equal, we derive the priority inequalities consistently.

Our goal is to provide fixed priorities to all messages. When we solve the derived priority inequalities, however, it may happen that different instances of the same message need to be assigned different priorities, due to the EDF heuristic used in the approach. These cases cannot be expressed directly with fixed priorities and are the sources for *priority assignment conflicts*.

We solve the issue by splitting the messages with inconsistent priority assignments into a number of new periodic messages with different priorities. The new message instances comprise all instances of the original set of messages. As a major concern is the number of priorities that may increase, we use ILP to find the priorities and the splits that yield the lowest number of messages that satisfy the inequalities, and implicitly the lowest number of priority levels. A description of the ILP problem formulation that can be easily adapted to CAN scheduling can be found in [17].

A major difference in CAN scheduling compared to task scheduling on processors is that frames are re-transmitted as soon they are identified as erroneous, rather than after the transmission of the whole message. Hence, a message consisting of N primary frames and R re-transmitted frames may need to be transited at a priority level p the first N frames, and at a priority p' for the rest R frames. However, ILP will make sure that, if possible, $p = p'$.

The final set of messages feasibly scheduled on CAN is presented in Figure 6. A1 has the highest priority and A2 the lowest. In Figure 6 maximum number of re-transmissions are performed upon transmission errors. In this case, A2 will be shed by the scheduler due to the system overload.

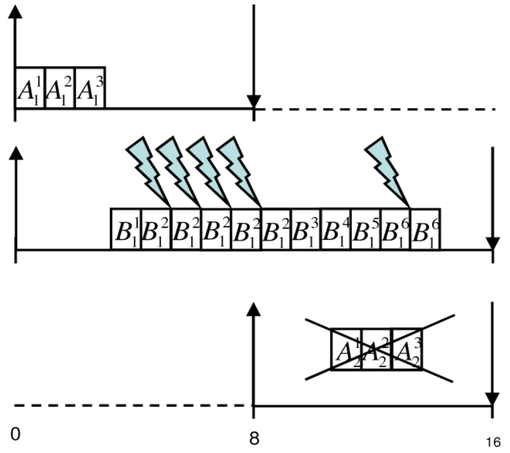

Figure 6. FT feasible message set

V. Conclusions

In this paper we have presented a new approach for the fault-tolerant scheduling of messages of mixed criticality in the Controller Area Network which enables the user to guarantee a variable level of redundancy for critical messages depending on their criticality levels. The new concepts that we have introduced are that of a fault-tolerant window for critical messages, fault-aware windows for non-critical messages, a variable level of redundancy requirement per message, and the optimal derivation of message priorities using ILP. We believe that such a methodology in principle could be further extended in other contexts of scheduling paradigms and networks. Our ongoing research includes formalization of utilization bounds as well as simulation studies to evaluate the effectiveness of the proposed methodology.

References

[1] N. Navet, “Controller Area Networks: CAN's use within Automobiles,” *IEEE Potentials*, pp. 12–14, October/November 1998.

[2] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg, “Volcano - a revolution in on-board communication,” *Volvo Technology Report 98-12-10*, 1998.

[3] I. Noble, “EMC and the Automotive Industry,” *IEE Electronics & Communication Engineering Journal*, pp. 263–271, October 1992.

[4] M. Short and M. Pont, “Fault-tolerant time-triggered communication using CAN,” *IEEE Transactions on Industrial Informatics*, vol. 3, no. 2, pp. 131–142, 2007.

[5] K. Tindell, A. Burns, and A. Wellings, “Calculating controller area network CAN message response times,” *Control Engineering Practice*, vol. 3, pp. 1163–1169, 1995.

[6] N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, “Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling,” *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, September 1993.

[7] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, “Controller area network (can) schedulability analysis: Refuted, revisited and revised,” *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[8] K. W. Tindell, A. H. Hansson, and A. J. Wellings, “Analysing Real-Time Communications: Controller Area Network (CAN),” *IEEE Real-Time Systems Symposium*, pp. 259–265, December 1994.

[9] S. Punnekkat, H. Hansson, and C. Norström, “Response time analysis under errors for CAN,” in *Proceedings of the* 6^{th} *IEEE Real-Time Technology and Applications Symposium (RTAS'00)*. Washington DC, USA: IEEE Computer Society, May-June 2000, pp. 258–265.

[10] I. Broster, “Flexibility in dependable real-time communication,” Ph.D. dissertation, Department of Computer Science, University of York, 2003.

[11] C. Bartolini, G. Lipari, and L. Almeida, “Using priority inheritance techniques to override the size limit of CAN messages,” *Proceedings of the* 7^{th} *IFAC International Conference of Fieldbuses and Networks in Industrial and Embedded Systems (FET)*, 2007.

[12] J. Charzinski, “Performance of the Error Detection Mechanisms in CAN,” *Proceedings of the 1st International CAN Conference, Mainz*, September 1994.

[13] ISO-11898, “Road Vehicles - Interchange of digital information - Controller area network (CAN) for high speed communication,” 1993.

[14] “CAN in Automation, CAN Specifications,” *http://www.can-cia.org*.

[15] M. Di Natale, “Scheduling the CAN bus with earliest deadline techniques,” in *Proceedings of the* 21^{st} *IEEE International Real-Time Systems Symposium (RTSS'00)*, Orlando, FL, USA, November 2000, pp. 259–268.

[16] H. Chetto and M.Chetto, “Some results of the earliest deadline scheduling algorithm,” *IEEE Transactions on Software Engineering*, vol. 15, no. 10, pp. 1261–1269, 1989.

[17] R. Dobrin, H. Aysan, and S. Punnekkat, “Maximizing the fault tolerance capability of fixed priority schedules,” *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2008.