Journal of
Engineering, Computing and Architecture ISSN 1934-7197
Volume 2, Issue 1, 2008

An Efficient Algorithm for the Computation of Response-time Bounds for CAN Messages

Imad Alzeer, Assis. Prof., Al-Quds University, im_alzeer@eng.alquds.edu
Naji Qatanani, Assoc. Prof., Al-Quds University, nqatanani@science.alquds.edu

Abstract
This paper presents an efficient computational exhaustive method that permits to calculate both upper and lower response-time bounds for CAN messages. Response-time analysis for CAN messages is relatively limited for computations of the worst case situation. It is computed assuming a maximum transmission time and critical instant releasing of messages in the CAN system. This pessimism implies the maximum interference between messages circulated on the bus. It may be correct from a hard real-time perspective when synchronous releasing, but it doesn't give good outlook when non-common messages releasing. Hence to obtain an analysis close to the reality, the investigated temporal constraints must take into account both effects of time phasing and bit-stuffing. By using a suitable data structure, our work introduces an elegant algorithm that is able to deal with the previous effects. The obtained results for best and worst cases response-times are different from previous results obtained when assuming an optimist and a pessimist bit-stuffing length.

Keywords: CAN messages, bit-stuffing, scheduling, response-time, computational algorithm.

1. Introduction

A lot of work has been done and many algorithms have been developed to compute the worst case response-time for CAN messages [4, 7]. The majority of the present studies deal with the circumstances where messages are queued at the critical instants. Because the schedulability analysis is quite pessimistic, it assumes that a missed deadline in the worst case is equivalent to always missing the deadline for all instant messages [13]. So in messages scheduling, the reliability is regarded as an objective issue and the performance of any scheduling algorithm is measured generally by two factors:

1. Its ability to generate a feasible schedule for the set of messages covering all possible combinations of transmitted messages.
2. Determining whether these messages will meet their deadlines or not.

Bit-stuffing is performed by CAN to maintain the phase-locked bit timing. When the transmitter logic detects five consecutive bits of the same level, it inserts a sixth complementary bit into the original stream [5]. According to the content of original message, an extra number of bits (stuff bits) is inserted and merged with the original frame.

Since the scheduling on CAN is non-preemptive, and despite of a fixed number of data bytes that may be conveyed in each message, the number of inserted stuff bits resulting from stuffing rule may vary from any transmission period to the next (when the transmitted data is variable). Therefore certain disordering in the transmission sequence will occur.

This paper extends a scheduling analysis to allow computations of the worst and the best case response-time taking into account the variation in stuff bits length when error-free message transmission.

2. CAN Messages

According to the terminology of CAN, two types of messages in CAN system may be used:

- Standard CAN frame with 11 bits identifier;
- Extended CAN frame with 29 bit identifier.

Value of the identifier is assigned statically before starting the communication process. As shown in Fig. 1, an extended CAN message format contains 67 bits of protocol control information, including 29 bits identifier associated with each message assigning its priority, 4 bits for a message length field, 15 bits for CRC field, 7 bits for the end-of-frame signal, and 3 bits for the intermission between frames.

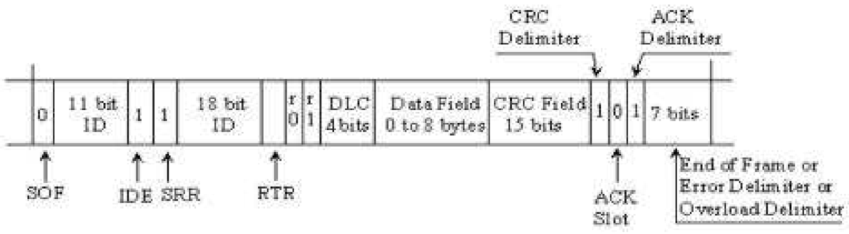


Figure 1: Extended message format in CAN

2.1. Message parameters

We suppose that the functionality of the system is guaranteed by a set of pre-determined number n of messages circulated on a CAN bus. Each message $m_i \in M$ such $M = \{m_i : i \in [1, n]\}$ may be defined as: $m_i(\pi_i, a_i, b_i, T_i, d_i)$, where

- The parameter π_i represents message's priority and it can be deduced from its period T_i in the case of Rate Monotonic [12], deadline or a value that depends on context of the application. Priorities are unique and indicated by the values of identifiers which are assigned in decreasing order, so the lowest value of π is attributed to the highest priority message in **M**.
- The parameter a_i represents time phasing or the arrival time at which message m_i is deposed and being ready for transmission.
- The number of data bytes that are conveyed in the message is represented by b_i.
- The parameter d_i represents the absolute deadline of the message. Very often, in distributed systems where CAN messages are the communication entities between tasks residing on different processors, it is possible to have deadlines greater than the period.

According to specifications of CAN [14], bit-stuffing rule has to be subjected on all bits starting with SOF bit until the last bit of CRC field that shown in Fig. 1. Thus, an extended CAN frame is specified such that only 54 of the 67 control bits are subjected to bit-stuffing rule. Thus, each message m_i has a certain part that represents the lower bound $C_i^{\downarrow}$ of its length, while the upper limit of this length is represented by $C_i^{\uparrow}$. According to the type of CAN frame and as a function of b_i, the two bounds are calculated in terms of bit time (τ_{bit}) as shown in Table 1.

Type	$C^{\downarrow}$	$C^{\uparrow}$
Standard	$(47 + 8 * DLC) * T_{bit}$	$(C^{\downarrow} + \lfloor \frac{34+8*DCL}{5} \rfloor) * T_{bit}$
Extended	$(67 + 8 * DLC) * T_{bit}$	$(C^{\downarrow} + \lfloor \frac{54+8*DCL}{5} \rfloor) * T_{bit}$

Table 1: Bounds of CAN message length

3. Scheduling and analysis

Message response-time will be denoted by R_i with the two superscripts $R_i^{\downarrow}$ and $R_i^{\uparrow}$ are the lower "best" and the upper "worst" case values respectively. R_i of message m_i is calculated as the interval between release instant a_i of the message and the latest transmission instant of that message. To guarantee the timing requirement of the system, the maximum response time $R_i^{\uparrow}$ has to be not greater than the permitted deadline d_i of the message.

Since messages scheduling on CAN is non-preemptive, an arrival of higher priority message may not-coincide with an instant of scheduling. Thus, the higher priority message will suffer from a bounded blocking time B_i before entering the next priority order competition [15]. The upper limit of blocking-time is given by:

$$B_i = \max_{k \in lp(i)} (C_k^{\uparrow} - 1) \tag{1}$$

where lp represents the subset of messages that are lower priority than the message m_i.

To focus on the effect of incertitude in the length and the time phasing on response-time computations, the releasing jitter delay of the message is integrated in the time phasing parameter a_i and will not be appeared or discussed here.

3.1. Scheduling Period

The proposed CAN model that we will study allows messages to be released asynchronously with arbitrary periods. Hence according to [11], when the messages are released synchronously, the schedule period *SP* is calculated as the least common multiple (LCM) of their periods, otherwise $SP = \max_{i=\{1..n\}}(a_i) + 2 * LCM$. Many studies are done to reduce the SP in the asynchronous release case [9]. To keep the scheduling duration small, whenever this is possible, the approach is to make message periods multiples of one another.

Now, to be familiar with the effect of bit-stuffing length on the response-time, we will investigate the scheduling interaction of the 3 independent CAN messages illustrated in Fig. 2. We assume that priority of m_1 is higher than that of m_2, and the priority of m_2 is higher than that of m_3.

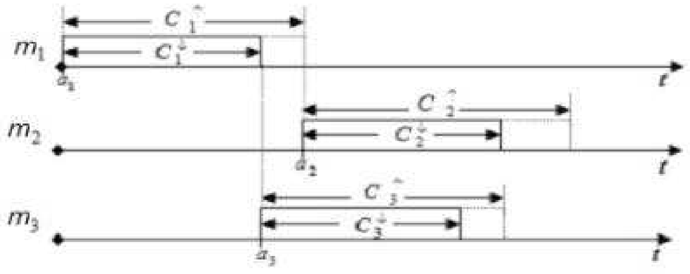


Figure 2: Configuration of 3 asynchronous messages

When assuming the traditional pessimist bit-stuffing result, each message has a longest possible transmission time. This assumption leads to the following worst response-times:

$$\begin{aligned} R_1^{\uparrow} &= C_1^{\uparrow} \\ R_2^{\uparrow} &= C_1^{\uparrow} + C_2^{\uparrow} \\ R_3^{\uparrow} &= C_1^{\uparrow} + C_2^{\uparrow} + C_3^{\uparrow} \end{aligned}$$

When the effect of time phasing and the incertitude in stuff bits are considered, messages transmission sequence may vary from one period to another. This variation will influence strongly the response-time bounds. Concerning the previous example and with the progress of time, two transmission sequences as shown in Fig. 3 are possible.

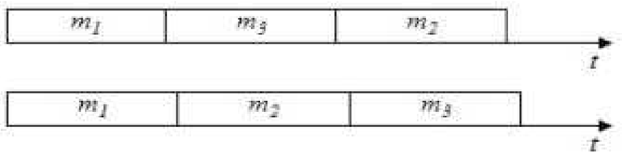


Figure 3: Two possible transmission sequences

3.2. Message deployment

Along the scheduling period SP, each message is represented by a set of instances that have their own parameters. Since the deadline of a message may be greater than its period, precedence constraints between the successive message instances have to be maintained. The system is schedulable if all messages in the system are schedulable and no message misses a deadline during SP.

Message deployment is achieved as follows: during SP, each m_i is deployed by a set of k_i instances, where $k_i = \left\lfloor \frac{SP}{T_i} \right\rfloor$. Thus, the extended set X contains $K = \sum_{i=1}^{n} k_i$ of deployed instances that have the same period SP. Parameters of the j^{th} instance of the message m_i are calculated as:

- since the message has a fixed priority, all instances of the message have the same priority: $\pi_i^j = \pi_i$;
- activation instants are shifted by the period and related with time phasing of the first instance: $a_i^j = a_i + (j-1) * T_i$;
- two bounds are the same of the original message: $C_i^{m\downarrow} = C_i^{\downarrow}$ and $C_i^{m\uparrow} = C_i^{\uparrow}$;
- absolute deadline of the instance is related with the period and the deadline of the first instance: $D_i^j = D_i + (j-1) * T_i$.

The deployment mechanism is explained by taking the 3-messages shown in 2. Without loss of the generality and for simplicity, we assume that $T_3 = 2 * T_1 = 2 * T_2 = 30$ and other parameters are: $m_1 = (\pi_1, a_1, C_1^{\downarrow}, C_1^{\uparrow}) = (1, 0, 3, 4), m_2 = (2, 4, 3, 5)$ and $m_3 = (3, 3, 3, 4)$. Since all instances of messages start during the LCM and terminates before the time instant 30, SP can be considered as LCM. So, the extended list X contains 5 elements as shown in Fig. 4.

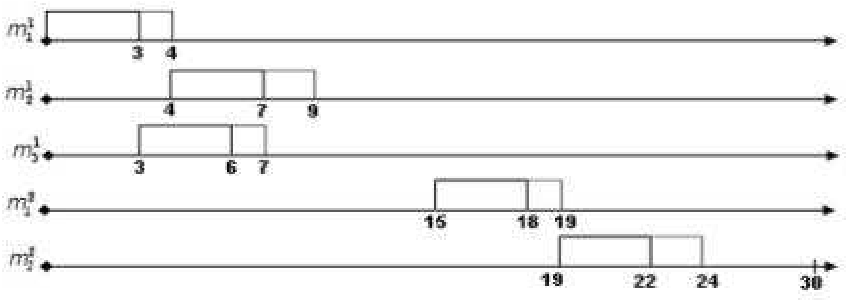


Figure 4: Deployed instances of 3 messages

3.2.1. Activation array

Since each deployed instance has its own activation date, elements of the extended list X are arranged according to their activation instants. When many instances are activated at the same instant, then they are mutually referenced. This mechanism prevents us from doing any non-necessary manipulation during the calculation. Concerning our example of five instances, this array is shown in Table 2.

#	instant	Instance
1	0	m_1^1
2	3	m_3^1
3	4	m_2^1
4	15	m_1^2
5	19	m_2^2

Table 2: Array of activation dates

3.3. Transmission Sequences

The large number of combinations that results from the application of bit-stuffing do not lead mostly to the same number of the transmission sequences. The number of combinations resulted from N instances is

$$\prod_{i=1}^{N} (C_i^{\uparrow} - C_i^{\downarrow} + 1).$$

Concerning our previous example, the incertitude in the length of 5 instances leads to 72 combinations as shown in Fig. 5. For simplicity we show partially the upper part of the combinational tree. The

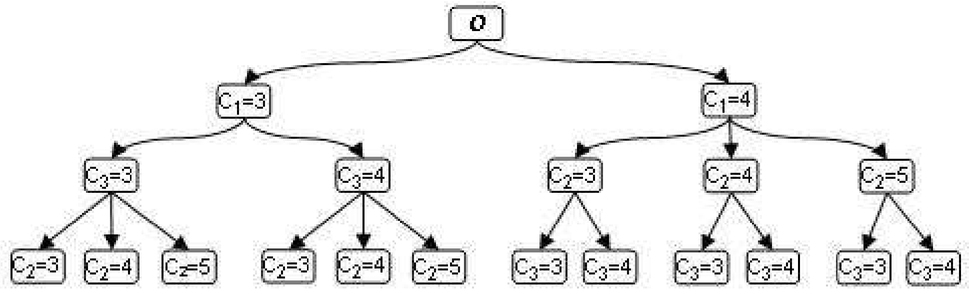


Figure 5: Tree transmission combinations

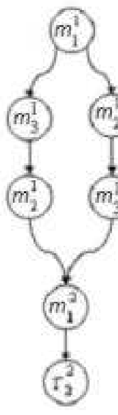


Figure 6: Transmission sequences

incertitude in the transmission duration leads only to the 2-sequences (scenarios) that are shown by the graph in Fig. 6. This graph that symbolizes the transmission sequence has graph acyclic [8] form.

We remark that: whatever the length of m_1^1, m_3^1 and m_2^1, only two sequences are possible : the first one is composed from $m_1^1 - m_3^1 - m_2^1$ and the second is composed from $m_1^1 - m_2^1 - m_3^1$. At the next scheduling instant (15), whatever the length of m_1^2 and m_2^2 are, the next sequence will be $m_1^2 - m_2^2$.

Conclusion and motivations: We note that when the next activation do not effect on the current transmission sequence, the new combinations will not bring more information about the response time. Hence, the investigation of the possible sequence (or scenarios) of transmission becomes our main objective [3, 1, 2].

3.4. Anticipated prevision about transmission sequences

In this section we will present a mechanism that permits us to verify whether the variation in the transmission time will not change the transmission sequence. This mechanism is based, on one hand, on the calculation of the busy period BP [10], and on the other hand, on the $k-level$ busy period. For the k^{th} instance that has priority π_k, the $k-level$ busy period is denoted by BP_k.

3.4.1. Calculation of BP and BP_k

Due to the variation in the transmission duration ($C_i^{\downarrow} \leq C_i \leq C_i^{\uparrow}$), the busy period BP will be bounded as: $BP^{\downarrow} \leq BP \leq BP^{\uparrow}$. As shown in Fig. 7, at the current instant t_c the bus is occupied by the transmission of the j^{th} instance in the extended list X (i.e. $j \in [1, K]$) since dur_j time instants.
Analytically, calculations of the two bounds is released as follows:

$$BP^{\downarrow} = \sum_{i \in I_{min}} C_i^{\downarrow} + \begin{cases} C_j^{\downarrow} - dur_j(t_c), & C_j^{\downarrow} > dur_j(t_c) \\ 0 \quad , & otherwise \end{cases} \tag{2}$$

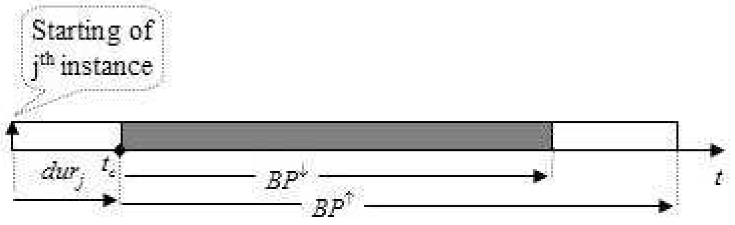


Figure 7: Busy period bounds

$$BP^{\uparrow} = \sum_{i \in I_{max}} C_i^{\uparrow} + C_j^{\uparrow} - dur_j(t_c) \tag{3}$$

The subsets I_{min} and I_{max} are related to $BP^{\downarrow}$ and $BP^{\uparrow}$ respectively. They result from:

1. Ready instances at the current instant t_c.
2. Those instances will be ready during the concerned busy period ($BP^{\downarrow}$ or $BP^{\uparrow}$)

However, the lower bound of $k - level$ busy period $BP^{\downarrow}$ is calculated as:

$$BP_k^{\downarrow} = \sum_{i \in (I_{min} \cap (hp(k) \cup k))} C_i^{\downarrow} + \begin{cases} C_j^{\downarrow} - dur_j(t_c), & C_j^{\downarrow} > dur_j(t_c) \\ 0 \quad , & otherwise \end{cases} \tag{4}$$

The subset $(I_{min} \cap (hp(k) \cup k))$ consists of all instances having a priority level equal or greater than that of the k^{th} instance providing that these instances are activated during the continuous period $BA_k^{\downarrow}$. Since all instances are included in the activation array, process of calculation and verification is effectuated iteratively.

We explain the previous method by an example of 3 configurations as shown in Fig. 8. At the time instant t_c, the 3 configurations lead to the same bounds: $BP^{\downarrow} = 14$ and $BP^{\uparrow} = 19$, but their transmission sequences may be different. Thus, according to the category of activation, we can judge if the trans-

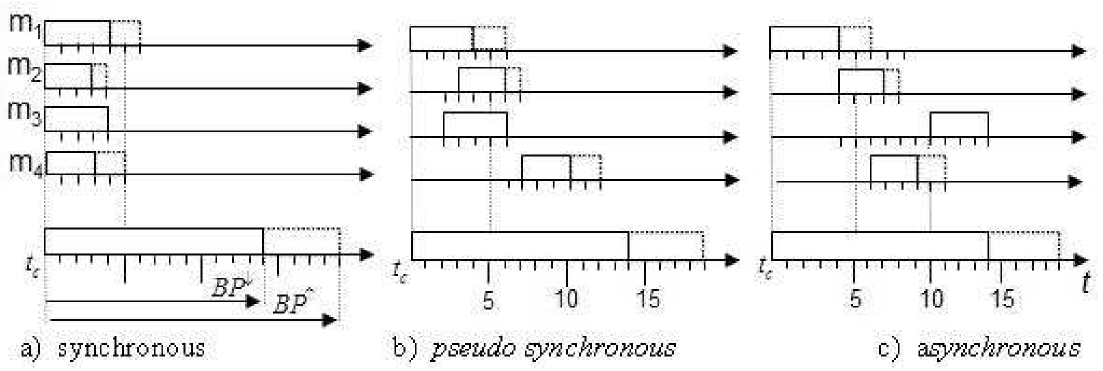


Figure 8: Different release configurations

mission sequence is certainly unique (unique transmission scenario) or not. Generally, three activation categories may be distinguished:

1. Synchronous activations: when all instances belong to the busy period are activated at the same instance (Fig. 8-a). In this case, whatever their durations, the transmission sequence will be unique.
2. Pseudo synchronous activations: assuming that $\pi_1 < \pi_2 < \pi_3 < \pi_4$, as shown in Fig. 8-b, the activation of an instance occurs certainly before the transmission of any other lower priority instance. This implies that the transmission of all instances belong to the busy period occurred according to their priority order.

3. Asynchronous activations: the transmission of the instances can not be guaranteed to agree with their priorities as shown in Fig. 8-c.

Due to the incertitude in the duration of m_1 and m_2, at $t = 10$, there is no guarantee that m_4 (which is less priority than m_3) has not yet started its transmission. Hence, to make sure that the instance j having a priority π_j is not preceded by the transmission of another less priority instance, the condition: $\forall t_c, a_i \geq t_c, a_j \in [t_c, BP_j^{\downarrow}]$ must holds.

4. Verification model

The investigated verification method is based basically on the calculation of response time boundaries. Because of the incertitude in the transmission duration, exhaustive calculations imply the prevision of all possible transmission scenarios (sequences), thereby calculate response times for instances composing each possible scenario. A performant prevision policy must be capable to predict and to verify all probable combinations of transmission patterns during the hyper-period. The system is schedulable if all instances in the system are schedulable and no one misses a deadline during the schedule period.

4.1. Computational method

The fundamental idea in our computational method is based on the investigation of all possible scheduling sequences of deployed instances in the list **X**. Since the deadlines (in our model) are not related to periods, a **FIFO** policy is used to schedule the successive instances of the same message.

During the scheduling process along the hyper-period, an instance can be in one of the following three states:

- Waiting to be released;
- Ready to run but not running;
- Running.

A released instance will be denoted as an activity. Each activity represented by a compact data structure called activity node (AN) that shown in Fig. 9-a. Each AN is symbolized by data structure that is capable to link the scheduled activities in a way giving low size scheduling map. Thus, the AN has the following data fields:

- **Ind** as the index of the activity in X.
- **Dur** to indicate the assumed duration of the CA that produces the current scenario.
- **Pre** to link the activity node with another node in the same scheduling scenario.
- **EOE** (**E**nd **O**f **E**xecution) to indicate the completion instant of the CA.

At each instant along the schedule period, the scheduling process on a resource is symbolized by a resource node (RN) data structure as shown in Fig. 9-b.

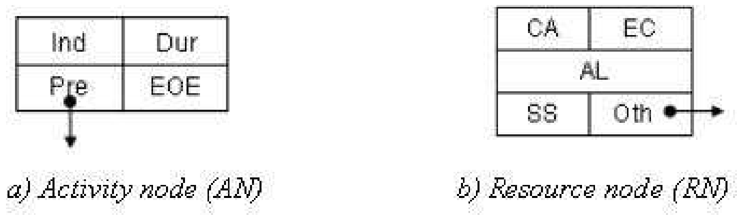


Figure 9: Used data structures

The RN data structure has the following data fields:

- **CA** to indicate the current activity that is in execution. If no activity is running $CA = -1$.
- **EC** as an execution counter for the activity executed on a resource. If no activity is in execution $EC = 0$.

- **AL** is the activity list that contains the dynamic list of all ready activities on the resource. Activities in the list are enchained as shown in Fig. 10

- **SS** to signify if the incertitude in the message length, absolutely, has no effect on the transmission sequence.

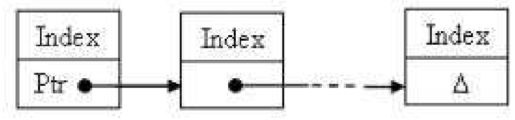


Figure 10: Enchained activities in the ordered AL

- **Oth** is used to enchain (at the current instant) the different resource node corresponding to the different possible scenarios. This permits to look into all existing nodes without knowing neither their contents nor their numbers in a way that leads to an acyclic graph form. The obtained acyclic graph is not ordered, so an activity in the map may have more than one node with the same EOE value.

Remark: List of activities have the following properties: activities are completely ordered so that the head activity has the highest priority, and the representation of lists is unique; there is only one node characterized by the values of Index and Ptr.

5. Activities submission

The submission of an activity implies the addition of that activity to the whole existing dynamic lists. So, any released instance will be inserted immediately in the existing ALs. When the bus becomes free and the list "RN.AL" is not empty, the higher priority activity HPA will be extracted to be scheduled. If the list RN.AL is empty, the corresponding resource node will be suppressed. This process is illustrated by Fig. 11.

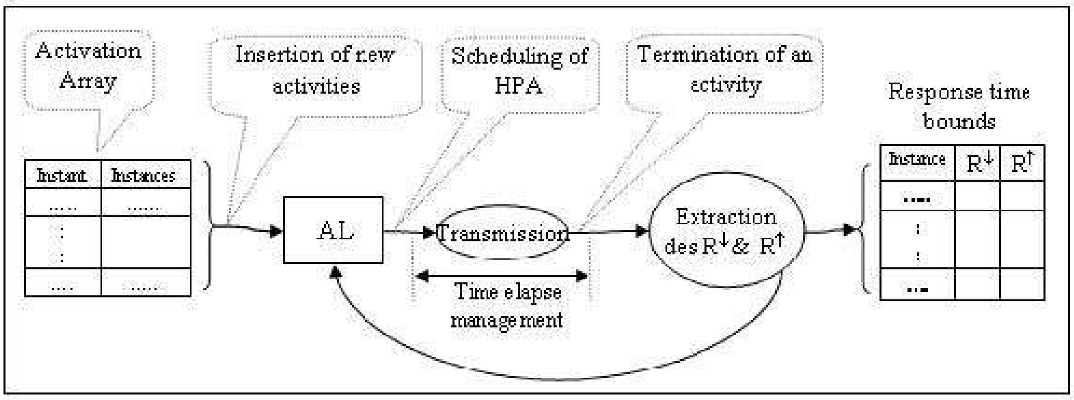


Figure 11: Submission process illustration

For simplicity, firstly we present in details the submission principles without optimization. In other words, we create all possible durations for instances that share the same activity period. Therefore, neither the uniqueness of transmission scenario realized by dint of the SS (Scenario Status) parameter, nor the concerned optimization will be applied. Then we apply the optimization techniques on the concerned parts.

5.1. Algorithm without optimization

The submission process is composed of 5 stages:

Stage 1: Initialization of intermediate index and response time bounds arrays.

Stage 2: Starting by zero instant, we look over all time instants such that $0 \leq Instant \leq SP$.

Stage 3: At this instant, we verify whether there are new released instances, if so, they will be inserted in all of the existing RN.AL. If no RN exists, a new RN will be created to include the released instance. The creation of an RN will be explained later.

Stage 4: Due to RN.Oth pointer, we can look quickly over all RNs performing the following jobs:

1. If $RN.CA = -1$, the head element which is the highest priority activity in RN.AL will be extracted and scheduled. Scheduling of the activity implies explicitly the creation of new AN and the suppression of that activity from the list AL.
2. Increment the execution counter $RN.EC++$. Supposing that the current activity (AN.CA) has the index i, we go to the following step.
3. If the current activity will be terminated surely ($RN.EC = C_i^{\uparrow}$), we do the following:
   - **a.** Memorizing the two bounds of response time; lower bound as $R_i^{\downarrow} = min(R_i^{\downarrow}, AN.EOE)$ and the higher one as $R_i^{\uparrow} = Max(R_i^{\uparrow}, AN.EOE)$.
   - **b.** We initialize the following parameters: $RN.EC = 0$ and $RN.CA = -1$ to signify that the current activity is terminated.
4. If the activity is not surely to be terminated ($C_i^{\downarrow} \leq RN.EC < C_i^{\uparrow}$), we do the following:
   - **a.** If $RN.EC = C_i^{\downarrow}$ we memorize $R_i^{\downarrow} = min(R_i^{\downarrow}, AN.EOE)$.
   - **b.** We clone the current RN producing **New_RN**. (this process will be explained later)

Stage 5: when the submission process on all RNs (enchained by the pointer Oth) is terminated, the similar RNs (with the same ECs and ALs) are merged to keep only one RN. Merging process lets us to apply the Acyclic Graph technique [8]. The Acyclic Graph offers a considerable reduction mechanism in the computations. The reduction is performed by lessening of next scheduling combinations.

5.1.1. Nodes creation

The creation of new RN implies the following initialization: $RN.CA = -1$ and $RN.EC = 0$. The cloning process of RN is achieved by the creation of New_RN and the initialization of this New_RN as follows: $New\_RN.AL = RN.AL$, $New\_RN.CA = -1$ and $New\_RN.EC = 0$.

5.1.2. Algorithmic description

In Fig. 12, we present the pseudo-code that describes the submission process (without optimization). It has as input the parameters of the deployed instances and as outputs the response time bounds for these instances. We use the following abbreviations:

K: number of instances;

SP: Scheduling period;

sai: static activation index;

SAT: Static Activation Table.

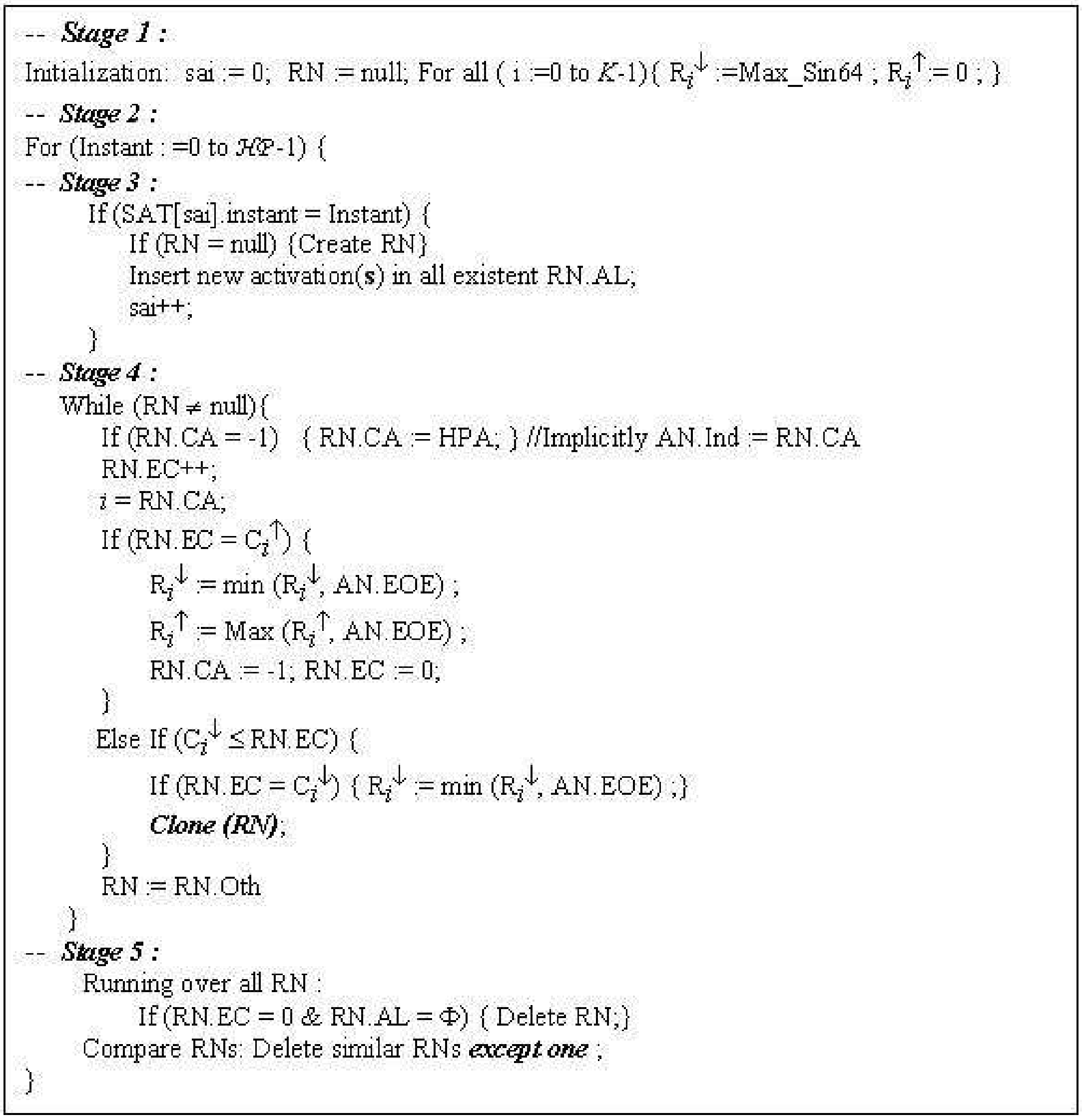
5.2. Submission with optimization

When the variation in the transmission times of the instances that share the current busy period do not have an influence on the transmission sequence, and the sequence is unique. Then, cloning process is not necessary to be done systematically and one scenario can be considered to generate the response time bounds of instances that share the busy period.

Therefore, to extract $R^{\downarrow}$ and $R^{\uparrow}$, we generate two scenarios; one is symbolized by **RN** and the other by **RN'**. The first (RN) is used to extract $R^{\downarrow}$s by considering the minimum duration bounds ($C_i^{\downarrow} : \forall i \in AL$). This is done by setting the parameter SS of RN as Lower Bound Unique Scenario (LBUS).

$R^{\uparrow}$s are extracted by considering the maximum duration bounds ($C_i^{\uparrow} : \forall i \in AL$). This is done by setting the parameter RN'.SS as Higher Bound Unique Scenario (HBUS). We remember that by default, at the instant of RN creation, the SS parameter is set as: Non Unique Scenario (NUS).

When we apply the optimization principles, the cloning process in the 4^{th} step of stage 4 (when $C_i^{\downarrow} \leq RN.EC < C_i^{\uparrow}$) will not be done systemically. Thus, the corresponding part of the algorithm becomes:



```
-- Stage 1 :
Initialization: sai := 0; RN := null; For all ( i :=0 to $K$-1){ $R_i^{\downarrow}$ :=Max_Sin64 ; $R_i^{\uparrow}$ := 0 ; }
-- Stage 2 :
For (Instant : =0 to $\mathcal{HP}$-1) {
-- Stage 3 :
        If (SAT[sai].instant = Instant) {
            If (RN = null) {Create RN}
            Insert new activation(s) in all existent RN.AL;
            sai++;
        }
-- Stage 4 :
    While (RN $\neq$ null){
        If (RN.CA = -1)   { RN.CA := HPA; } //Implicitly AN.Ind := RN.CA
        RN.EC++;
        $i$ = RN.CA;
        If (RN.EC = $C_i^{\uparrow}$) {
            $R_i^{\downarrow}$ := min ($R_i^{\downarrow}$, AN.EOE) ;
            $R_i^{\uparrow}$ := Max ($R_i^{\uparrow}$, AN.EOE) ;
            RN.CA := -1; RN.EC := 0;
        }
        Else If ($C_i^{\downarrow}$ $\leq$ RN.EC) {
            If (RN.EC = $C_i^{\downarrow}$) { $R_i^{\downarrow}$ := min ($R_i^{\downarrow}$, AN.EOE) ;}
            Clone (RN);
        }
        RN := RN.Oth
    }
-- Stage 5 :
    Running over all RN :
        If (RN.EC = 0 & RN.AL = $\Phi$) { Delete RN;}
    Compare RNs: Delete similar RNs except one ;
}
```

Figure 12: Pseudo code describing the submission process (without optimization)

Stage 4: ... 4) When $C_i^{\downarrow} \leq RN.EC < C_i^{\uparrow}$, we perform the following:

A. If $RN.EC = C_i^{\downarrow}$, we memorize the lower bound of response time as $R_i^{\downarrow} = min(R_i^{\downarrow}, AN.EOE)$,

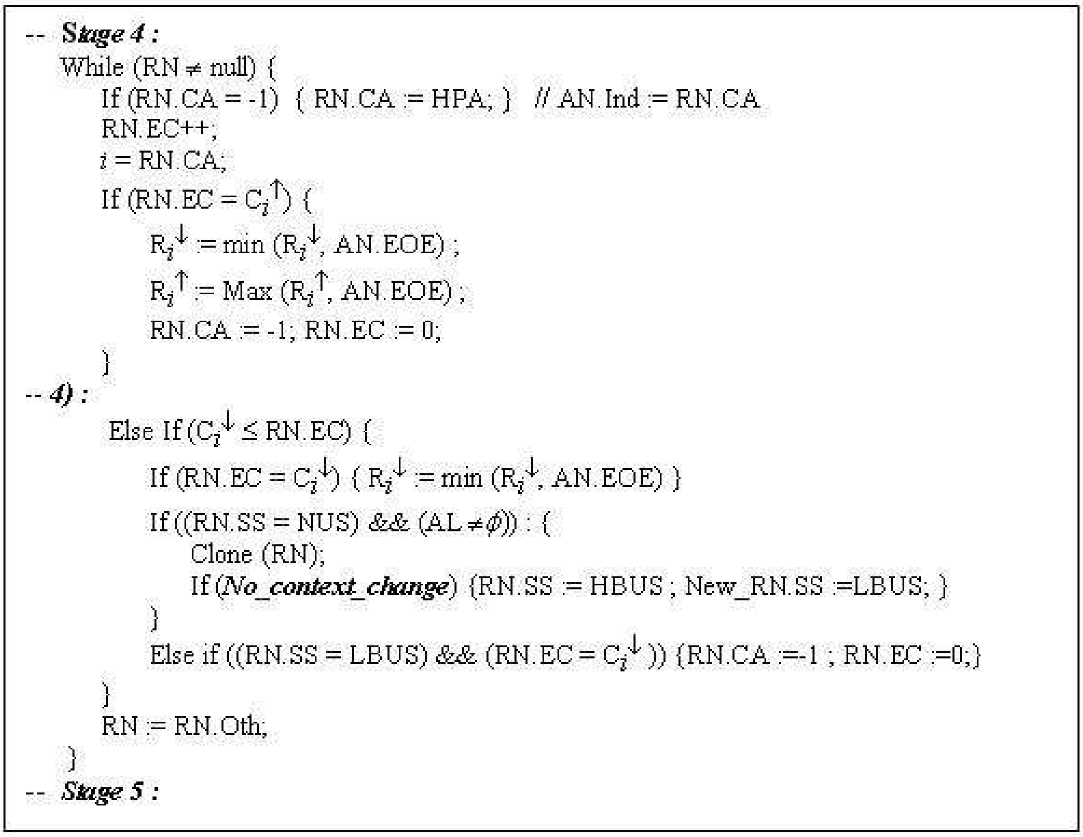
B. If $RN.SS = NUS$ (non unique scenario) and the current AL is not empty, we calculate the busy period Eq. 3, then we do one of the following operations:

- If the new activations (if any) during the current $PA^{\uparrow}$ do not change certainly the transmission sequence, we clone the current RN setting its SS as HBUS while New_RN.SS as LBUS.
- Otherwise, or in other words, if the new activations can change the context or it is not possible to determine if the new activations can change current sequence, we clone the current RN without changing any SS parameter.

C. If RN.SS = LBUS, this signifies that the transmission scenario is unique and the current resource node is used to generate strictly the lower bounds of response times. Therefore we terminate the current activity immediately (this means at $RN.EC = C_i^{\downarrow}$). The termination is achieved by setting $RN.EC = 0$ and $RN.CA = -1$.

D. Otherwise, when RN.SS = HBUS or (SS = NUS while AL is empty), we go to stage 5.

These modifications in the previous procedure are illustrated by the pseudo code illustrated in Fig. 13. The function No_context_change realizes the anticipated prevision mechanism (§3.4). It is implemented in a recursive manner. It takes the following parameters as input: current time instant and the upper bound of the busy period. It returns a Boolean value; **yes** when no change in the context and **no** in the contrary case.



```
-- Stage 4 :
   While (RN ≠ null) {
       If (RN.CA = -1)  { RN.CA := HPA; }   // AN.Ind := RN.CA
       RN.EC++;
       $i$ = RN.CA;
       If (RN.EC = $C_i^{\uparrow}$) {
            $R_i^{\downarrow}$ := min ($R_i^{\downarrow}$, AN.EOE) ;
            $R_i^{\uparrow}$ := Max ($R_i^{\uparrow}$, AN.EOE) ;
            RN.CA := -1; RN.EC := 0;
       }
-- 4) :
        Else If ($C_i^{\downarrow}$ ≤ RN.EC) {
            If (RN.EC = $C_i^{\downarrow}$) { $R_i^{\downarrow}$ := min ($R_i^{\downarrow}$, AN.EOE) }
            If ((RN.SS = NUS) && (AL ≠ $\phi$)) : {
                Clone (RN);
                If (No_context_change) {RN.SS := HBUS ; New_RN.SS :=LBUS; }
            }
            Else if ((RN.SS = LBUS) && (RN.EC = $C_i^{\downarrow}$)) {RN.CA :=-1 ; RN.EC :=0;}
       }
       RN := RN.Oth;
   }
-- Stage 5 :
```

Figure 13: Stage 4 of the pseudo code after optimization

5.3. Response time extraction

Our method permits to extract dynamically response time bounds without the memorization of results related to the created RNs. Extraction of absolute bounds ($R_i^{\downarrow}$ and $R_i^{\uparrow}$) of the message m_i from the bounds of instances related to the message the achieved as:

$$\begin{cases} R_i^{\downarrow} = min\{R_i^{m\downarrow} - (m-1) * T_i\} \\ R_i^{\uparrow} = max\{R_i^{m\uparrow} - (m-1) * T_i\} \end{cases} \quad , \; for \; m = 1...k_i. \tag{5}$$

where $k_i = \left\lfloor \frac{SP}{T_i} \right\rfloor$.

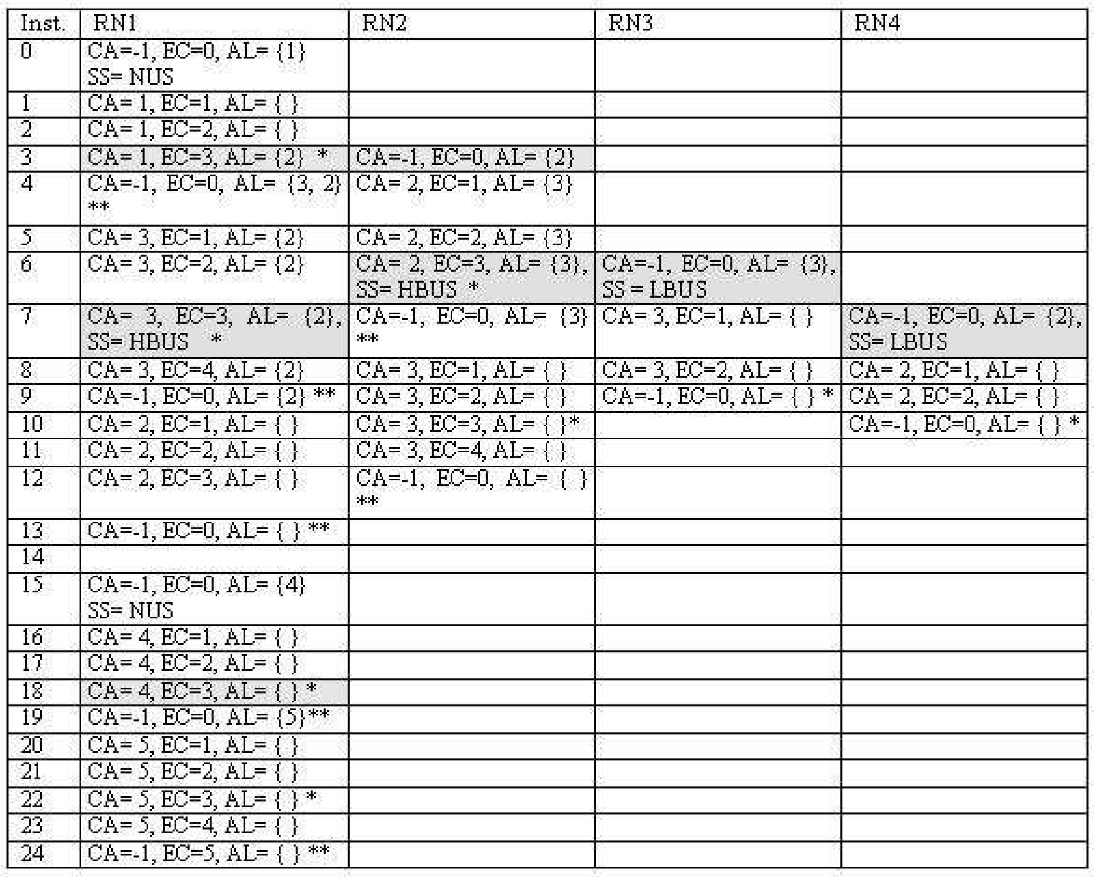
6. Case studies

To be familiar with our developed computational algorithm, we consider two examples. As a first example we show a simple computation model in details, whereas for the second example we show only results without schedule map.

6.1. Example 1

We study a simple system composed of the three messages which have the deployed instances shown in Fig. 4. As shown in Tab. 2, the deploying leads to five instants. At the first activation instant (instant 0), the resource node RN1 will be created. The evolution in the bus state is illustrated by the parameters of $RN1, \cdots, RN4$ as shown in Tab. 3.

The shadowed cells represent instants at which new activity nods are created. The single star notation



Inst.	RN1	RN2	RN3	RN4
0	CA=-1, EC=0, AL= {1} SS= NUS			
1	CA= 1, EC=1, AL= { }			
2	CA= 1, EC=2, AL= { }			
3	CA= 1, EC=3, AL= {2} *	CA=-1, EC=0, AL= {2}		
4	CA=-1, EC=0, AL= {3, 2} **	CA= 2, EC=1, AL= {3}		
5	CA= 3, EC=1, AL= {2}	CA= 2, EC=2, AL= {3}		
6	CA= 3, EC=2, AL= {2}	CA= 2, EC=3, AL= {3}, SS= HBUS *	CA=-1, EC=0, AL= {3}, SS = LBUS	
7	CA= 3, EC=3, AL= {2}, SS= HBUS *	CA=-1, EC=0, AL= {3} **	CA= 3, EC=1, AL= { }	CA=-1, EC=0, AL= {2}, SS= LBUS
8	CA= 3, EC=4, AL= {2}	CA= 3, EC=1, AL= { }	CA= 3, EC=2, AL= { }	CA= 2, EC=1, AL= { }
9	CA=-1, EC=0, AL= {2} **	CA= 3, EC=2, AL= { }	CA=-1, EC=0, AL= { } *	CA= 2, EC=2, AL= { }
10	CA= 2, EC=1, AL= { }	CA= 3, EC=3, AL= { }*		CA=-1, EC=0, AL= { } *
11	CA= 2, EC=2, AL= { }	CA= 3, EC=4, AL= { }		
12	CA= 2, EC=3, AL= { }	CA=-1, EC=0, AL= { } **		
13	CA=-1, EC=0, AL= { } **			
14				
15	CA=-1, EC=0, AL= {4} SS= NUS			
16	CA= 4, EC=1, AL= { }			
17	CA= 4, EC=2, AL= { }			
18	CA= 4, EC=3, AL= { } *			
19	CA=-1, EC=0, AL= {5}**			
20	CA= 5, EC=1, AL= { }			
21	CA= 5, EC=2, AL= { }			
22	CA= 5, EC=3, AL= { } *			
23	CA= 5, EC=4, AL= { }			
24	CA=-1, EC=5, AL= { } **			

Table 3: RNs parameters along the study period

(*) is used to signify the time instant at which the value AN.EOE is memorized as $R^{\downarrow}$, While the double star is used to memorize $R^{\uparrow} = AN.EOE$.

6.1.1. Graphical submission structure

Now we illustrate the scheduling map graphically. This map is based on resource node data structures. Applying the algorithm described in the previous section, we sweep all time instants from 0 to 30 effectuating the submission process. Submission includes the cloning of RNs and the extraction of response time bounds. This process is illustrated by the graphical structure shown in Fig. 14.

We can follow the evolution in the first created node RN1 until the termination of the last submitted activity (having the index 5). Nodes RN1 and RN2 serve in the calculation of the upper response time bounds for the instances 1, 2 and 3. While the nodes RN3 and RN4 serve in the calculation of the lower response time bounds for the same instances. RN1' serves in the extraction of both bounds for instances 4 and 5.

6.1.2. Response time results

Response time bounds for the deployed instances and the final response times for the three messages are shown in the response time (Tables 4-a and -b).

6.2. Example 2

Now we consider a more sophisticated system composed of the 12 standard messages [6]. All messages are supposed to be released synchronously. Their parameters are shown in Tab. 5-a. As we can remark that task periods are not multiples of each other, so the scheduling $SP = 420xT1 = 1,050000$. The extended set X contains 2267 instances and (169) resource nodes are created. To be used 67,227 times. The calculated response time bounds are shown in the last two columns of Tab. 5-b. Computations are done within 0.8 seconds (on PC with 1.7GHz CPU) and 17.9 MB of RAM is used.

#	Instance	$R^{\downarrow}$	$R^{\uparrow}$
1	m_1^1	3	4
2	m_3^1	6	13
3	m_2^1	7	12
4	m_1^2	18	19
5	m_2^2	22	24

a) instances

m_i	$R^{\downarrow}$	$R^{\uparrow}$
m_1	3	4
m_2	7	12
m_3	6	13

b) messages

Table 4: Response times

The column lg_lp contains the length of the longest lower priority message. This length is calculated as: $lg\_lp_i = Max\{\forall k \in lp(i)\}$. Applying the analytical method used by Tindell, we obtain the worst case response times results shown in the last column of the Tab. 5-b. The difference between each upper response time bound calculated by our method ($R_i^{\uparrow}$) and that (WCET) calculated by Tindell equal always the blocking time minus one unity ($lg\_lp_i - 1$). This means that no blockage by a lower priority message will be produced along the scheduling period and no priority inversion will happen.

m_i	π_i	b_i	$C_i^{\downarrow}$	$C_i^{\uparrow}$	T_i
m_1	1	8	111	135	2500
m_2	2	3	71	85	3500
m_3	5	3	71	85	5000
m_4	3	2	63	75	3750
m_5	6	5	87	105	5000
m_6	8	5	87	105	10000
m_7	4	4	79	95	3750
m_8	9	5	87	105	12500
m_9	7	4	79	95	5000
m_{10}	11	7	103	125	25000
m_{11}	10	5	87	105	12500
m_{12}	12	1	55	65	25000

a) parameters

m_i	$R_i^{\downarrow}$	$R_i^{\uparrow}$	lg_lp_i	*WCRT*
m_1	111	135	125	259
m_2	71	220	125	344
m_3	182	475	125	599
m_4	63	295	125	419
m_5	269	580	125	704
m_6	435	780	125	904
m_7	142	390	125	514
m_8	198	885	125	1009
m_9	348	675	125	799
m_{10}	625	1115	65	1179
m_{11}	285	990	125	1114
m_{12}	680	1180	0	1180

b) response times

Table 5: 12 CAN messages

7. Conclusions

The reliability of a CAN model depends on its ability to provide a good conception about absolute limits of schedulability behavior during the hyper duration that symbolizes the eternal life of the system. Focusing on the impact of the incertitude in stuff bits, we have introduced a general computational method able to generate and compute response-times for all possible transmission combinations of CAN messages. To reduce the computational resources when aiming to calculate absolute response-time bounds, we applied the acyclic graph technique. Then we investigated the direct impact of stuffing result on these bounds. By this, we have shown that the worst and the best case scenarios that have been discussed are different from the case of non-common messages releasing.

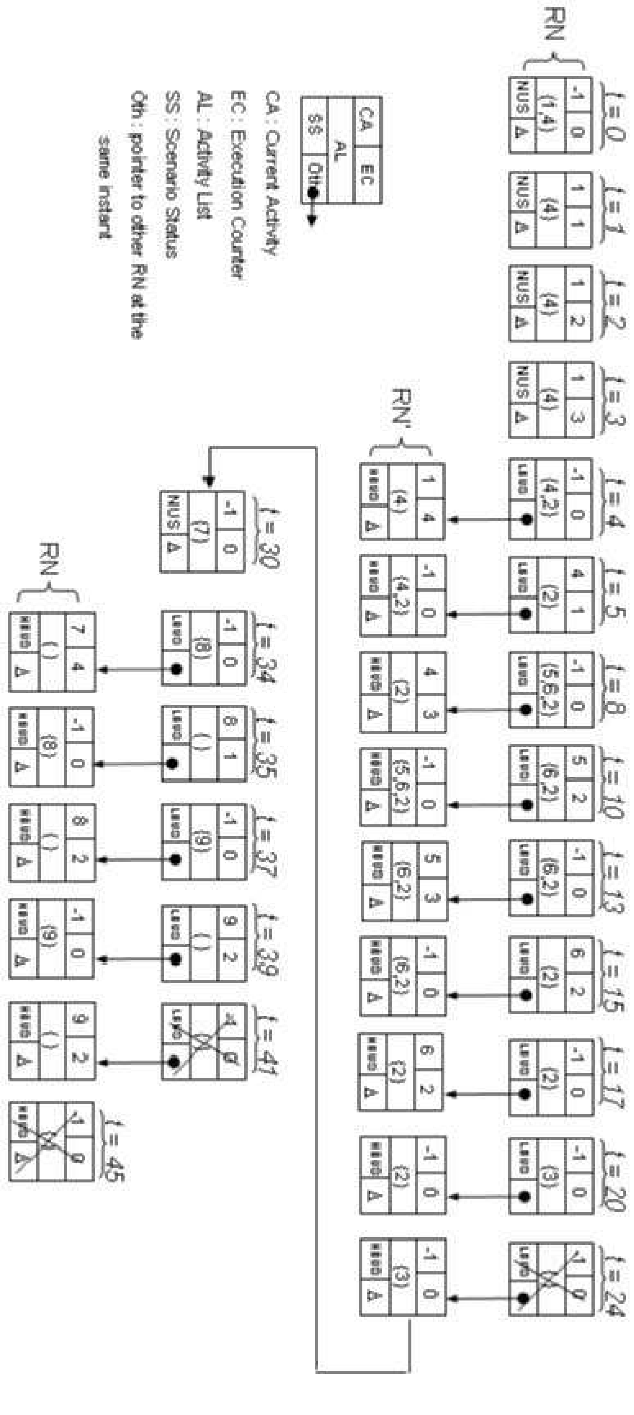


Figure 14: Scheduling structure illustrating evolution in resource nodes

References

[1] I. Alzeer. Analyse exhaustive du comportement temporel de tâches et messages temps réel. *Thèse de doctorat*, Université de Nantes, France, Décembre 2004.

[2] I. Alzeer, P. Molinaro, and Y. Trinquet. Calcul exhaustif du temps de réponse de tâches et messages dans un système temps réel réparti. *Proc. of 12th Real-Time Systems Conference*, RTS'05, Paris, April 2005, pp. 304-322.

[3] I. Alzeer, P. Molinaro, and Y. Trinquet. Response time calculations for non-preemptive tasks with variable execution time. *ETFA'2003, 19th IEEE Conference On Emerging Technologies And Factory Automation*, Lisbon, Portugal, Sept. 2003, pp. 131-136.

[4] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5), Sept. 1993, pp. 284-292.

[5] R. Bosch. CAN specifications Version 2.0, http://www.can.bosch.com/. *BOSCH, Stuttgart*, 1991.

[6] P. Castelpietra, Y. Song, F. Simonot, and O. Cayrol. Performance evaluation of a multiple networked in-vehicle embedded architecture. *Proc. of IEEE International Workshop on Factory Communication Systems*, Sep. 2000, pp. 187-194.

[7] M. Colnaric, D. Verber, R. Gumzej, and W. Halang. Implementation of hard real-time embedded control systems. *Real-Time Systems*, 14, May 1998, pp. 293-310.

[8] R. Diestel. Graph Theory. Book: Electronic ed. 2000. *ftp://ftp.math.uni-hamburg.de/pub/unihh/math/books/diestel/GraphTheoryII.pdf*.

[9] E. Grolleau, A. Choquet-Geniet, and F. Cottet. Cyclicité des séquences d'ordonnancement au plus tôt des systèmes de tâches temps réel à contraintes strictes. *Rapport de Recherche, LISI-ENSMA*, 1997.

[10] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *Proc. 11th IEEE Real-Time Systems Symposium*, Lake Buena Vista, FL, USA, Dec. 1990, pp. 201-209.

[11] J. Leung and M. Merrill. A note on preemptive scheduling of periodic real-time tasks. *Information processing letter*, 11(3), 1980, pp. 115-118.

[12] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973, pp. 46-61.

[13] T. Nolte, H. Hansson, C. Norstrom, and S. Punnekkat. Using Bit-Stuffing distribution in CAN Analysis. *IEEE Real-Time Embedded Systems Workshop*, Dec, 2001.

[14] R. Siegl and J. R. Howell. http://www.can-cia.de, CAN in Automation. *CAN specifications 2.0. Part-A and Part-B*.

[15] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analyzing fixed priority hard real time tasks. *Real-Time Systems*, 6, 1994, pp. 133-151.