A Generalised Error Model and Schedulability Guarantees for Dependable Real-Time Systems

Hüseyin Aysan and Sasikumar Punnekkat
Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
{huseyin.aysan, sasikumar.punnekkat}@mdh.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 changes in the environments of usage or even based on technological advances. In this paper we propose a novel probabilistic burst error model in lieu of the commonly used simplistic fault assumptions. We introduce an approach to reason about real-time systems schedulability under the proposed error model in a probabilistic manner. We first present 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. Further, we identify potential sources of pessimism in the calculations and propose an algorithm that refines the results.

I. 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. These systems are often 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. 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 eg., nano-level shrinking of electronic devices are making them highly susceptible to transient errors and a recent study [1] 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 reserch focus in the nano-electronics community[2].

Traditionally, systems found in, e.g., aerospace, avionics or nuclear domains, typically employed the preemptive fixed priority scheduling (FPS) paradigm and were built with high replication and redundancy, with the objective to maintain the properties of correctness and timeliness even under error occurrences. Additionally, these systems typically work in harsh environments where they are exposed to frequent transient faults such as power supply jitter, network noise and radiation. [3] stated that, as per the published statistics, the ratio between the frequencies of transient and permanent faults is found to vary from 4 to 1000.

In order to provide real-time guarantees for fault tolerant systems, it is necessary to take into account the fault hypothesis, as no system can cope with an arbitrary number of faults over a bounded time interval. Pandya and Malek [4] 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 (RM) scheduling. Ramos-Thuel and Strosnider [5] 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. [6] 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. [7][8] provided exact schedulability analysis for fault-tolerant task sets under specified failure hypothesis and different fault tolerant strategies. Lima and Burns [9]

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. [10] extended the *last chance strategy* described by Chetto and Chetto [11] 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 [12]. 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 more errors during that period.

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 [13]. As the errors in a burst are caused by a single fault source, they will have a different probability of occurrence 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. [14] 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 to the environment and more experimental studies are required in order to determine valid distributions for different domains. An example of such a study was performed by [13] for telecommunication systems. Punnekkat et. al. [15] proosed an approach to schedule real-time messages on CAN in a fault-tolerant manner using fixed priority scheduling (FPS). Later on, Broster [16] addressed the reliability of message transmission on CAN assuming probabilistic fault models. [17] presented an approach to reduce the response time of multi-frame messages by using the Priority Inheritance Protocol. However, none ot the previous works on networks have taken explicitly into account the complex effects of error bursts.

In this paper, we introduce a novel probabilistic burst error model and propose the associated schedulability analysis for real-time tasks scheduled under the 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 multiple error parameters, we propose an approach that combines schedulability analysis with sensitivity analysis to provide probabilistic schedulability guarantees.

The rest of the paper is organized as following: in Section II we introduce the task and error model, followed by the proposed methodology in Section III where we present the schedulability analysis. Finally, section IV concludes the paper.

II. System model

A. Real-time task model

We assume a sporadic task set, $\Gamma = \{\tau_1, \tau_2, .., \tau_n\}$, scheduled by a preemptive FPS paradigm where each task represents a real-time thread of execution. Each task τ_i has a period (or 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 periods.

We assume that, upon a task failure, each task τ_i executes an alternate task with a worst-case execution time equal to the original worst-case execution time C_i and a deadline equal to the original deadline D_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.

B. Error model

Our *error model* consists of the following parameters:

1) T_E**:** The minimum inter-arrival time between independent error bursts.
2) T_E^{burst}**:** The minimum inter-arrival time between errors *within* a burst.

3) l**:** The length of the error burst.

Consequently, we obtain the following probability functions:

1) $Pr_{error}(t)$**:** The probability of an error occurrence within a time interval of length t can be calculated by using the Poisson probability distribution as described by Burns et al. [12]. Poisson distribution is a discrete probability distribution used for finding the probability of a number of events occurred in a fixed time period, assuming that the events occur at a constant rate and their occurrences are independent. In our case, the events are error occurrences, hence the error occurrence rate for transient errors is assumed to be constant. This rate (the expected number of events in a unit time as denoted by λ) 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. [14] and Rufino et al. [18].
The probability of m events during a time period of t is calculated as shown below.

$$Pr_m(t) = \frac{e^{-\lambda t}(\lambda t)^m}{m!}$$

If we assume that the event is an error, then the probability of no error during the lifetime or mission time (L) of the system is given by

$$Pr_{no\_error}(t = L) = e^{-\lambda L}$$

Thus, the probability of at least one error during L is

$$Pr_{at\_least\_one\_error}(t = L) = 1 - e^{-\lambda L}$$

Lifetime or a mission time of a system can vary largely depending on the domain, typically ranging from 1 hour for a plane to take a short trip to 15 years for a satellite to complete its lifetime.
2) $f(l)$**:** The probability mass function for the error burst length l which is a function that gives the probability that an error burst length is exactly equal to a specified value of l .
3) $Pr_{error|burst}(t)$**:** The probability of an error under an error burst during a time interval of length t which is a function of the error burst length l.

In this paper, we assume that $Pr_{error|burst} = 1$, and hence $T_E^{burst} = 0$. This will mean that any task scheduled even partially under the error burst period will be considered as failed.

III. Schedulability Analysis under Error Bursts

The schedulability analysis given in this section will show us whether the task set is schedulable under a combination of error inter-arrival time thresholds (minimum inter-arrival time of independent errors T_E and errors within a burst T_E^{burst}) and a burst length (l).

Our ultimate goal is to find the probability that the given task set is schedulable. In order to compute this probability, we need to perform a set of sensitivity analyses to derive the minimum inter-arrival times of independent errors (T_E), for each discrete l value. These T_E values and the burst lengths (l) will then be used to find the probability of schedulability for each l. Finally the cumulative probability of schedulability will be calculated using the probability mass function $f(l)$.

In the scope of this paper, we present a schedulability analysis under burst errors which is the main tool to perform the sensitivity analysis, as well as a method for calculating the probability of schedulability from T_E, λ and l values. The response-time calculations will differ in the following scenarios:

1) $l \geq T_E$: The burst length is greater than or equal to the minimum inter-arrival time between bursts. In this case, every burst can start before the end of the previous one, hence affecting the duration of the whole mission. Therefore, the analysis assumes that no task can successfully complete before its deadline.
2) $l < T_E \ and \ \ l > T_i - C_i$: The burst length is shorter than the minimum inter-arrival between bursts, but exceeds the threshold that guarantees task τ_i's completion before its deadline.
3) $l < T_E \ and \ \ l \leq T_i - C_i$: The burst length is shorter than the minimum inter-arrival time between bursts, as well as it allows task τ_i to feasibly complete before its deadline.

In Scenarios 1 and 2, the worst-case response time calculations are similar to the method we proposed in [19], except that the minimum inter-arrival time of independent errors T_E is replaced with the minimum inter-arrival time of the errors within a burst $T_E^{burst} \neq 0$. In the scope of this paper, we focus on Scenario 3, since in Scenarios 1 and 2, no task can be guaranteed to meet its deadline under the assumption $T_E^{burst} = 0$. In Scenario 3, the minimum inter-arrival times (T_E and T_E^{burst}), the burst length (l), as well as the task attributes should be taken into account in the calculations. A simple example is shown in Figure 1 where A^{pri} denotes the primary execution of task A, and similarly A^{alt} denotes one of its alternate executions.

The worst-case response time R_i for each task τ_i is computed by using the following equation assuming that

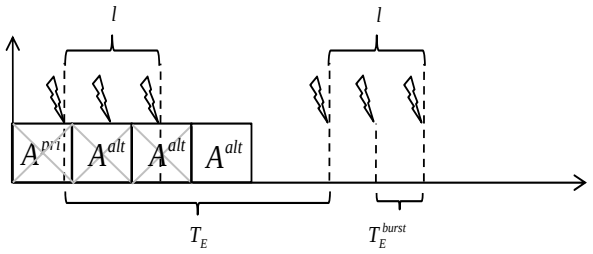

Fig. 1. FT execution under error bursts

there are no task failures and no recovery attempts [20]:

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

where $hp(\tau_i)$ 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:

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

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 errors in task τ_i or any higher priority task. Based on this assumption, the worst-case response times are computed [7] by using the following equation:

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

where F_k is the extra computation time needed by task τ_k, T_E is a known minimum inter-arrival time for errors and $hep(\tau_i)$ is the set of tasks with priority equal to or higher than the priority of task τ_i ($hep(\tau_i) = hp(\tau_i) \cup \tau_i$). The last term calculates the worst-case interference arising from the recovery attempts.

This equation can again be 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 single error model and our burst error model are:

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

Hence, the worst case scenario required for calculating the worst case response times is not the same in case of a burst errors as compared to the model introduced in [12].

We define worst-case erroneous section $WCES_i$ for task τ_i as the largest interference for task τ_i in which the effects of a burst error can stop the system from performing any 'meaningful' execution of any task. This can be viewed as the wasted time, which includes the incomplete task executions hit by an error and the burst durations.

Lemma III.1. *The worst case erroneous section for task τ_i caused by an error burst with a length l is:*

$$WCES_i = max(2 \max_{k \in hep(i)} (C_k), \sum_{j \in hep(i)} C_j) + l - \epsilon \quad (4)$$

where ϵ is an arbitrarily small positive real number.

Proof: An error burst of length l will interfere with τ_i either directly (i.e., the burst starts during τ_i's execution), or indirectly, i.e., the burst starts during the execution of a task with a higher priority than τ_i, which preempts τ_i. Hence, the task set under observation is $\{\tau_k | k \in hep(i)\}$. We have only two cases:

- Case 1: The burst affects only one task during its length. In this case, the worst case scenario occurs when the burst affects the task with the longest execution requirement among the tasks with higher or equal priority than τ_i, , i.e., $\max_{k \in hep(i)}(C_k)$, and it starts very close (ϵ) to the completion of the task, and it ends right after the start of the execution of an alternate. The scenario is illustrated in Figure 2 where the sum of the computation requirements of all alternates except last one equals $l - \epsilon$. Hence, $WCES_i = 2 \max_{k \in hep(i)}(C_k) + l - \epsilon$. (In figure 2, $\epsilon = \epsilon_1 + \epsilon_2$)

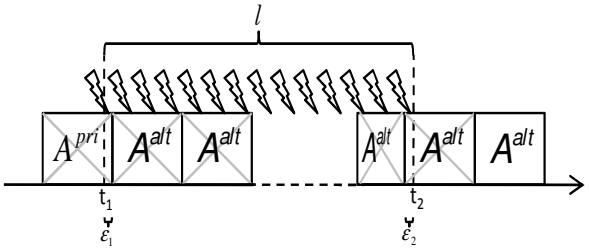

Fig. 2. Worst case erroneous section when the burst affects a single task

- Case 2: The burst affects more than one task during its length. In this case, the only possible scenario is

when the burst affects tasks preempting each other, thus executing in increasing priority order. In this case, the worst case occurs when all tasks of higher or equal priority than τ_i are involved in the preemption during the burst. The scenario is illustrated in Figure 3. Hence, $WCES_i = \sum_{j \in hep(i)} C_j + l - \epsilon$. (in figure 3, $\epsilon = \epsilon_1 + \epsilon_2$)

■

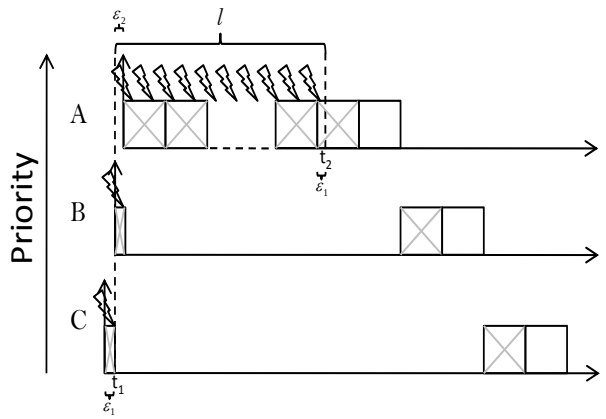

Fig. 3. Worst case erroneous section when the burst affects several tasks

The total interference I_i experienced by a task τ_i as the sum of the maximum interference caused by the higher priority tasks, I_i^{hp} and the maximum (i.e., the *longest*) 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 [21], [20]:

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

Lemma III.2. *The maximum interference caused by an error burst of length l and minimum interarrival time T_E on a task $\tau_i \in \Gamma$ in the interval $(0, R_i]$ is,*

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

Proof: In case of a single burst, the worst case interference on τ_i is given by the longest erroneous section encountered by any of the higher or equal priority tasks, i.e., $I_i^{err} = WCES_i$. In case the bursts are separated by T_E, then the maximum number of times τ_i can get hit in the interval $(0, R_i]$ is given by $\left\lceil \frac{R_i}{T_E} \right\rceil$. ■

Theorem III.3. *The worst case response time of a task $\tau_i \in \Gamma$ under error bursts with minimum interarrival time T_E and length l, is given by the relation:*

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

Proof: The proof follows directly from Lemmas III.1 and III.2 ■

While the above test is sufficient, it may provide pessimistic results, as it assumes that during each error burst, an erroneous section equal to the $WCES_i$ will contribute to the interference. However, depending on the relation between the period of the tasks and the minimum error interarrival time, this can be highly improbable. In the next section we present a refinement to the response time calculation that eliminates this pessimism.

A. Refined response time analysis under error bursts

We use a simple example throughout the description of the refined approach. Let our task set consists of 3 tasks, as shown in Table I where columns P, T, C, D represent the priority, period, worst case execution time and the deadline respectively. Priorities are ordered from 1 to 3 where 3 is the lowest priority. Let us also assume that $T_E = 12$ and $l = 2 + \epsilon$.

Task	P	T	C	D
A	1	50	4	50
B	2	50	2	50
C	3	25	1	25

TABLE I. Example Task Set 1

From Equation 7, the worst case response time of Task C in the above task set is calculated 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 4.

One source that leads to this pessimism is the relation between the task periods and the minimum interarrival time of the error bursts. In this example, during the actual response time $R_C = 23$ of Task C, maximum two error bursts can occur. However, not both of them can cause an error section equal to $WCES_C = 10$ to interfere Task C's execution. This is because period of Task A is greater than the minimum error interarrival time T_E, hence there is no execution requirement for Task A when the second burst arrives. Note that we assume that all tasks are recoverable even under the worst case error scenario if executed alone, i.e., the recovery of the Task A is assumed to be possible *before* a second error burst arrives.

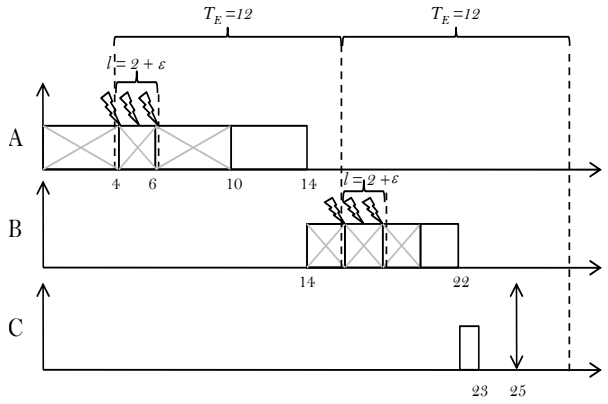

Fig. 4. Actual worst case response time for task C

Task	P	T	C	D
A	1	50	4	50
B	2	50	2	50
C	3	25	3	26

TABLE II. Example Task Set 2

Another source of the pessimism is the assumption of error bursts occurring *with* an *exact* interarrival time of T_E. Let us assume the task set shown in table II as an example to demonstrate this source of pessimism.

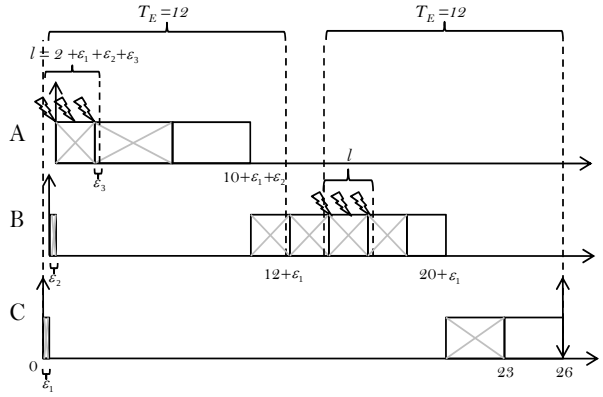

Fig. 5. Pessimism reduction in the assumption of the maximum number of bursts

The worst case erroneous section is calculated to $WCES_C = 11$ from Equation 4. Here, again, only one error burst (instead of two) can occur during the response time of Task C, and can cause an error section equal to $WCES_C = 11$ that interferes with Task C's execution. Figure 5 shows the execution scenario that gives the worst case response time for Task C, i.e., 26. Note that, in order to cause the longest possible erroneous section, the second burst must arrive with an *interarrival time larger than* $T_E = 12$ (Figure 6 shows the scenario where the error bursts arrive with an interarrival time of $T_E = 12$, and the response time of Task C is less than 26). In the worst case response time scenario (Figure 5), the term $\left\lceil \frac{R_i}{T_E} \right\rceil$ in Equation 6 calculates the maximum number of error burst arrivals to three, however, we can see that this is a pessimistic number as there are only two errors in the execution scenario that give the largest erroneous section.

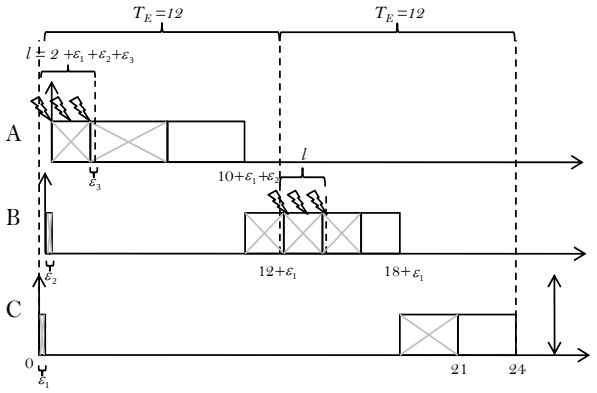

Fig. 6. Shorter response time when error bursts' interarrival time equals T_E

In this section, we present Algorithm 1, that computes the worst-case interference arising from the recovery attempts, during the execution of task τ_i for R_i, while reducing the pessimism described above.

Inputs to the algorithm are the task τ_i, the task set $hep(\tau_i)$, the response-time R_i in the current iteration of the recurrence relation, the minimum interarrival time T_E between error bursts and the assumed length l of the bursts. Set S on Line 3 is the set of tasks that can potentially be affected by errors. On Line 4, the algorithm calculates the maximum number of error bursts that can occur during the response-time R_i in the current iteration of the recurrence relation, which is the number of times that the largest erroneous sections for each burst will be added to the sum I_i^{err}. On Line 8, the algorithm adds the tasks that have been released after the previous burst to S which were previously removed from this set due to the completion of their previous instances. Line 9, computes the maximum erroneous section that can occur among the tasks within S. Line 10 removes the tasks from S that will be completed before the earliest arrival of the next error burst. If the minimum interarrival time between the $(i-1)^{th}$ and the i^{th} error burst that generates the longest erroneous section for the i^{th} error burst is greater than T_E (as in the example

Algorithm 1: I_i^{err}

input : τ_i, $hep(\tau_i)$, R_i, T_E, l
output: worst case error interference for task τ_i in response time R_i

begin
  $output \leftarrow 0$;
  $S \leftarrow hep(\tau_i)$;
  $n \leftarrow \left\lceil \frac{R_i}{T_E} \right\rceil$; // number of bursts
  $i \leftarrow 1$; // burst index
  $delays \leftarrow 0$;
  while $i <= n$ **do**
    $S \bigcup set\ of\ new\ arrivals$;
    $output \leftarrow output+$ longest error section for the i^{th} error burst;
    $S - set\ of\ completed\ tasks$;
    $T_E^i \leftarrow$ minimum interarrival time between the $(i-1)^{th}$ and the i^{th} error burst that generates the longest erroneous section for the i^{th} error burst ;
    if $T_E^i - T_E > 0$ **then**
      $delays \leftarrow delays + T_E^i - T_E$;
      if $\left\lceil \frac{R_i - delays}{T_E} \right\rceil < n$ **then**
        $n \leftarrow \left\lceil \frac{R_i - delays}{T_E} \right\rceil$;
      end
    end
    $i \leftarrow i + 1$;
  end
  return $output$;
end

in Figure 5), the algorithm checks if the number of bursts is less than the previously calculated value. If this is the case, the maximum number of error bursts is updated before going into the next loop or termination of the algorithm. Finally, the algorithm outputs the accumulated worst case error interference I_i^{err} for task τ_i in response time R_i.

B. Probabilistic schedulability bounds

In this paper, we make the similar assumption as in [12] that during a mission, if the actual shortest interval between two errors W is less than the assumed minimum interarrival time of error bursts T_E, then the task set is unschedulable. Hence, the probability of unschedulability $Pr(U)$ is equal to $Pr(W < T_E)$. By using the Poisson probability distribution, in [12], Burns et al. show the upper and lower bounds for $Pr(W < T_E)$ as shown below:

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

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

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

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

Burns et al. [12] also derived the following two useful approximations for the upper and lower bounds:

c) Approximate upper bound:: An approximate the upper bound for $Pr(W < T_E)$ as given by Equation 8 is

$$Pr(W < T_E) \lesssim \frac{3}{2}\lambda^2 L T_E \quad (10)$$

provided that λT_E, $\lambda^2 L T_E$ are small and $L >> T_E$.

d) Approximate lower bound:: An approximate lower bound for $Pr(W < T_E)$ as given by Equation 9 is

$$Pr(W < T_E) \gtrsim \frac{1}{2}\lambda^2 L T_E \quad (11)$$

provided that λT_E, $\lambda^2 L T_E$ are small.

We propose to use a similar approach to find the probability of schedulability of a given task set under burst errors. We first use the schedulability test proposed in Section III to perform a series of sensitivity analyses for each error burst length in the the probability mass function $f(l)$. These analyses give us the minimum error interarrival time T_E values for each error burst length l in the probability mass function $f(l)$. Then, by using Equation 10, we calculate the approximate probability of having no anomalies $(1 - Pr(W < T_E))$. Finally based on this probability values and the probability values for each l extracted from $f(l)$, it is straightforward to calculate the cumulative probability of the schedulability of the given task set.

IV. Conclusions

In this paper, we have introduced a burst error model together with the associated schedulability analysis for real-time tasks scheduled under FPS. We presented a sufficient test 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 with 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] K. Lingasubramanian and S. Bhanja, "An error model to study the behavior of transient errors in sequential circuits," *VLSI Design, International Conference on*, vol. 0, pp. 485–490, 2009.

[2] W. Rao, A. Orailoglu, and R. Karri, "Towards nanoelectronics processor architectures," *J. Electron. Test.*, vol. 23, pp. 235–254, June 2007. [Online]. Available: http://dx.doi.org/10.1007/s10836-006-0555-7

[3] M. Pizza, L. Strigini, A. Bondavalli, and F. D. Giandomenico, "Optimal discrimination between transient and permanent faults," *3rd IEEE International Symposium on High-Assurance Systems Engineering*, pp. 214–223, 1998.

[4] M. Pandya and M. Malek, "Minimum achievable utilization for fault-tolerant processing of periodic tasks," *IEEE Transactions on Computers*, vol. 47, no. 10, 1998.

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

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

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

[8] S. Punnekkat, A. Burns, and R. I. Davis, "Analysis of checkpointing for real-time systems." *Real-Time Systems*, vol. 20, no. 1, pp. 83–102, 2001.

[9] G. Lima and A. Burns, "An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems," *IEEE Transactions on Computers*, vol. 52, no. 10, pp. 1332–1346, October 2003.

[10] 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*, vol. 52, no. 3, pp. 362–372, 2003.

[11] 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.

[12] 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*, pp. 361–378, Nov 1999.

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

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

[15] 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.

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

[17] 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.

[18] 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.*, pp. 150–159, 1998.

[19] H. Aysan, R. Dobrin, and S. Punnekkat, "Task-level probabilistic scheduling guarantees for dependable real-time systems - a designer centric approach," *IEEE International Workshop on Object/component/service-oriented Real-time Networked Ultra-dependable Systems*, 2011.

[20] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal - British Computer Society*, vol. 29, no. 5, pp. 390–395, October 1986.

[21] 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.