Probabilistic Schedulability Guarantees for Dependable Real-time Systems under Error Bursts *

Hüseyin Aysan[1], Radu Dobrin[1], Sasikumar Punnekkat[1], and Rolf Johansson[2]
[1]Mälardalen University, Västerås, Sweden
[2]SP Technical Research Institute of Sweden, Borås, Sweden
{huseyin.aysan, radu.dobrin, sasikumar.punnekkat}@mdh.se, rolf.johansson@sp.se

Abstract

The fundamental requirement for the design of effective and efficient fault-tolerance mechanisms in dependable real-time systems is a realistic and applicable model of potential faults, their manifestations and consequences. Fault and error models also need to be evolved based on the characteristics of the operational environments or even based on technological advances. In this paper we propose a probabilistic burst error model in lieu of the commonly used simplistic fault assumptions in the context of processor scheduling. We present a novel schedulability analysis that accounts for the worst case interference caused by error bursts on the response times of tasks scheduled under the fixed priority scheduling (FPS) policy. Further, we describe a methodology for the calculation of probabilistic schedulability guarantees as a weighted sum of the conditional probabilities of schedulability under specified error burst characteristics. Finally, we identify potential sources of pessimism in the worst case response time calculations and discuss potential means for circumventing these issues.

1 Introduction

Ubiquitous deployment of embedded systems is having a great impact on our society since they interact and control our lives in many critical real-time applications. Typically those embedded systems used in safety or mission critical applications (e.g., aerospace, avionics, automotive or nuclear domains) have the design objective to maintain the properties of correctness and timeliness even under error occurrences. They are characterized by high dependability requirements, where fault tolerance techniques play a crucial role towards achieving them. The fundamental requirement for the design of effective and efficient fault-tolerance mechanisms is a realistic and applicable model of potential faults, their manifestations and consequences. These systems typically work in harsh environments where they are exposed to frequent transient faults such as power supply jitter, network noise and radiation. Fault and error models also need to be evolved based on the changes in the environments of usage or even based on technological advances. For example, nano-level shrinking of electronic devices are making them highly susceptible to transient errors and a recent study [14] has shown that even significantly low individual gate error probabilities could produce many-fold higher output error probabilities. Though single event upsets (SEU) traditionally were a concern only in memory devices, increased clock frequencies also increases the chance of a transient pulse getting latched thus affecting the logic parts as well. Increased sensitivity to noises results in an unacceptably large number of soft-errors and timing faults and design of appropriate fault-tolerant techniques and architectures have become a recent research focus in the nano-electronics community [21].

Due to the strict timeliness requirements in many safety or mission critical systems, real-time schedulability analysis techniques such as fixed priority preemptive scheduling have been increasingly used during their design. In order to provide real-time guarantees for fault tolerant systems, it is necessary to take into account an appropriate fault hypothesis, as no system can cope with an arbitrary number of faults over a bounded time interval. The majority of the previous research works assumed a worst case error distribution, e.g., single faults with a minimum inter-arrival time equal or greater than the largest period in the task set, or schedulability-centric approaches based on fault assumptions modeled as stochastic events [6]. However, once an error occurs, it is likely that the fault causing this error will be in effect for a certain duration and will cause a burst of errors rather than a single error during that period.

As the errors in a burst are caused typically by a single fault source, their probability of occurrence needs to be

*This work was partially supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS.

modeled differently than the errors caused by independent faults. This probability depends on several factors, such as the type and the severity of the fault, the resistance of the hardware to the fault, and the reaction of the fault detection and fault tolerance mechanisms to the fault. Furthermore, the error bursts can have different durations due to various reasons. For example, if we imagine a vehicle as our system under observation, which passes through a field with strong electromagnetic interference (EMI), the duration of the exposure to this fault is related to the area of this field as well as the velocity of the vehicle. Though there exist research works addressing error bursts in the context of computer networks, they do not explicitly consider the complex effects of error bursts and their impacts on real-time tasks. Accounting the duration of errors and their consequences are equally relevant from the processor scheduling perspective as well and to the best of our knowledge, the proposed work is first of its kind in this direction. In the context of real-time systems which traditionally follow a read-compute-write semantics, burst errors occurring in networks (connecting the controller to the sensors and actuators) during read or write phase could potentially behave and affect the system as burst errors in the tasks.

In this paper, we introduce a novel probabilistic burst error model and propose the associated schedulability analysis for real-time tasks scheduled under the fixed priority scheduling (FPS) policy. In particular, we are interested in the probabilities of the tasks meeting their deadlines based on the error rate assumptions. Due to the stochastic nature of the error occurrences as well as the complex effects due to the variations in the error parameters, we propose an approach that combines schedulability analysis with sensitivity analysis to provide probabilistic schedulability guarantees for the real-time task sets.

The rest of the paper is organized as following: in Sections 2 and 3, we introduce the task and the error model respectively, followed by the proposed methodology in Section 4 where we present the schedulability analysis. In Section 5 we illustrate our methodology with an example. The sources of pessimism in our analysis together with indicators for potential solutions are discussed in Section 6. In Section 7, the related works are briefly described followed by Section 8 which concludes the paper.

2 Real-time task model

We assume a sporadic task set, $\Gamma = \{\tau_1, \tau_2, .., \tau_N\}$, scheduled by the preemptive FPS paradigm where each task represents a real-time thread of execution. Each task τ_i has a minimum inter-arrival time T_i, a known worst-case execution time (WCET) C_i, a deadline D_i and a priority P_i. We assume a single processor platform and that the tasks have deadlines equal to or less than their minimum inter-arrival times.

We assume that, upon a task failure, each task τ_i executes an alternate task τ_i^{alt} with a worst-case execution time C_i^{alt} less than or equal to the original worst-case execution time of its primary C_i, a deadline equal to the original deadline D_i and a minimum inter-arrival time equal to the original minimum inter-arrival time T_i. This alternate can typically be a re-execution of the same task, a recovery block, an exception handler or an alternate with imprecise computations. We assume that each task failure is detected before the completion of the failed task instance. Although somewhat pessimistic, this assumption is realistic since in many implementations, errors are detected by acceptance tests which are executed at the end of task execution or by watchdog timers that interrupt the task once it has exhausted its budgeted worst case execution time. In case of tasks communicating via shared resources, we assume that an acceptance test is executed before passing an output value to another task to avoid error propagations and subsequent domino effects.

3 Fault and error model

We assume that the main sources of errors are the EMI and the transient hardware faults that affect, e.g. the sensors and the network systems. Examples to these errors are incorrect input values from sensors or failure in delivering the output values via network messages. Errors are detected at the end of task executions by observing, e.g., the out of range output values or omitted outputs, missing acknowledgements in case the outputs are transmitted as network messages. The error detection mechanisms we have considered are usage of sanity checks, range checks, checksums (for network messages) for the value correctness and the usage of watchdog timers for the time correctness. We assume that the watchdog timers are implemented as simple hardware that run in parallel with the tasks and interrupt in case of detected errors and the overhead of the value error detectors are included in the task worst case execution times. We further assume that a single error do not propagate into several tasks by the design of fault containment regions, such that, for instance, an input signal from a sensor is not allowed to be shared by several tasks.

Every *fault* is characterized by three parameters:

1. Duration: Faults affect systems for certain durations as in the vehicle example which passes through a field with strong EMI. The factors affecting the duration in this example is the speed of the vehicle and the area of the field under EMI. While an individual error has no "duration", in case a fault materializes in more than one error, we obtain an *error burst* with a length l between the first and the last error, bounded by the *duration* of that particular fault. If the fault materializes

in only one error, it is a *single error* with no length. However, for the sake of presentation, we use the term *error burst* for both error bursts and single errors with lengths $l \geq 0$.

The duration of the faults is very much domain specific, and in this paper, we assume that the information regarding the probability distribution of the fault durations (in the form of error burst lengths) is available.

2. Intensity: The *intensity* of a fault, determines the likelihood of causing errors during its presence. Hence it is directly proportional to the minimum inter-arrival time between two errors within a burst.

3. Rate: The *rate* of a fault determines the minimum inter-arrival time between two *independent errors* in the form of either error bursts or single errors.

   The number of fault events in a unit time is denoted by λ which not only depends on the system but also on the type of environment. For a given system, the common values for λ range from 10^2 errors per hour in aggressive environments to 10^{-2} errors per hour in lab conditions as presented by Ferreira et al. [9] and Rufino et al. [22].

Our *error model* consists of the following three parameters as illustrated in Figure 1, where A denotes the primary execution of task A, and A^{alt} denotes one of its alternate executions under sporadic arrivals of an error burst β:

1. T_E**:** The minimum inter-arrival time between error bursts.

2. T_E^{burst}**:** The minimum inter-arrival time between errors *within* a burst. In this paper we assume $T_E^{burst} = 0$. This implies that any task instance scheduled even partially under the error burst will be considered as failed.

3. $f(l)$**:** The probability mass function for the error burst length l. This function gives the probability that an error burst length is equal to a specified value of l for potential values from the range of l.

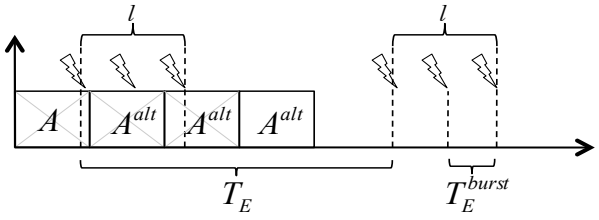


Figure 1. FT execution under error bursts

4 Methodology

Our research goal is to find $Pr(S)$, the probability that the given task set is schedulable. This probability is dependent on the parameters of the error burst (T_E and l) as well as the conditional probability that the tasks set is schedulable under a specific set of values of these parameters. Considering the interplay between T_E and l, we plan to perform a set of sensitivity analyses to derive the minimum inter-arrival times between error bursts (T_E) for each discrete l value. The burst lengths and the corresponding T_E values will then be used to find the probability of schedulability for each burst length l. Finally the schedulability of the system will be computed as a cumulative sum of these individual conditional probabilities. The steps involved in the methodology are illustrated in Figure 2 and briefly described below.

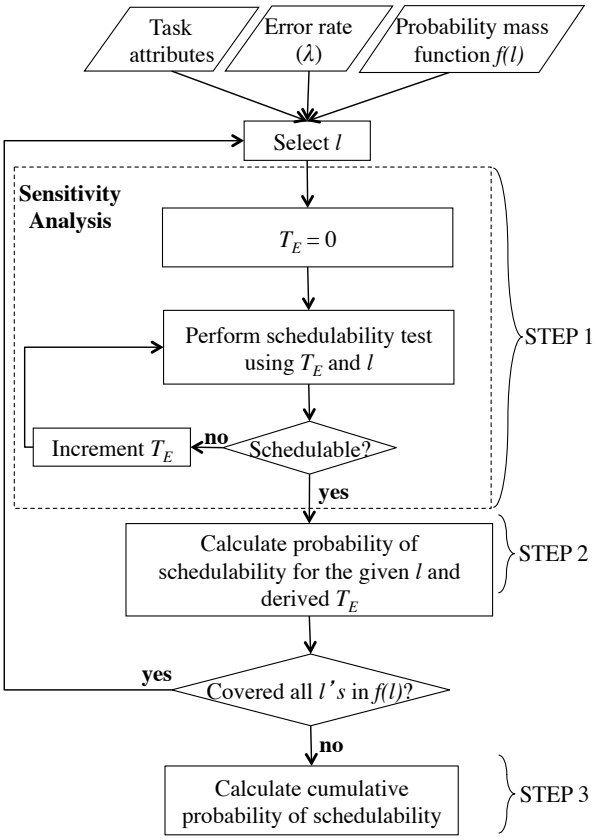


Figure 2. Methodology overview

STEP 1: Sensitivity analyses: In this step, a series of sensitivity analyses are performed for each discrete l in the probability mass function $f(l)$ in order to derive the minimum inter-arrival time between error bursts (T_E) that renders the taskset schedulable. The new schedu-

lability analysis proposed in this paper is the main tool for performing these sensitivity analyses.

STEP 2: Probability calculation of the validity of the minimum inter-arrival times between error bursts: The goal of this step is to derive the probability that the actual inter-arrival times between bursts will not be shorter the previously calculated minimum inter-arrival times. It involves the usage of previously proposed statistical approaches to find the probability of the bursts occurring with inter-arrival times larger than or equal to the T_E by taking into account λ and the mission time L. The mission time (or lifetime) L of a system varies largely depending on the domain, typically ranging from minutes for a car to take a short trip to years for a satellite to complete its mission.

STEP 3: Calculation of the cumulative probability of schedulability: Finally, based on the probability mass function $f(l)$ as well as the derived probabilities for each discrete l, we derive the cumulative probability of schedulability.

4.1 Worst case response time analysis under error bursts

In this section, we propose a worst case response time analysis that identifies whether a given task set is schedulable when affected by error bursts with a specified length l and a minimum inter-arrival time T_E. One should note that if the burst length is greater than or equal to the minimum inter-arrival time between bursts, every burst can start before the end of the previous one, hence the error bursts can potentially affect the whole mission. If this is the case, or if the burst length is greater than the minimum inter-arrival time of the task whose worst case response time is to be calculated, schedulability of this task cannot be guaranteed under the assumption $T_E^{burst} = 0$.

The traditional response time analysis calculates the worst-case response time R_i for each task τ_i using the following equation assuming that there are no task failures and, hence, no recovery attempts [12]:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \qquad (1)$$

where $hp(i)$ is the set of higher priority tasks than task τ_i, B_i is the maximum blocking time caused by the concurrency protocols used for accessing the shared resources.

The following recurrence relation is used for solving Equation 1:

$$r_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil C_j \qquad (2)$$

where r_i^0 is assigned the initial value of C_i. r^n is a monotonically non-decreasing function of n and when r_i^{n+1} becomes equal to r_i^n then this value is the worst-case response time R_i for task τ_i. If the worst-case response time R_i becomes greater than the deadline D_i, then the task cannot be guaranteed to meet its deadline, and the task set is therefore unschedulable.

If we assume an FT scheduler where the failed tasks are re-executed, then the execution of task τ_i will be affected by both errors as well as the execution of the higher priority tasks. Based on this assumption, the worst-case response times are computed [5] by using the following equation:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \left\lceil \frac{R_i}{T_E} \right\rceil \max_{k \in hep(i)} (F_k) \quad (3)$$

where F_k is the extra computation time needed by task τ_k, T_E is a known minimum inter-arrival time between errors, and $hep(i)$ is the set of tasks with priority equal to or higher than the priority of task τ_i ($hep(i) = hp(i) \cup \tau_i$). The last term calculates the worst-case interference arising from the recovery attempts. This equation is also solved by a recurrence relation as in the previous case. If all R_i values are less than or equal to the corresponding D_i values, then the task set is guaranteed to be scheduled under the condition that no two errors occur closer than the T_E value.

The main differences between the error characteristics in the traditional single error model and our proposed burst model are:

- An error burst contains multiple errors within itself
- An error burst can affect multiple tasks

Hence, the worst case scenario required for calculating the worst case response times is not the same in case of error bursts as compared to the model introduced in [6].

Definition 1. *We define the **worst-case error overhead** E_i for a task τ_i caused by an error burst β as the largest amount of time required to recover from the effects of the burst.*

Theorem 4.1. *The worst case error overhead for task τ_i caused by error burst β of length l is:*

$$E_i = \max\Big(\max_{k \in hep(i)} (2C_k^{alt} + l - \epsilon), \sum_{k \in hep(i) - \{\tau_h\}} C_k^{alt} + \max(bC_h^{alt} + C_h^{alt} - C_h + l - \epsilon, C_h^{alt})\Big) \qquad (4)$$

where τ_h is the highest priority task in the task set Γ, ϵ is an arbitrarily small positive real number, and b is a boolean variable indicating whether the burst ends before the highest priority task τ_h completes its execution.

$$b = \begin{cases} 0, & if\ l \leq C_h + \epsilon \\ 1, & otherwise \end{cases}$$

Proof. An error burst β of length l can either affect one task instance alone, or together with a set of task instances preempting it. Note that a single burst cannot affect several task instances executing non-preemptively in a sequence, since the first instance needs to recover from the burst (i.e., the burst needs to end) before the next in the sequence can start its execution. Hence we have two cases:

- Case 1: The burst hits only one task during its length l. Here we have 2 scenarios: the burst either affects τ_i directly, or a higher priority task instance from $hep(i)$ that delays the execution of τ_i. In both the scenarios, the worst case occurs when the burst starts just prior to the completion of the affected task instance (say a small ϵ before), and ends right after the start of the execution of one of its alternates. The scenario is illustrated in Figure 3 where the sum of the computation requirements of all failed alternates except last failed one equals $l - \epsilon$. Hence, $E_i = \max_{k \in hep(i)} 2C_k^{alt} + l - \epsilon$ (in Figure 3, $\epsilon = \epsilon_1 + \epsilon_2$).

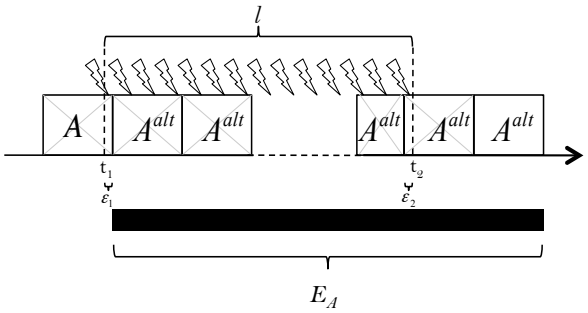


Figure 3. Worst case error overhead when the burst hits only one task

- Case 2: The burst hits multiple tasks during its length l. In this case, the only possible scenario is when the burst hits while higher priority tasks preempt lower priority ones, thus released in increasing priority order. In this case, the worst case occurs when all task instances in $hep(i)$ are involved in the preemption during the burst. Here again we have 2 scenarios depending on whether the burst ends before the highest priority task τ_h completes its execution. In the scenario where the burst ends before the highest priority task τ_h completes its execution, each task in $hep_(i)$ contributes to the worst case error overhead for task τ_i with one alternate. The scenario is illustrated in Figure 4.a where $E_i = \sum_{k \in hep(i) - \{\tau_h\}} C_k^{alt} + C_h^{alt}$.

  In the second scenario, i.e., where the highest priority task τ_h completes its execution before the burst ends, task τ_h contributes with an additional overhead consisting of a number of alternates affected by the burst. Here, the worst case occurs, similarly to Case 1, when the burst ends right after the start of the execution of one of its alternates. The scenario is illustrated in Figure 4.b where the worst case error overhead is given by $E_i = \sum_{k \in hep(i) - \{\tau_h\}} C_k^{alt} + (2C_h^{alt} - C_h + l - \epsilon)$. Hence, $E_i = \sum_{k \in hep(i) - \{\tau_h\}} C_k^{alt} + \max(bC_h^{alt} + C_h^{alt} - C_h + l - \epsilon, C_h^{alt})$ where τ_h is the highest priority task in the task set Γ (in Figure 4.a, $\epsilon = \epsilon_1 + \epsilon_2$ and in Figure 4.b, $\epsilon = \epsilon_1 + \epsilon_2 + \epsilon_3$).

□

The total interference I_i experienced by a task τ_i is the sum of the maximum interference caused by the higher priority tasks, I_i^{hp}, and the maximum interference caused by error bursts I_i^{err}.

$$\forall \tau_i \ \in \Gamma, I_i = I_i^{hp} + I_i^{err} \tag{5}$$

Note that I_i^{hp} is given by the traditional response time analysis [1, 12]:

$$I_i^{hp} = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Consequently, the worst case error interference that needs to be accounted for in the response time analysis is obtained by multiplying the maximum number of bursts that can occur during the response time of a task τ_i by its E_i. In this case, the maximum interference caused by an error burst with a minimum inter-arrival time T_E on a task $\tau_i \in \Gamma$ in the interval $(0, R_i]$ is,

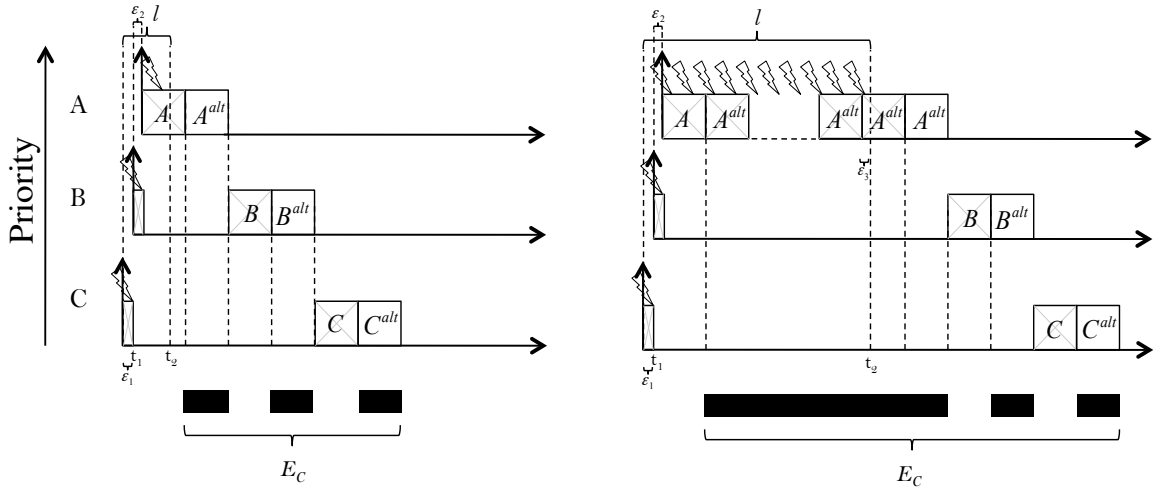
$$I_i^{err} = \left\lceil \frac{R_i}{T_E} \right\rceil E_i \tag{6}$$

Hence, the equation that gives the worst case response time for a task τ_i under error bursts is:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \left\lceil \frac{R_i}{T_E} \right\rceil E_i \tag{7}$$

4.2 Probabilistic schedulability bounds

Based on the sensitivity analyses that use the worst case response time analysis presented in the previous section (Equation 7) we obtain the minimum inter-arrival time T_E between error bursts under which the given task set is still found schedulable. Here, we make the similar assumption as in [6] that during a mission, if the actual shortest interval between two errors W is less than the derived minimum inter-arrival time of error bursts T_E, then the task set is unschedulable. Hence, the probability of unschedulability for a given l, $Pr(U \,|\, l)$, is equal to $Pr(W < T_E \,|\, l)$.



a. The burst ends before the highest priority task completes

b. The highest priority task completes before the burst ends

Figure 4. Worst case error overhead when the burst hits several tasks

Hence, our ultimate goal to find the probability of schedulability of a given task set, is translated to the derivation of the probability that, during the mission time L, no two consecutive error bursts arrive with an inter-arrival time shorter than the derived T_E.

We use a previously proposed approach [6] that uses the Poisson probability distribution, to find the probability of a number of events occurring in a fixed time period, assuming that the events occur at a constant rate (denoted by λ) and their occurrences are independent. The following approximations for the upper and the lower bounds for $Pr(W < T_E \,|\, l)$ were presented in this approach:

Upper bound: If $L/(2T_E)$ is a positive integer then

$$Pr(W < T_E \,\big|\, l) < 1 + [e^{-\lambda T_E}(1 + \lambda T_E)]^{\frac{L}{T_E}+1} - 2[e^{-2\lambda T_E}(1 + 2\lambda T_E)]^{\frac{L}{2T_E}} \quad (8)$$

Lower bound: If $L/(2T_E)$ is a positive integer then

$$Pr(W < T_E \,\big|\, l) > 1 - [e^{-\lambda T_E}(1 + \lambda T_E)]^{\frac{L}{T_E}} \quad (9)$$

Finally based on the derived probabilities of schedulability corresponding to each burst length l, $Pr(U|\, l)$, as well as the probability values for each l extracted from the probability mass function $f(l)$, we calculate the cumulative probability of the schedulability $Pr(S)$ of the given task set.

$$Pr(S) = \sum_{l} (1 - Pr(U \,|\, l)) f(l) \quad (10)$$

5 Example

We consider a single processor system on which a task set consisting of 4 tasks as shown in Table 1 is allocated. The columns P, T, C, C^{alt}, and D represent the tasks' priority, minimum inter-arrival time, worst case execution time of the primary, worst case execution time of the alternate, and relative deadline respectively. Priorities are ordered from 1 to 4 where 4 is the lowest priority. The time

$Task$	P	T	C	C^{alt}	D
A	1	30	6	4	30
B	2	40	4	4	40
C	3	40	2	2	40
D	4	100	8	4	100

Table 1. Example task set

unit is *milliseconds*. We assume a mission time of half an hour ($L = 1/2h$), and a discrete probability distribution for burst length l as shown in Figure 5. Expected number of error bursts in unit time is assumed as $\lambda = 10^0 = 1$.

We used Equation 7 to perform a sensitivity analysis for each error burst length l in the probability mass function $f(l)$ in Step 1, and found the minimum inter-arrival times between error bursts, T_E at which the task set is guaranteed to be schedulable. In Step 2, we used the statistical approach presented in Section 4.2 and derived the probabilities of unschedulability for each l and the corresponding T_E as shown in Table 2.

Finally in Step 3, we used Equation 10 and based on the

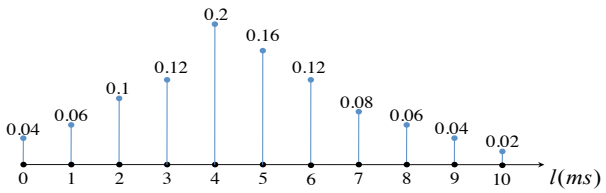


Figure 5. Probability mass function $f(l)$

l	T_E	$P(U \mid l)$
0	39	$8.1250\text{x}10^{-6}$
1	39	$8.1250\text{x}10^{-6}$
2	39	$8.1250\text{x}10^{-6}$
3	40	$8.3333\text{x}10^{-6}$
4	44	$9.1667\text{x}10^{-6}$
5	45	$9.3750\text{x}10^{-6}$
6	49	$1.0208\text{x}10^{-5}$
7	50	$1.0417\text{x}10^{-5}$
8	58	$1.2083\text{x}10^{-5}$
9	59	$1.2292\text{x}10^{-5}$
10	60	$1.2500\text{x}10^{-5}$

Table 2. Probabilities of unschedulability

derived probabilities of unschedulability, $P(U \mid l)$, as well as the probability of each l extracted from $f(l)$, we calculated the cumulative probability of schedulability as $P(S) = 1 - 8.6212 * 10^{-7} = 0.99999913788036$. This analysis showed that the example task set is schedulable with the probability 0.99999913788036 during a $1/2h$ mission where $\lambda = 1$, for the burst length characteristics given by $f(l)$.

6 Discussion

While the response time analysis based on Equation 7 introduced in Section 4.1 is sufficient, it may provide pessimistic results. In this section we discuss a number of sources of pessimism in the proposed approach, together with potential solutions towards an exact analysis.

6.1 Sources of pessimism

First source of pessimism: This is due to the assumption that *each* error burst can cause an error overhead equal to worst case error overhead E_i. However, depending on the relation between the minimum inter-arrival times of tasks and error bursts, this can be a pessimistic assumption.

We use a simple example to illustrate this case. Let our task set consist of 3 tasks, as shown in Table 3 where columns P, T, C, D represent the tasks' priority, minimum inter-arrival time, worst case execution time and deadline respectively. A lower value of P represents a higher priority. Let us also assume that $T_E = 12$ and $l = 2 + \epsilon$.

$Task$	P	T	C	C^{alt}	D
A	1	50	4	4	50
B	2	50	2	2	50
C	3	25	1	1	25

Table 3. Example Task Set 1

By using Equation 7, the worst case response time of task C in the above task set is calculated, based on its worst case error overhead $E_C = 2C_A^{alt} + l - \epsilon = 10$, as $R_C = 34$, which indicates that the task set is unschedulable. However, the actual worst case response time R_C of task C is 23 as shown in Figure 6 and hence the task set is in fact schedulable.

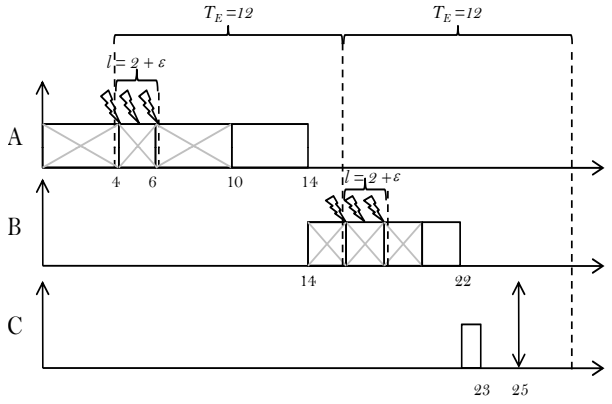


Figure 6. Exact worst case response time for task C

This scenario typically occurs when higher priority tasks that do not have outstanding computations during a burst, are in fact accounted for in the worst case error overhead E. In our example, during the actual response time $R_C = 23$ of task C, maximum two error bursts can occur. However, only one of them can cause an error overhead equal to 10, while Equation 7 accounts it two times. This is because the first instance of task A has already completed its execution before the second error burst arrives and the release of task A's second instance comes later than the end of the this burst. Hence, A has no outstanding computation during the second burst.

Second source of pessimism: This is due to the assumption that error bursts arrive with an *exact* inter-arrival time

of T_E. However, in order to generate worst case error overheads, the error bursts may need to arrive with a inter-arrival time larger than T_E to ensure non-overlap of consecutive worst case error overheads. This will imply a potential reduction in the maximum number of error bursts.

This phenomenon is exemplified by using the task set represented in Table 4. We use the same error burst characteristics as in the previous example ($T_E = 12$ and $l = 2+\epsilon$).

$Task$	P	T	C	C^{alt}	D
A	1	50	4	4	50
B	2	50	2	2	50
C	3	25	3	3	26

Table 4. Example Task Set 2

In this case, the worst case error overhead for C is calculated as $E_C = C_B^{alt} + C_C^{alt} + 2C_A^{alt} - C_A + l - \epsilon = 11$ from Equation 4. Here, again, only one error burst (instead of two) can occur during the response time of task C, that can cause an error overhead equal to 11. Figure 7 shows the execution scenario that gives the worst case response time for task C, that is $R_C = 26$.

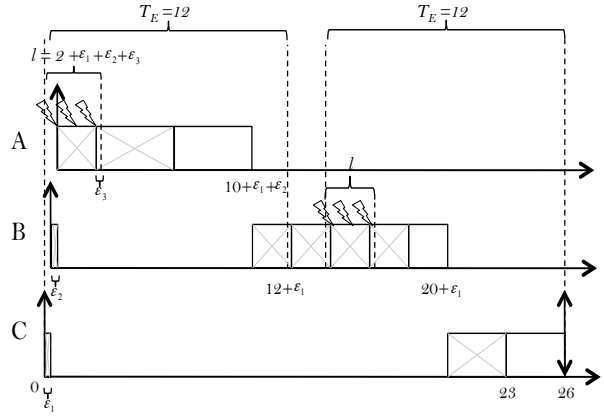


Figure 7. Worst case response time when bursts are separated by more than T_E

Note that the worst case error overhead for the second burst occurs only if the second burst arrives with an *inter-arrival time larger than* $T_E = 12$ (Figure 8 shows the scenario where the error bursts arrive with an inter-arrival time of $T_E = 12$, and the response time of task C is less than 26). In the worst case response time scenario (Figure 7), the term $\left\lceil \frac{R_i}{T_E} \right\rceil$ in Equation 6 calculates the maximum number of error burst arrivals as three, however, we can see that this is a pessimistic number as there are only two errors in the execution scenario that give the worst case error overheads for each burst.

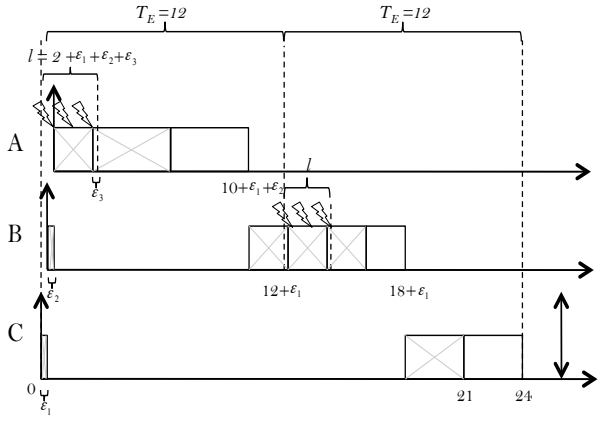


Figure 8. Shorter response time when error bursts' inter-arrival time equals T_E

6.2 Pessimism reduction

In order to address the above mentioned sources of pessimism in the response time calculations, we further refine some of the terminology and definitions used so far.

We denote the set of bursts interfering with a task τ_i, i.e., arriving after the release of τ_i, by $\{\beta^j | \, j = 1, 2, \ldots\}$. Consequently, the largest interference in terms of failed task executions due to β^j that affects the response time of τ_i, is denoted as the worst-case error overhead E_i^j.

Additionally, we denote by S_i^j the set of task instances with higher or equal priority than that of τ_i, that may be dispatched for execution during the burst β^j.

Potential solution for the first source of pessimism: By replacing the $hep(i)$ with the actual S_i^j we can eliminate this source of pessimism, as we exclude the tasks that cannot be hit by burst β^j during the response time of τ_i from the calculation of E_i^j. Hence, equation for deriving the worst case error overhead for task τ_i caused by error burst β with a length l becomes:

$$E_i^j = \max(\max_{k \in S_i^j}(2C_k^{alt} + l - \epsilon),$$

$$\sum_{k \in S_i^j - \{\tau_h\}} C_k^{alt} + \max(bC_h^{alt} + C_h^{alt} - C_h + l - \epsilon, C_h^{alt})) \quad (11)$$

where τ_h is the highest priority task in the task set S_i^j, ϵ is an arbitrarily small positive real number, and b is a boolean

variable indicating whether the burst ends before the highest priority task τ_h completes its execution.

$$b = \begin{cases} 0, & if\ l \leq C_h + \epsilon \\ 1, & otherwise \end{cases}$$

Potential solution for the second source of pessimism: This requires the identification of the worst case inter-arrival times between bursts, i.e., the burst inter-arrival times that give the worst case error overheads.

Definition 2. *We define the* ***worst-case inter-arrival time*** $T_E^{i,j}$ *between the error bursts* β^j *and* β^{j-1} *as the minimum inter-arrival time between those bursts that gives the worst case error overhead for task* τ_i *under burst* β^j.

Using this information together with the start time of the first burst interfering with τ_i ($start(\beta^1)$), one can derive the maximum number of burst occurrences, N, during a given time interval as follows.

$$N = \left\lceil \frac{R_i - \sum_{j=2}^{N}(T_E^{i,j} - T_E) - start(\beta^1)}{T_E} \right\rceil \qquad (12)$$

Combining Equations 11 and 12, the refined worst case response time of a task $\tau_i \in \Gamma$ under error bursts with minimum inter-arrival time T_E and length l, is given by the following relation:

$$R_i = C_i + B_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \sum_{j=1}^{N} E_i^j \qquad (13)$$

7 Related Work

There had been significant research efforts in fault-tolerant scheduling of real-time task sets. Pandya and Malek [17] showed that single faults with a minimum inter-arrival time of largest period in the task set can be recovered if the processor utilization is less than 0.5 under Rate Monotonic scheduling. Ramos-Thuel and Strosnider [20] used Transient Server approach to handle transient errors and investigated the spare capacity to be given to the server at each priority levels. Ghosh et al. [10] presented a method for guaranteeing that the real-time tasks will meet the deadlines under transient faults, by resorting to reserving sufficient slack in queue-based schedules. Burns et al. [5][18] provided exact schedulability analysis for fault-tolerant task sets under specified failure hypothesis and different fault tolerant strategies. Lima et al. [13] extended the uniprocessor scheduling analysis to the case of multiple faults as well as for the case of increasing the priority of a critical task's alternate upon fault occurrences. Han et al. [11] extended the *last chance strategy* described by Chetto and Chetto [8] for fixed priority preemptive scheduling. They assume an imprecise computation model, and aim to guarantee *either the primary or alternate* version of each task while trying to maximize primary executions. The majority of the previous works assumed a worst case error distribution, e.g., single faults with a minimum inter-arrival time of largest period in the task set, or schedulability-centric approaches based on fault assumptions modeled as stochastic events [6]. However, once an error occurs, it is likely that the fault causing this error will be in effect for a certain duration, will cause more errors during that period and have an adverse effect on the task response times.

Burton and Sullivan defined error bursts consisting of errors that are occurring during the period that a fault is in effect and if two successive errors within that duration does not exceed a certain maximum error-free period [7]. Ferreira et al. [9] show that 90% of the errors occurring in a network, e.g., Controller Area Network (CAN), are in the form of error bursts with an average length of $5\mu sec$ in an aggressive environment (factory conditions). However, the probability distribution of the burst length is highly dependent on the environment and more experimental studies are required in order to determine valid distributions for different domains. An example of such a study for telecommunication systems can be seen in [7]. Punnekkat et al. [19] proposed an approach to schedule real-time messages on CAN in a fault-tolerant manner using fixed priority scheduling (FPS). Navet et al. [16] proposed a probabilistic schedulability analysis for message scheduling on CAN. Later on, Broster et al. [2, 3, 4] addressed the reliability of message transmission on CAN assuming probabilistic fault models. Both Navet et al. and Broster et al. presented error burst models where they model the error bursts as a number of errors within a given time interval. However, they did not model the important characteristics of error bursts such as burst length and the error intensity within a burst, which are crucial for performing accurate schedulability analyses. Recently, Many and Doose [15] presented an error burst model and provided recovery strategies in FPS under error bursts. However their model treats the error burst as a black box, hence it is not capable of modeling the error behavior within bursts. Moreover, they assume that a task instance is hit by at most one burst during its response time.

8 Conclusions

Design of dependable real-time systems demands advances in both dependability modeling as well as scheduling theory in tandem in order provide system level guarantees that potential error scenarios are addressed in an effective as well as efficient manner. In this paper, we have introduced a burst error model together with the associated schedulability analysis for real-time systems. We presented

a sufficient analysis that accounts for the worst case interference caused by error bursts on the response times of tasks scheduled under the fixed priority scheduling (FPS) policy, which we further refined by addressing the potential sources of pessimism in the calculations. We have outlined a method to derive probabilistic scheduling guarantees from the stochastic behavior of errors by performing a joint schedulability– and sensitivity analysis.

Our ongoing research includes extending this approach to handle error probabilities that are less than 1 within an error burst, as well as consideration of multiple criticality levels of real-time tasks for efficient usage of resources.

References

[1] 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*, 8(5):284–292, September 1993.

[2] I. Broster. *Flexibility in Dependable Real-time Communication*. PhD thesis, Department of Computer Science, University of York, 2003.

[3] I. Broster, A. Burns, and G. Rodrguez-Navas. Probabilistic analysis of CAN with faults. *23rd IEEE Real-Time Systems Symposium*, pages 269–278, 2002.

[4] I. Broster, A. Burns, and G. Rodrguez-Navas. Timing analysis of real-time communication under electromagnetic interference. *Real-Time Systems*, pages 55–81, 2005.

[5] A. Burns, R. I. Davis, and S. Punnekkat. Feasibility analysis of fault-tolerant real-time task sets. *Euromicro Real-Time Systems Workshop*, 1996.

[6] A. Burns, S. Punnekkat, L. Strigini, and D. Wright. Probabilistic scheduling guarantees for fault-tolerant real-time systems. *Dependable Computing for Critical Applications 7, 1999*, pages 361–378, Nov 1999.

[7] H. Burton and D. Sullivan. Errors and error control. *Proceedings of the IEEE*, pages 1293–1301, 1972.

[8] H. Chetto and M.Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, 1989.

[9] J. Ferreira. An experiment to assess bit error rate in can. *3rd International Workshop of Real-Time Networks*, pages 15–18, 2004.

[10] S. Ghosh, R. Melhem, and D. Mosse. Enhancing real-time schedules to tolerate transient faults. *IEEE Real-Time Systems Symposium*, 1995.

[11] C.-C. Han, K. G. Shin, and J. Wu. A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults. *IEEE Trans. Computers*, 52(3):362–372, 2003.

[12] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal - British Computer Society*, 29(5):390–395, October 1986.

[13] G. Lima and A. Burns. An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems. *IEEE Transactions on Computers*, 52(10):1332–1346, October 2003.

[14] K. Lingasubramanian and S. Bhanja. An error model to study the behavior of transient errors in sequential circuits. *VLSI Design, International Conference on*, 0:485–490, 2009.

[15] F. Many and D. Doose. Scheduling analysis under fault bursts. *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 113–122, 2011.

[16] N. Navet, Y.-Q. Song, and F. Simonot. Worst-case deadline failure probability in real-time applications distributed over controller area network. *Journal of Systems Architecture*, pages 607–617, 2000.

[17] M. Pandya and M. Malek. Minimum achievable utilization for fault-tolerant processing of periodic tasks. *IEEE Transactions on Computers*, 47(10), 1998.

[18] S. Punnekkat, A. Burns, and R. I. Davis. Analysis of checkpointing for real-time systems. *Real-Time Systems*, 20(1):83–102, 2001.

[19] 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)*, pages 258–265, Washington DC, USA, May-June 2000. IEEE Computer Society.

[20] S. Ramos-Thuel and J. Strosnider. The transient server approach to scheduling time-critical recovery operations. In *IEEE Real-Time Systems Symposium*, pages 286–295, December 4-6 1991.

[21] W. Rao, A. Orailoglu, and R. Karri. Towards nanoelectronics processor architectures. *J. Electron. Test.*, 23:235–254, June 2007.

[22] J. Rufino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in can. *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, 1998. Digest of Papers.*, pages 150–159, 1998.