# Verification of Real-Time Systems using Statistical Model Checking

Jeffery P. Hansen* Lutz Wrage* Sagar Chaki* Dionisio de Niz*
Mark Klein*
*Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA*

**Software for airborne systems have become more and more complex, yet the need to verify correct behavior remains constant. Adding to the challenge is the fact that these systems operate in uncertain environments where the workload they must process may not be known in advance. Statistical model checking is one tool that has been shown to be useful in verification of large complex systems operating under uncertainty. One challenge area for software validation in UAS systems is the evaluation of the performance of task scheduling policies. In this paper, we apply statistical model checking to characterize the timing behavior of a system using zero-slack rate-monotonic scheduler. We compare this to the timing behavior of traditional rate-monotonic scheduling. We show that task deadline requirements can impact the relative performance of the two scheduling policies.**

## I. Introduction

Verification of timing correctness in autonomous systems is a critical area of concern as systems become more complex. While theoretical frameworks for timing verification such as Rate Monotonic Analysis (RMA)[1] exist, these techniques are dependent on knowing absolute Worst Case Execution Time (WCET) values. In practice, numerous factors make knowing absolute WCET difficult or impossible. These factors include data dependent execution time, caching, memory and input-output bottlenecks, and branch prediction techniques used in modern processors. All of these factors make verification using formal techniques difficult or impossible. Unfortunately, traditional simulation-based validation methods also fall flat as they often lack confidence bounds. In addition, simulation time can be unacceptably long in order to calculate the probability of rare deadline miss events.

Statistical model checking[2] has emerged as an approach that combines the rigorous mathematical analysis of formal methods with Monte-Carlo-based simulation. It is a formalized simulation-based approach for estimating the bounded probability of failure in systems where failure is defined using a temporal logic such as Bounded Linear Temporal Logic (BLTL). Advanced simulation techniques, such as "importance sampling",[3] are applied to reduce the number of simulations needed to accurately estimate the probability that a property holds in a system. In our experiments, we show that importance sampling can reduce the simulation effort by many orders of magnitude compared to brute force simulation in estimating the probability of rare events.

To illustrate our approach, we apply statistical model checking to compare the deadline-miss performance of a Rate Monotonic (RM) scheduler with a Zero-Slack Rate Monotonic (ZSRM) scheduler in a mixed criticality system. As a model scenario, we use an autonomous UAS search/find application where tasks

*Carnegie Mellon University, Software Engineering Institute, 4500 Fifth Ave., Pittsburgh, PA 15213

such as flight control and collision avoidance are more critical than those performing surveillance, since failure of these high critical tasks can lead to catastrophic failure of the system. We demonstrate how our approach can be used to select the most appropriate scheduling algorithm based on system criticality requirements.

The rest of the paper is organized as follows. In Section II, we present background definitions and techniques. In Section III we present our model scenario. In Section IV we describe our approach. In Section V, we present our experimental results. In Section VI, we survey related work, and in Section VII, we conclude.

# II. Background

## II.A. Scheduling

Periodically executed real-time tasks are typically characterized by the following scheduling-relevant properties:

**Period** How often the task must be executed. For example, a controller may need to sense the plant, compute a correction, and perform the actuation once every 100ms. Each computation is called a job. After the job is completed, the task waits for the start of the next period.

**Release Time** The time at which the next job becomes becomes ready for execution. This is a multiple of the task's period.

**Execution Time** How long each execution takes, e.g., 10ms. Typically the worst case execution time (WCET) is used for analysis.

**Deadline** How long after the start of a period the task must be finished. This is most often the same as the task period, meaning that the task must be finished before it must execute again.

**Hyper-Period** The least common multiple of the periods of a task set. This is useful for analyses because the pattern of release times and deadlines is the same in any two hyper-periods.

We assume that tasks are independent of each other in that the processor is the only shared resource. This constraint simplifies the scheduling problem because each task can execute at any time as there is no need to wait for another task to finish some operation or release a shared resource that can only be used in a mutually exclusive manner. We also assign each task a fixed priority at design time. The jobs of the task run at this priority and are interrupted only if a higher priority job is released. At this time the low priority job is preempted to allow the higher priority job to run. The preempted job continues once the high priority job finishes. This is known as fixed-priority scheduling.

### *II.A.1. Rate Monotonic Scheduling*

In rate monotonic scheduling[1] (RMS), a task's priority is derived from the period (where, rate = 1 / period). The task with the shortest period has the highest priority (= 1), the next longer period has priority 2, etc. The scheduler always allocates the processor to the *active* (i.e., having a job ready for execution) task with the highest priority. This priority assignment is *optimal* for fixed priority scheduling when each task's deadline is equal to its period, i.e., if there exists any fixed priority assignment such that each task always meets its deadline, then the tasks are also schedulable under RMS.

While being simple, RMS has the disadvantage that if a task overruns its execution time, e.g., as the result of a fault or other exceptional condition, not only the task itself but any other task of lower priority could miss its deadline. In situations where a task's execution time varies, it is therefore necessary to provide a conservative estimate of the WCET to avoid this effect. However, the disadvantage of RMS is that if the maximum execution time is rarely used and significantly larger than the average execution time, then only few tasks can be scheduled on a single processor even though the average processor utilization is low, leading to the wastage of CPU cycles.

*II.A.2. Zero Slack Rate Monotonic Scheduling*

RMS assumes that all tasks are equally important and must meet their deadlines. This assumptions is often too restrictive because in many situations tasks can be divided into classes based on their importance or criticality. A high criticality task must always meet its deadline for the system to operate, whereas a deadline miss of a low criticality task may result in degraded system operation but not in complete failure.

In such a *mixed criticality* environment, RMS is not sufficient because a high criticality task with a long period (and therefore low RMS priority) must be protected from lower criticality tasks with shorter periods. In particular, the overrun of a low criticality task must not cause a high criticality task to miss its deadline. One example of a mixed criticality environment is a UAS where critical flight control tasks share the processor with less critical mission tasks such as surveillance.

Zero slack rate monotonic scheduling[4] (ZSRM) assigns each task a criticality level (1 is the highest criticality level; 2,3,... are lower levels). The ZSRM algorithm works with two execution times per task, a normal execution time and an overloaded execution time. The algorithm guarantees the following for any task $t$: If no task of higher criticality than $t$ overruns its normal execution time, then $t$ can execute for its overloaded execution time without missing its deadline. To achieve this guarantee, the scheduler may cancel or skip executions of lower criticality tasks to make their execution time available to higher criticality tasks. As long as all tasks stay within their normal execution time no task execution will be canceled or skipped.

## II.B. Statistical Model Checking

Statistical Model Checking (SMC) is a Monte-Carlo-based simulation approach to verifying the correct behavior of stochastic systems. Stochastic systems are those whose outcome depends on stochastic inputs to the system. The stochastic inputs typically reflect random variations in the environment in which the system operates (e.g., sensor readings, task execution times, etc.). The goal is to probabilistically characterize the correct behavior of the system (e.g., to prove that the probability of failure is no more than $10^{-9}$). While there are formal methods for analyzing stochastic systems (such as probabilistic model checking, see Section VI), many real-world systems have complexity that is beyond the capability of these approaches. For this reason, stochastic model checking has emerged as an approach that combines the formal treatment of model checking with Monte-Carlo-based simulation.

When SMC[2] was first proposed, the emphasis was on hypothesis testing. The goal was to show that a formal property of a system held with at least a specified probability. Bounds on the accuracy of the estimate would determine the number of samples required. More recently, an estimation approach has been taken in which the goal is to estimate the probability that a system property holds. In this approach, a metric such as relative error (the ratio of the standard deviation to the mean of an estimator) is used to characterize the accuracy of an estimate. The estimation based approach is more advantageous than the hypothesis testing based approach when applied to testing "rare" conditions (e.g., failures) with low probabilities of occurring. In such cases, advanced simulation techniques such as importance sampling can be applied which can drastically reduce the number of samples required to obtain a probability estimate.

Formally, the SMC problem is to estimate the probability that a system model $\mathcal{M}$ satisfies some property $\Phi$ given a joint probability distribution $f(x)$ on the model inputs $x$. The input $x$ is a vector of arbitrary length and represents any random inputs to the system (sensor readings, execution times, etc.). We typically assume the system $\mathcal{M}$ is stochastic, but deterministic. That is, we assume that for given fixed values of the input vector $x$, the behavior of the system is also fixed. We write $\mathcal{M} \models \Phi$ to mean that property $\Phi$ is satisfied in model $\mathcal{M}$, and the goal of SMC is to estimate $p = Pr[\mathcal{M} \models \Phi]$.

The property $\Phi$ can be specified in a temporal logic such as Bounded Linear Temporal Logic (BLTL). It can consist of a simple logical expression for the ending state, or it can include temporal operators indicated the order in which conditions should hold over the course of execution of a model. Since our approach is based on detection of rare events, we will assume that $\Phi$ is some undesirable or fault property of the system.

SMC estimation is performed by conducting a series of Bernoulli trials, modeling each trial as a Bernoulli random variable. A Bernoulli random variable is a random variable which can be 1 with probability $p$ or 0 with probability $1-p$. For each trial, a random vector $x$ distributed by $f(x)$ is generated. The system is then simulated to generate a trace which is then tested against the property $\Phi$. The outcome of the Bernouilli trial is either 1 if the property holds, or 0 if the property does not hold.

If we define $I_{\mathcal{M}\models\Phi}(x)$ as an indicator function that returns 1 when the model $\mathcal{M}$ under input $x$ satisfies

the property $\Phi$, and 0 when the property does not hold, then the probability $p$ can be calculated as:

$$p = \int I_{\mathcal{M} \models \Phi}(x) f(x) dx \tag{1}$$

This probability can be estimated as:

$$\hat{p} = \sum_{i=1}^{N} I_{\mathcal{M} \models \Phi}(x_i) \tag{2}$$

where $N$ is the number of trials, and where the $x_i$ are distributed according to $f(x)$. As $N$ goes to infinity, the estimator $\hat{p}$ will go to $p$. The quality of the estimate $\hat{p}$ can be quantified using a relative error calculation. Relative error is defined as the ratio of the standard deviation to the expected value of the estimator:

$$RE(\hat{p}) = \frac{\sqrt{Var(\hat{p})}}{E[\hat{p}]} \tag{3}$$

Assuming small $p$, it has been shown[5] that the relative error can be approximated by:

$$RE(\hat{p}) \approx \frac{1}{\sqrt{Np}} \tag{4}$$

which can be rearranged as:

$$N \approx \frac{1}{RE(\hat{p})^2 p} \tag{5}$$

This gives us a relation between the probability of the condition we wish to test, and the number of samples required for some target relative error. For example if we wish to estimate the probability of an event at $p = 10^{-5}$ with a relative error of 0.01, then the number of samples required would be $N \approx 10^9$.

### II.C. Importance Sampling

One problem that is immediately clear from Equation 5 is that as the probability of the event we are testing becomes smaller, the required number of simulation samples increases rapidly. This can result in an impractical number of simulations being required to accurately estimate the probability of a rare event in a system. Importance Sampling[3] is a variance reduction technique that has been applied in SMC to reduce the number of samples required to estimate the probability of rare events.[5] The reduction in simulation effort can be significant, often multiple orders of magnitude.

The key idea behind importance sampling is to first simulate the system under a different probability distribution with lower variance, and then mathematically adjust the result back to the original probability distribution. If we let $g(x)$ be the modified distribution and $f(x)$ be the original distribution, we can rewrite Equation 1 as:

$$p = \int I_{\mathcal{M} \models \Phi}(x) \frac{f(x)}{g(x)} g(x) dx \tag{6}$$

Note that original equation can be recovered by canceling the $g(x)$ as long as $g(x) \neq 0$. We can then define $W(x) = \frac{f(x)}{g(x)}$ and rewrite as:

$$p = \int I_{\mathcal{M} \models \Phi}(x) W(x) g(x) dx \tag{7}$$

which is the function $I_{\mathcal{M} \models \Phi}(x) W(x)$ integrated over the probability density function $g(x)$. The function $W(x)$ is called a weight function and provides a mapping between the original and modified distributions. The estimator for this form is:

$$\hat{p} = \sum_{i=1}^{N} I_{\mathcal{M} \models \Phi}(x_i) W(x_i) \tag{8}$$

where the $x_i$ are distributed by $g(x)$. By carefully choosing $g(x)$, the sample variance can be significantly reduced, thus reducing the number of samples required to estimate the probability of an event at a target relative error by multiple orders of magnitude.

The biggest challenge in applying importance sampling is in the choice of a good $g(x)$. The idea is to move some of the density from $f(x)$ into the fault region where $I_{\mathcal{M} \models \Phi}(x) = 1$ to make faults more likely

to occur during the simulation. It is important to choose $g(x)$ that has non-zero density everywhere where $I_{\mathcal{M}\models\Phi}(x) = 1$. Otherwise, the simulation will yield an incorrect result (note the divide by zero resulting in Equation 6 from such a choice).

One common method for selecting $g(x)$ is to choose a distribution from the same family as $f(x)$, with modified distribution parameters. For example, if $f(x)$ is the exponential distribution $f(x) = \frac{1}{s}e^{-x/s}$, then we might choose $g(x;k) = \frac{1}{sk}e^{-x/(sk)}$ where $k$ is a "tilting" parameter. The tilting parameter gives us a family of potential $g(x)$ depending on the choice of $k$. In this example, choosing $k = 1$ gives us the original distribution. Choosing larger $k$ give us a distribution with larger mean values.

The problem now is to choose the value of the tilting parameter(s) $k$ (potentially one $k$ for each dimension in $x$) that gives the best $g(x)$. This is typically done by using a two phase simulation approach. In the first phase, short probe simulations are performed to choose the best value of $k$. In the second phase, the main simulation is performed using the best value of $k$ chosen in the first phase. One approach to choosing $k$ is the cross entropy method.[5] A less complex approach is to simply conduct probe simulations over a range of $k$ values, and choose the one that results in the lowest relative error for a fixed number of trials.

## III. Scenario

The system in our experiments is a UAS that flies through an area searching for mines. The UAS is equipped with a downward facing mine sensor. This sensor periodically emits a kind of ground penetrating signal to scan an area below the UAS for buried objects. The reflected signal is analyzed in two phases. In the first phase the signal is pre-processed to detect all buried objects. Some of these may be mines whereas others could be rocks, etc. The second phase is a detailed analysis of the signal to distinguish a mine from other objects. This is repeated for each buried object.

The UAS should operate with little supervision throughout the mission, so it must be able to fly autonomously in a given area. The requires a capability to avoid obstacles in its flight path. We assume that the UAS has a forward-pointing obstacle sensor that finds potential obstacles. Like the mine sensor, the obstacle sensor periodically emits a signal to scan the area in front of the UAS for obstacles. Based on the detected obstacles the UAS must determine if it is necessary to avoid one of them by altering the flight path.

In addition to the tasks mentioned above, there is another periodic task to control the basic flight functions of the UAS. This task keeps the UAS stable and adjusts rotor speeds to execute navigation actions, e.g., to avoid obstacles.

Given this, we have four tasks to consider: (1) flight control, (2) obstacle detection, (3) obstacle avoidance, and (4) mine detection. These tasks differ in their relative importance to the success of the UAS's mission and the relative priority at which they need to be executed. The importance of a task is expressed as the task criticality whereas the the priority is determined by the tasks' rates.

**Flight control task** The flight control task is the most critical because without it the UAS would not be able to fly at all. This task executes the main control loop and is assumed to run at the highest rate in the system.

**Mine detection task** Since the purpose of the UAS mission is to detect mines, this task is the only one that directly contributes to the mission outcome. However, without flight control and obstacle detection and avoidance, mine detection cannot be successful. Also, we assume that loss of the UAS due to a collision is less acceptable than degraded mine detection performance. Therefore, we assign mine detection the lowest relative criticality of the four tasks.

Mine detection requires the detection of buried objects which in itself is a difficult task. To account for this difficulty we assume that the success rate at which a buried mine is detected during a single sensor signal emission is limited. This is compensated for by operating the sensor at a higher rate such that many sensor signals hit the mine from different angles, thus increasing the chance of detection.

**Obstacle detection and avoidance tasks** The criticality of detecting and avoiding obstacles lies between the criticality of the flight control and mine detection tasks. For the purposes of our evaluation we assign obstacle detection a higher criticality level than obstacle avoidance.

Under the assumption that the UAS forward velocity is low, we expect that the number of obstacles does not change rapidly, such that the obstacle detection task can run at a low rate. In contrast, obstacle avoidance is run at a higher rate to increase the safety of the UAS.

Real-time task scheduling is deterministic if the number of tasks and execution times are known. In our scenario, however, execution times depend on factors external to the UAS system, namely the number of obstacles detected in the flight path and the number of objects detected by the mine sensor. These numbers vary depending on the location and the orientation of the UAS. This randomness makes it difficult to apply classical analysis techniques (such as RMA) to our system. These analyses assume a fixed worst-case execution time for all tasks and analyze if all tasks will meet their deadlines under this assumption. As a result, such analyses are typically overly pessimistic.

Also, lower criticality tasks can tolerate deadline misses. If, for example, the mine detection task misses its deadline because there are too many objects to analyze in the allotted maximum execution time, it may report an incorrect result for this single execution, but this will not necessarily result in mission failure. Similarly, missing one deadline of the obstacle detection task is unlikely to result in a crash. Only several consecutive deadline misses would jeopardize the overall mission. This shows that a statistical analysis with simulated obstacle and mine densities may be more appropriate for our system.

## IV. Methodology

In our experiments, we assume the execution time of the obstacle detection/avoidance tasks and the mine detection task to be composed of a base amount plus an incremental amount dependent on the number of obstacles (or mines). We assume the number of obstacles and mines is random depending on the environment. In our simulations, we model both the obstacle and mine count as independent, identically distributed Binomial random variables with $n = 16$ and $p = 0.5$.

| | | Execution Time [ms] | | | |
|---|---|---|---|---|---|
| Task | Period [ms] | Base | Per Obj. | Priority | Criticality |
| Flight Control | 100 | 10 | n/a | 1 | 1 |
| Obstacle Detection | 1000 | 50 | 5 | 4 | 2 |
| Obstacle Avoidance | 500 | 50 | 6 | 3 | 3 |
| Mine Detection | 250 | 50 | 5 | 2 | 4 |

**Table 1. Example Task Set**

The details for the four UAS tasks modeled in our experiments are shown in Table 1. The task period is the time between releases of jobs for each task. The "base" time is the portion of the execution time for each job of a task that is fixed and does not depend on the workload. The "Per Obj." time is the portion of the execution time for each job that is required per object processed (mine or obstacle depending on task type).

Each Bernoulli trial in our experiments was conducted by executing the model task set for one hyper-period (1000 ms for this task set). We can limit the execution to one hyper-period since for both RM and ZSRM, all scheduling effects reset at the end of a hyper-period.

Importance Sampling was performed using the two-phase approach as described in Section II.B. In the tilting phase, tilting was performed by first mapping each Binomal random variable $b$ to a unit exponential random variable $e$ with the mapping $b = F_B^{-1}(F_E(e))$ where $F_B^{-1}(x)$ is the inverse CDF of the Binomial and $F_E(x)$ is the CDF of a unit exponential. The exponential random variable $e$ was then tilted by increasing its mean by a tilting parameter $s$.

The tilting parameter $s$ was varied between 1 and 10 in steps of 0.05, and short probe simulations of 50,000 Bernoulli trials were conducted for each value. The tilting parameter resulting in the minimum relative error was then selected to be used for importance sampling in the phase two simulation. In phase two, simulation was continued until the relative error estimate $RE(\hat{p})$ fell below the target value of 0.01.

Our simulations were performed on a 2.2 GHz Intel Core i7 processor with 4 cores. The simulation framework consists of five threads, a dispatcher thread and four worker threads in which each worker thread executed on one of the four cores. Note that the dispatcher thread was mostly idle except to start simulations on worker threads and collect results.

Bernoulli trials were started on the worker threads in batches of 4 by the dispatch thread. Each worker thread executed the model $\mathcal{M}$ and determined success or failure against the property $\Phi$ being tested. The dispatch thread then waits for all 4 trials to complete before determining success or failure of each trial,

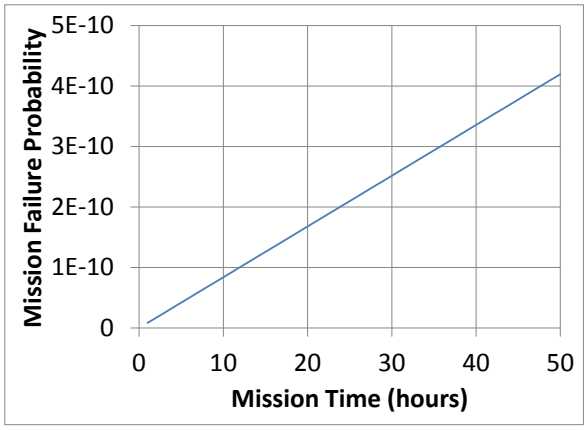


**Figure 1. Mission Failure Probability**

updating the values for $N$, $\hat{p}$ and $RE(\hat{p})$, then starting a new batch of 4 trials. A set of Bernouilli trials was terminated under the exit criterion of $N = 50,000$ for the Phase 1 probe simulations, or $RE(\hat{p}) < 0.01$ for the main importance sampled Phase 2 simulations.

Since some of the 4 worker threads may complete sooner than others, there can be idle time with this approach. It is tempting to allow each worker thread to operate as fast as possible to avoid this idle time, however this can lead to a bias in the estimation since it is possible there can be a correlation between the success/failure of a simulation, and the time required for the simulation to complete. Dispatching the trials in batches eliminates this potential bias.[6]

The approach described here estimates the probability of failure in one hyper-period. However, the mission time of a system typically consists of multiple hyper-periods. If we assume hyper-period failures are independent, we can estimate the mission failure probability estimate $\hat{p}_M$ in terms of the hyper-period failure probability estimate $\hat{p}$ as:

$$\hat{p}_M = 1 - (1 - \hat{p})^K \tag{9}$$

where $K$ is the number of hyper-periods in the mission. If $T_M$ is the mission time and $T_H$ is the length of a hyper-period then $K = \lceil \frac{T_M}{T_H} \rceil$. It can be shown that when $\hat{p}K$ is small (less than about 0.1), that $\hat{p}_M$ can be estimated as:

$$\hat{p}_M \approx \hat{p}K \tag{10}$$

and that $RE(\hat{p}_M) \approx RE(\hat{p})$.

As an example, if we let the hyper-period failure probability be $p = 2.36 \times 10^{-15}$ and the length of a hyper-period $T_H$ be one second, then Figure 1 shows the probability of mission failure $p_M$ as a function of the mission time $T_M$. This shows us that even for a 40 hour mission, the failure probability would be less than $5 \times 10^{-10}$.

## V. Results

Let $\mathcal{M}_{\text{RM}}$ and $\mathcal{M}_{\text{ZSRM}}$ be models for the timing behavior of our UAS software system with Rate Monotonic, and Zero-Slack Rate Monotonic scheduling, respectively. We have conducted two sets of simulations comparing each of these models against the following two properties:

- $\Phi_1$ = No deadline miss of any task
- $\Phi_2$ = No deadline miss of flight, or obstacle detection/avoidance tasks

Under the requirement $\Phi_1$, a single deadline miss is considered a system failure. Under requirement $\Phi_2$, only deadline misses of tasks that affect flight safety are considered a system failure.

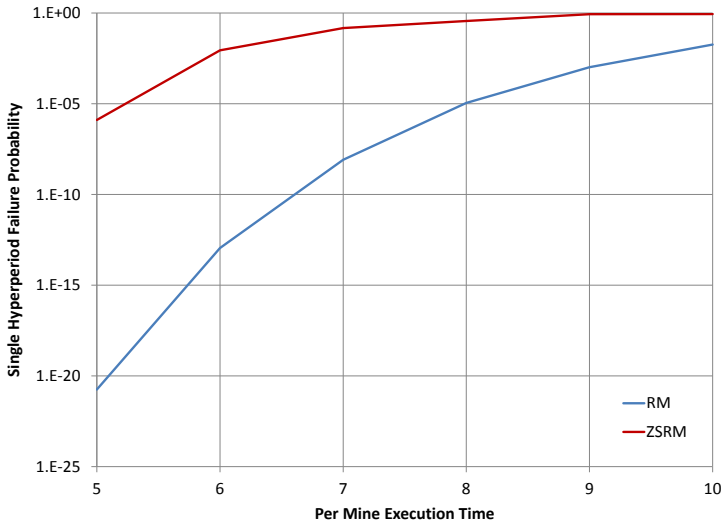


**Figure 2. Failure Probability for Failure on Any Deadline Miss**

### V.A. Experiment 1 – Strict Deadline Enforcement

In Experiment 1, we vary the per mine execution time of the mine detection task from 5ms to 10ms and compare estimates for the probability of failure over one hyper-period in $\mathcal{M}_{\text{RM}}$ and $\mathcal{M}_{\text{ZSRM}}$ when any task miss is considered a failure (i.e., we compare $Pr[\mathcal{M}_{\text{RM}} \models \Phi_1]$ with $Pr[\mathcal{M}_{\text{ZSRM}} \models \Phi_1]$). Each simulation run was conducted until the Relative Error (RE) of the probability estimates were at or below 0.01. Importance sampling was used to decrease the required execution time.

The results from the first set of experiments are shown in Figure 2. As the per mine execution time decreases from 10ms to 5ms, the overall demand on the system decreases. The result we see from the graph is that failure probability decreases at a super exponential rate. This suggests that the probability for this mode of failure is extremely sensitive to task execution time, and that small optimizations in performance can have dramatic effects.

Another observation from these experiments is that $\mathcal{M}_{\text{RM}}$ outperforms $\mathcal{M}_{\text{ZSRM}}$ at each point on the curve. This is not surprising, since we know that the design goal of ZSRM was to protect some tasks (the high criticality tasks) at the expense of other tasks (low criticality tasks). These results show that that selective protection comes at a price in terms of overall miss rate.

### V.B. Experiment 2 – Flight Critical Deadline Enforcement

In Experiment 2, we follow the same procedure as in the first experiment, except we use $\Phi_2$ (system failure is assumed only when flight critical task fails) instead of $\Phi_1$. That is, we compare $Pr[\mathcal{M}_{\text{RM}} \models \Phi_2]$ and $Pr[\mathcal{M}_{\text{ZSRM}} \models \Phi_2]$ as the per mine execution time changes between 5ms and 10ms. We use a target relative error of 0.01 for these simulations as well. The results from this experiment are shown in Figure 3. We see the same sensitivity to the per-mine execution time that we saw in the previous experiment with failure rate dropping rapidly as the execution time decreases.

Most notably in Experiment 2, we see that $\mathcal{M}_{\text{ZSRM}}$ performs better than $\mathcal{M}_{\text{RM}}$ under this modified condition. The system failure probability for $\mathcal{M}_{\text{ZSRM}}$ drops dramatically from the results in Experiment 1, while the failure probability for $\mathcal{M}_{\text{RM}}$ is nearly unchanged. This is because under ZSRM we have set the criticality of the mine detection task to the lowest value meaning the flight-critical tasks are better protected. RM on the other hand does not respect criticality, and so job deadlines are missed indiscriminately.

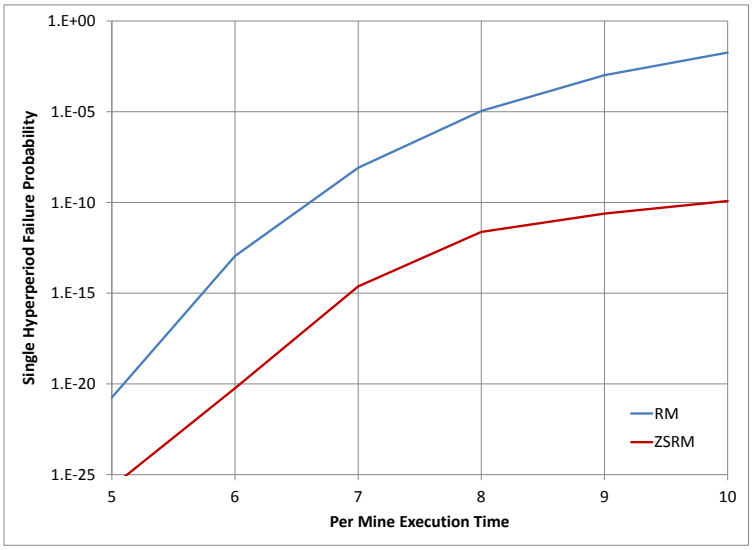


**Figure 3. Failure Probability for Failure on Flight Critical Deadline Miss**

### V.C. Experiment 3 – Effect of Importance Sampling

In Experiment 3, we explore the effect that use of Importance Sampling has on simulation time. Figure 4 shows the required simulation time to calculate $Pr[\mathcal{M}_{\mathrm{RM}} \models \Phi_2]$ (probability of failure for RM with any deadline miss) using brute-force Crude Monte-Carlo simulation, compared to simulation with Importance Sampling. In both cases the target target relative error was 0.01.

We can see that the required simulation time increases dramatically for Crude Monte-Carlo as the per mine execution time decreases. We know from Equation 5 that for a constant relative error, a 10-fold decrease in the probability of the event being estimated will result in a 10-fold increase in the number of simulations required to estimate that probability. As a result, the decreasing failure probability as per mine execution time decreases results in the observed rapid increase in simulation time. While we were only able to measure the simulation time down to a per-mine execution time of 8ms, we can estimate using Equation 5 that the required simulation time for a 7ms per-mine execution time would be approximately 34 days.

When importance sampling is used, we see a dramatic reduction in required simulation time over most of the curve. For the 7ms per-mine execution time case discussed above, Importance Sampling resulting in reducing the simulation time from 34 days to 30 minutes. The one exception where Crude Monte-Carlo is faster occurs for the 10ms per-mine execution time case. The reason for this is that the failure probability at this point is not sufficiently rare, and thus the benefits of Importance Sampling are lost due to the overhead of the tilting parameter estimation phase.

## VI. Related Work

Like statistical model checking, probabilistic model checking[7] is also an automated, algorithmic approach for computing numerical properties of stochastic systems. However, in this approach, the system is modeled as a finite state probabilistic automaton, e.g., a discrete time Markov chain (DTMC), a continuous time Markov chain (CTMC), or a Markov decision process (MDP) which is exhaustively explored in the analysis. The property is expressed as formula in a temporal logic, e.g., probabilistic Computation Tree Logic (PCTL).[8] For example, the system could be a DTMC modeling the repeated throwing of a biased coin that comes up "heads" with probability 0.55 and "tails" with probability 0.45. The property could be the "probability of seeing 3 heads followed by 4 tails followed by 5 heads". Probabilistic model checking then involves an exhaustive exploration of the DTMC's statespace to compute the property's value. Typically, this involves the

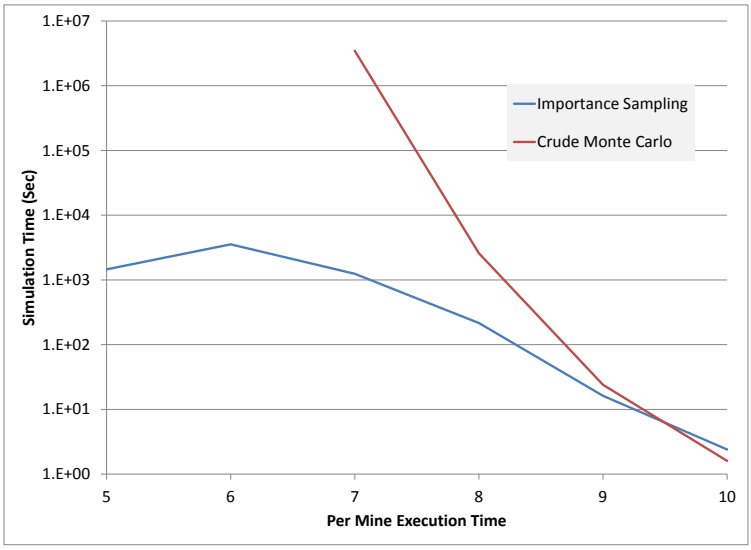


**Figure 4. Comparison of Simulation Time for Crude Monte-Carlo vs. Importance Sampling**

following steps: (i) compute the set of reachable discrete states $S$ of an automaton constructed by composing the system with the property; (ii) construct a set of equations $Q$ whose solution corresponds to the steady-state probabilities of $S$; and (iii) solve $Q$ numerically and extract the value of the property from the solution. Probabilistic model checking is an active area of research, involving both theoretical advancements[9] and practical tool development.[10] It has been used to verify systems ranging from pacemakers,[11] root contention protocols[12] and biological pathways.[13] For our approach, we used statistical model checking, since the systems we want to verify are too complex to be modeled as Markov chains.

## VII. Conclusion

We have shown that statistical model checking is a useful tool for evaluating software systems operating in stochastic environments. In particular, when models are too large or complex for formal methods, statistical model checking can be used to obtain quantifiable failure predictions with specific accuracy bounds. We have also shown that importance sampling can be a useful technique to reduce the simulation effort when estimating systems with the low failure probabilities necessary for deployment in airborne systems. We have applied these techniques to compare the deadline miss performance of two different scheduling algorithms and shown how changes in system requirements can affect the optimal choice of scheduling algorithm.

## References

[1]Liu, C. L. and Layland, J. W., "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM*, Vol. 20, No. 1, Jan. 1973, pp. 46–61.

[2]Younes, H. L. S., *Verification and planning for stochastic processes with asynchronous events*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2004.

[3]Srinivasan, R., *Importance Sampling: Applications in Communications and Detection*, Engineering online library, Springer, 2002.

[4]Niz, D. d., Lakshmanan, K., and Rajkumar, R., "On the Scheduling of Mixed-Criticality Real-Time Task Sets," *Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, RTSS '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 291–300.

[5]Clarke, E. M. and Zuliani, P., "Statistical Model Checking for Cyber-Physical Systems." *ATVA*, edited by T. Bultan and P.-A. Hsiung, Vol. 6996 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 1–12.

[6]AlTurki, M. and Meseguer, J., "PVeStA: A Parallel Statistical Model Checking and Quantitative Analysis Tool," *Proceedings of CALCO11: The 4th International Conference on Algebra and Coalgebra in Computer Science, Winchester, UK*,

August 2011, pp. 386–392.

[7]Stoelinga, M., *Alea jacta est: verification of probabilistic, real-time and parametric systems*, Ph.D. thesis, University of Nijmegen, the Netherlands, 2002, Available via `http://www.soe.ucsc.edu/~marielle`.

[8]Hansson, H. and Jonsson, B., "A Logic for Reasoning about Time and Reliability," *Formal Aspects of Computing (FACJ)*, Vol. 6, No. 5, December 1994, pp. 512–535.

[9]Feng, L., Kwiatkowska, M. Z., and Parker, D., "Automated Learning of Probabilistic Assumptions for Compositional Reasoning," *Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering (FASE '11)*, edited by D. Giannakopoulou and F. Orejas, Vol. 6603 of *Lecture Notes in Computer Science*, Springer-Verlag, Saarbrücken, Germany, March–April 2011, pp. 2–17.

[10]Kwiatkowska, M. Z., Norman, G., and Parker, D., "PRISM 4.0: Verification of Probabilistic Real-Time Systems," *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11)*, edited by G. Gopalakrishnan and S. Qadeer, Vol. 6806 of *Lecture Notes in Computer Science*, Springer-Verlag, Snowbird, UT, July 14 - July 20, 2011. New York, NY, July 2011, pp. 585–591.

[11]Chen, T., Diciolla, M., Kwiatkowska, M. Z., and Mereacre, A., "Quantitative Verification of Implantable Cardiac Pacemakers," *Proceedings of the 33rd Real-Time Systems Symposium (RTSS '12)*, IEEE Computer Society, San Juan, PR, USA, December 2012, pp. 263–272.

[12]Kwiatkowska, M. Z., Norman, G., and Sproston, J., "Probabilistic Model Checking of Deadline Properties in the IEEE 1394 FireWire Root Contention Protocol," *Formal Aspects of Computing (FACJ)*, Vol. 14, No. 3, April 2003, pp. 295–318.

[13]Heath, J., Kwiatkowska, M. Z., Norman, G., Parker, D., and Tymchyshyn, O., "Probabilistic model checking of complex biological pathways," *Theoretical Computer Science (TCS)*, Vol. 391, No. 3, February 2008, pp. 239–257.