A model for probabilistic monitoring and proactive restart of real-time operating systems under intensive state changes in cyber-physical systems⋆

Oleksandr Kozelskyi[1,*,†], Andriy Drozd[1,†], Bohdan Savenko[1,†], Piotr Gaj[2,†]

[1] *Khmelnytskyi National University, Khmelnytskyi, Instytutska street 11, 29016, Ukraine*
[2] *Silesian University of Technology, Gliwice, ul. Akademicka 2A, 44-100, Poland*

Abstract

The article presents a model of probabilistic monitoring and proactive restart of real-time operating systems (RTOS) in conditions of intensive changes in the states of cyber-physical systems. The proposed model uses a probabilistic approach to assess the risk of failures, which allows for timely prediction of potential problems and prevention of their critical impact on the functioning of the system. It provides proactive local recovery of problematic components using a dual watchdog mechanism, which includes both hardware and software control levels. This approach allows not only to detect failures at an early stage, but also to promptly respond to deviations from the normal operating mode of the system. A key aspect of the developed model is the ability to locally restart individual tasks before the hardware watchdog timer is triggered. This significantly reduces the overall computational load on the microcontroller, preventing excessive use of resources and reducing the likelihood of a complete emergency restart of the system. Integration of the proposed approach with real-time operating systems, such as FreeRTOS, provides ease of implementation and increases the efficiency of resource management in systems with limited capabilities. In addition, the model allows you to adapt to changing operating conditions and provides flexible tuning of monitoring parameters in accordance with the specifics of the operation of cyber-physical systems. Experimental studies have demonstrated the high effectiveness of the proposed method in reducing downtime and increasing the overall reliability of real-time operating systems. The proposed approach allows you to minimize the consequences of unexpected failures, increase the system's resistance to changes in the external environment and ensure the stability of its operation even in critical operating conditions.

Keywords

RTOS, probabilistic monitoring, watchdog timer, fault tolerance, cyber-physical systems, reliability

1. Introduction

Modern real-time operating systems (RTOSs) are characterized by a large number of concurrent tasks and a deep integration of hardware and software components. This complexity poses significant challenges for maintaining continuous system operation while satisfying stringent timing constraints (deadlines). Such requirements are especially critical in cyber-physical systems, where even minor errors or delays in managing physical processes can lead to severe consequences — system failures, equipment damage, or the compromise of safety-critical functions [1,2]. Given the potentially high cost of these failures, ensuring system reliability and fault tolerance remains a primary research focus in embedded and real-time computing.

A primary mechanism for mitigating failure risks and enhancing reliability is the watchdog timer, which may be implemented at both hardware and software levels. The conventional approach relies on rebooting the microcontroller when a crash or core failure is detected. However, this reactive

ICyberPhyS'25: 2nd International Workshop on Intelligent & CyberPhysical Systems, July 04, 2025, Khmelnytskyi, Ukraine
1* Corresponding author.
† These authors contributed equally.
oleksandr.kozelskiy@khmnu.edu.ua (O. Kozelskyi); andriydrozdit@gmail.com (A. Drozd); savenko_bohdan@ukr.net (B. Savenko); piotr.gaj@polsl.pl (P. Gaj);
0009-0002-7157-6499 (O. Kozelskyi); 0009-0008-1049-1911 (A. Drozd); 0000-0001-5647-9979 (B. Savenko); 0000-0002-2291-7341 (P. Gaj);

strategy addresses issues only after they occur, incurring significant computational and temporal overhead. As system complexity and reliability demands grow, hybrid methods that combine hardware and software monitoring have become increasingly relevant. In particular, a dual-watchdog architecture can enable proactive, localized restarts of individual software components via a probabilistic risk-assessment model, facilitating early failure detection, avoiding full-system reboots, and substantially reducing overall downtime.

Embedded systems often operate under severe constraints — limited processing power, tight real-time deadlines, and restricted memory availability [3]. Under these conditions, computationally intensive analytical techniques, such as machine learning algorithms or comprehensive formal models, may be impractical. Simplified probabilistic approaches, in contrast, allow real-time estimation of failure risk based on a small set of key system parameters. This capability supports a software-based predictive control layer that identifies early warning trends before system freezes or critical failures occur. Consequently, targeted local restarts of malfunctioning tasks, drivers, or modules can be executed to prevent complete system failures.

The main scientific contributions of this study include the development of a lightweight stochastic monitoring model that can run inside an RTOS task; the proposal of a two-level fault-tolerance architecture combining a Predictive Watchdog Task (PredictiveWDT) with a hardware watchdog timer (HW WDT); the implementation of a prototype on an STM32F407 microcontroller that detects task hangs in 0.11–0.13 s and finishes a local soft reset in 0.35 s—whereas a full recovery via the HW WDT takes almost 3 s—thereby cutting total system downtime from 4.5 % to 1.88 % (-42 %); and the demonstration that the approach markedly reduces global restarts while introducing only negligible computational overhead, together with a clear statement of its practical limitations.

2. Analysis of known solutions

Existing fault-tolerance strategies for embedded real-time operating systems (RTOS) can be classified into hardware-based, software-based, and hybrid models, each with its own characteristics in terms of performance, resource usage, and implementation complexity. The main classes of fault-tolerance models are illustrated in Figure 1: hardware watchdog timers, redundancy-based architectures, formal verification methods, machine-learning models.

A HW WDT is typically implemented either as an integrated peripheral module within the microcontroller or as an external supervisory device with its own clock generator. In such a scheme, the application is required to periodically refresh the timer counter or send a signal indicating normal system operation. If this refresh does not occur within a specified interval, the watchdog interprets it as a system fault and initiates a complete system reset. Due to its independence from the real-time operating system (RTOS) kernel, the timer is capable of restoring operation even in cases of total system hang or deadlock. However, this approach is purely reactive, since the reset is triggered only after a failure has already occurred, resulting in the loss of execution context and system downtime. In safety-critical domains such as automotive electronics, industrial automation, or medical devices, even short interruptions lasting several seconds may be unacceptable. The redundancy-based model entails duplicating critical system components to enhance overall fault tolerance. Common implementations include N-modular redundancy with majority-voting logic or hot/cold-standby configurations, in which a backup module immediately (or near-instantaneously) takes over when the primary fails [20,21]. Such schemes are essential in applications where even a momentary interruption is unacceptable, for example, in avionics systems [16] or continuous industrial conveyor operations. However, redundancy requires substantial additional hardware, sophisticated state-synchronization mechanisms, and incurs considerable financial cost [12,17], rendering it impractical for many resource-constrained embedded platforms [5].

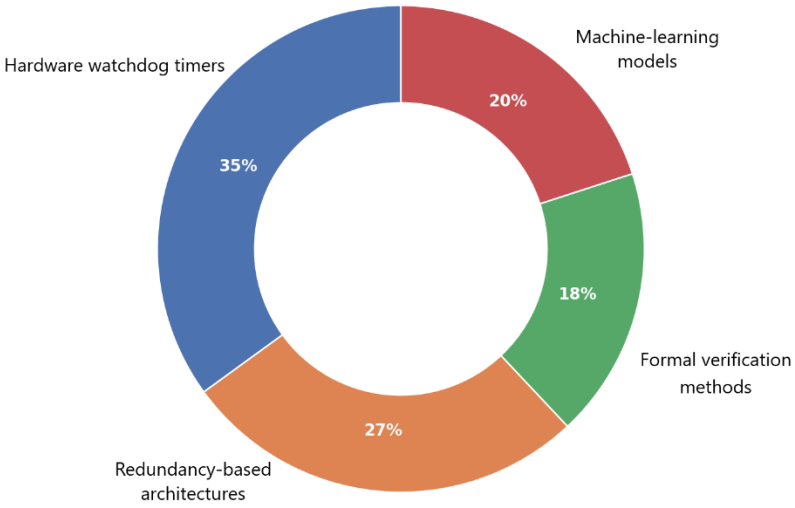

Figure 1: Taxonomy of existing fault-tolerance models for RTOS.

Formal methods and verification techniques — such as timed automata, Petri nets, and model-checking — are extensively employed in safety-critical domains (e.g., DO-178C, ISO 26262) to demonstrate the absence of specific error classes and deadlocks during the design phase [4]. These techniques enable rigorous analysis of system dynamics and formal proof of specification conformance. However, their application to operational (online) failure detection is frequently constrained [10] by high computational demands [8], implementation complexity, and the state-space explosion problem encountered in realistic systems [6,7].

Machine-learning (ML) techniques have been widely adopted in large-scale cloud infrastructures and high-performance computing (HPC) clusters for tasks such as load forecasting, anomaly detection and predictive task migration [14]. Despite their potential for real-time data analysis, the performance of ML models critically depends on the chosen evaluation metrics [26], while also imposing substantial demands on system resources — namely, memory capacity, processing power, and high-quality training datasets. Deploying even relatively lightweight ML algorithms on resource-constrained microcontrollers (e.g., STM32, AVR, ESP32) can therefore be prohibitive in terms of computational overhead and energy consumption, as illustrated by the challenges of achieving Byzantine-fault tolerance [15]. Furthermore, guaranteeing deterministic behavior of ML-driven methods under rare or corner-case scenarios is difficult, which conflicts with the stringent verification requirements of safety-critical domains such as avionics and medical devices.

In summary, while hardware watchdog timers ensure reactive recovery from global hangs and software monitors enable more granular, rapid local response, each approach has drawbacks: redundancy schemes incur heavy hardware overhead, and formal or ML-based models are often too resource-intensive for simple embedded platforms.

A hybrid strategy that combines these paradigms offers the most promise. In this model, a hardware watchdog acts as the ultimate safeguard against catastrophic system freezes, while a lightweight software layer proactively detects emerging fault trends and performs localized restarts of only the affected components. Implementations of this predictive layer range from simple threshold-based heuristics to analytical probabilistic risk-assessment models [11]. Given the constraints of embedded environments and the requirement for low-latency execution, simplified probabilistic models with a limited number of discrete states represent a practical compromise. Such approaches have been successfully applied in domains like automated guided vehicle (AGV) systems and predictive maintenance of industrial equipment [13], where analysis of historical state transitions enables early risk estimation and thus reduces reliance on full-system resets — minimizing downtime and enhancing operational continuity.

The objective of this study is to develop and validate a proactive component-restart model for embedded real-time systems, grounded in probabilistic failure-risk monitoring and integrated with conventional hardware watchdog mechanisms. This approach aims to reduce the frequency of system-wide reboots, thereby minimizing the time and computational costs associated with recovery. Moreover, integrating the proposed model with hardware-level controls is expected to enhance the fault tolerance of resource-constrained cyber-physical systems subject to rapid state changes. By providing a computationally efficient solution, the model offers practical value to embedded-system designers, ensuring high availability of critical applications while mitigating the impact of potential failures. It should be noted that optimizing task allocation in multiprocessor embedded systems remains an important research direction, as confirmed, in particular, by the findings of study [27].

3. Formulation of the problem

An evaluation of existing fault-tolerance mechanisms for embedded real-time systems reveals inherent trade-offs. Hardware watchdog timers guarantee a full system reboot following a critical failure; however, they activate only after system operability has been lost, resulting in total context loss and necessitating complete reinitialization. Software watchdogs, by contrast, can detect failures at the task level at an earlier stage, yet their efficacy depends on the underlying RTOS kernel remaining operational. Although redundancy-based schemes achieve high reliability, they impose significant overhead in terms of hardware replication and complex synchronization algorithms. Similarly, formal verification methods and machine-learning-based approaches often require substantial computational resources or incur prohibitive verification costs, making their deployment on resource-constrained microcontrollers impractical.

To address these limitations in resource-limited devices, a hybrid fault-tolerance architecture is proposed. In this design, the HW WDT serves as a last-resort mechanism for recovering from global system freezes, while an intelligent software layer continuously monitors system metrics, detects emerging fault conditions, and executes localized recovery procedures — thereby reducing the frequency of full system resets.

The principal challenge in realizing this hybrid model lies in devising a lightweight yet effective state-assessment mechanism capable of estimating the likelihood of critical events — such as system freeze, task-deadline violations, or queue overflows — in real time. Employing oversimplified triggers (e.g., fixed thresholds on queue occupancy) risks generating false alarms, whereas sophisticated predictors based on machine learning or formal methods exceed the computational budgets of typical embedded processors.

Accordingly, this work aims to develop a resource-efficient crash-prediction model that enables rapid estimation of the probability of transition to a critical system state, while integrating seamlessly with a software watchdog timer to initiate soft resets of individual components. The model is designed to preserve normal operation in the presence of non-fatal anomalies and to delegate full recovery to the HW WDT in cases of unrecoverable system freezes.

Given these requirements, hybrid models that combine implementation simplicity with adequate prediction accuracy present a promising solution [18]. In particular, lightweight probabilistic frameworks — requiring only minimal computational resources — offer an effective balance between rapid system-state assessment and reliable failure forecasting [9].

4. The main part

4.1. Basic architecture of the RTOS and its modification taking into account probabilistic monitoring.

Effective resource management and maintaining stable operation of real-time operating systems (RTOS) are key tasks for their reliable operation, especially in the case of use in cyber-physical systems (CPS) and the Internet of Things (IoT). The expansion of RTOS application areas is

accompanied by the need for fast processing of large amounts of data in real time, which requires effective mechanisms for ensuring fault tolerance.

The standard RTOS architecture includes several key components. The scheduler determines the execution order of tasks using algorithms such as priority-based scheduling or Earliest Deadline First (EDF). Tasks are independent processes responsible for various functions, including data collection from sensors, information exchange, and control of actuators. Task execution can be either periodic or event-driven. System services such as timers, queues, and semaphores facilitate inter-task communication, synchronization, and data transfer. They may also include logging and configuration utilities. Device drivers provide interfaces for interacting with hardware modules such as UART, SPI, I^2C, and ADC, offering the necessary APIs for software components. The HW WDT functions as an autonomous microcontroller module that monitors system stability and triggers a forced restart in case of a loss of functionality.

In a standard RTOS, the watchdog timer is configured to have its counter periodically refreshed by calling feed_watchdog(), which prevents it from expiring during normal operation. If this refresh does not occur in time — for example, because a critical component has frozen — the watchdog timer triggers a full microcontroller reset.

Although this mechanism reliably restores system functionality, it suffers from a fundamental limitation: it is purely reactive, invoking a reset only after a failure has already occurred. Consequently, the system's execution context is irrevocably lost — a behavior that can be unacceptable in stringent real-time environments such as industrial process control or embedded medical devices.

In practice, developers attempt to mitigate unnecessary resets by instrumenting key execution points with calls to feed_watchdog(). While this approach can reduce avoidable restarts, it provides no proactive assessment of system health and offers no means to confine faults locally. Under severe conditions — such as a scheduler deadlock or data-bus blockade — the system still must perform a full restart, leading to service interruption and the loss of all volatile state.

An interaction model of a conventional RTOS setup utilizing a hardware watchdog for basic fault recovery is shown in Figure 2 (left).

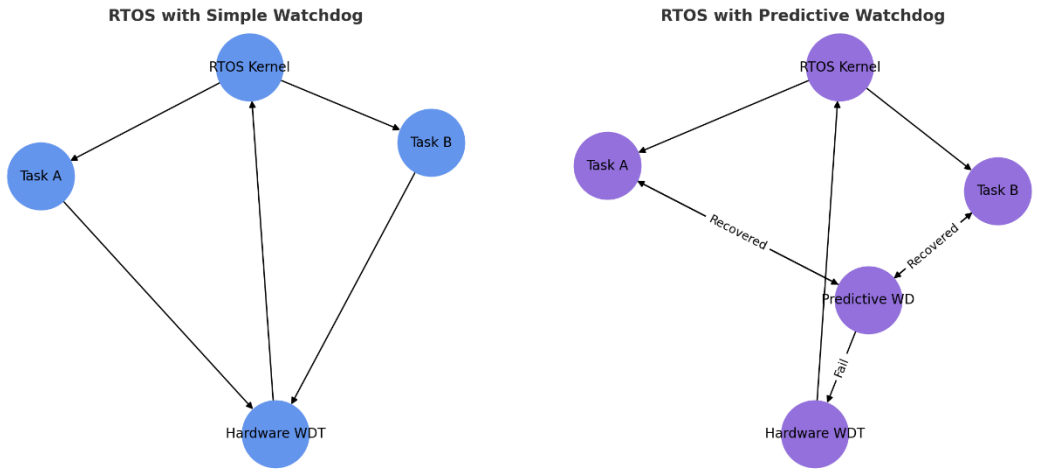

Figure 2: Graph-based models of typical RTOS with HW WDT and an enhanced probabilistic monitoring system.

Figure 2 (right) illustrates an enhanced RTOS supervision model that complements the classical scheme with a predictive and locally recoverable layer. This additional layer enables the system to intercept and mitigate emerging faults before they escalate into critical failures. The full system architecture, including all interacting components, is depicted in Figure 3 and is designed in such a way as to enable the determination of the system's next steps [23, 24, 25].

The enhanced architecture comprises the following primary components:

- Scheduler continues to execute standard task scheduling algorithms without changes.
- Tasks executes the function similarly to the traditional RTOS architecture.
- System services and device drivers: retain their existing interfaces to the kernel — timers, queues, semaphores, and peripheral abstractions (e.g., UART, SPI) — unchanged.
- HW WDT serves as a secondary safeguard, triggering a full microcontroller reset if the system experiences an unrecoverable freeze.
- PredictiveWDT has been added — a novel module responsible for continuous system-state monitoring and proactive recovery. Although referred to as the Predictive Watchdog Timer in the architectural diagram (see Figure 3), this component operates as a full-fledged RTOS task rather than a conventional timing mechanism. Its main functions are as follows: system activity analysis gathers operational metrics (e.g., queue lengths, task execution times, heartbeat signals) via message queues or global variables to assess system stability; failure prediction employs a lightweight probabilistic model to estimate the likelihood of a transition into a critical state; preventive error correction initiates a localized restart of a failing task (via vTaskDelete and xTaskCreate) or reinitializes a faulty driver (ReInitDriver) whenever the predicted risk exceeds a predefined threshold; HW WDT supervision refreshes the hardware watchdog during normal operation. If local recovery fails, withholding the refresh prompts the HW WDT to execute a full system reset.

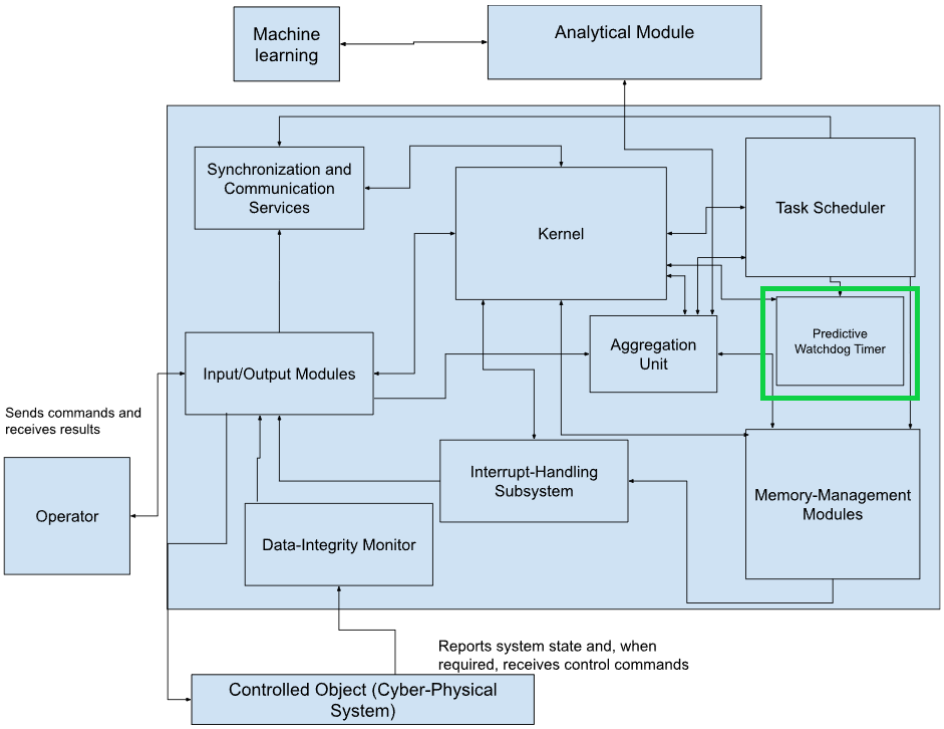

Figure 3: General architecture of the proposed RTOS for cyber-physical systems with intensive state changes, with the Predictive Watchdog Timer module highlighted in green.

Algorithm for interaction of the PredictiveWDT with system components:

1. Each critical task periodically transmits an activity signal — for example, by updating lastSeenTime[taskID] or sending a heartbeat message to the PredictiveWDT queue.
2. The PredictiveWDT runs at high priority at regular intervals (e.g., every 150–250 ms), processes the gathered signals, and assesses the current system state.
3. If the analysis indicates a high probability of a hang or critical state, a soft reset of the problematic module is performed. If after several attempts at local recovery the system

remains in a critical state, the PredictiveWDT stops updating the watchdog timer, which leads to a global reset.
4. Under normal conditions, the PredictiveWDT transmits a control signal to the HW WDT, which confirms stable system operation.

Thus, the proposed architectural model with PredictiveWDT provides an optimal combination of preventive analysis, local fault elimination and the classic emergency restart mechanism. This allows to increase the RTOS resistance to failures, reduce downtime and improve the system adaptability to changing operating conditions.

4.2. Failure prediction and preventive component restart model in RTOS

The proposed model involves the integration of a hardware watchdog as a mechanism of the final level of protection and intelligent software control (Software Watchdog), which predicts possible failures and locally restores components to their critical state.

The functional scheme provides a two-stage mechanism:

1. Software level of preventive control – analyzes the system state and performs local soft reset of individual tasks or drivers in case of detection of anomalous deviations in operation. This can be both a simple failure and anomalous actions for example from the point of view of malicious software [19,22].
2. Hardware level of protection — in case of ineffective software recovery or general system freeze, the HW Watchdog Timer is activated, which initiates a full restart.

Such a two-level strategy significantly reduces the number of full reboots (resets), reduces downtime and ensures stable operation of embedded and cyber-physical systems with limited resources and strict time requirements. The proposed model assumes that the system state is evaluated discretely, at time points $t = k \cdot \Delta t$, where Δt is the execution period of the PredictiveWDT in the real-time operating system. At each step, the system is in one of the defined states (for example, $\{N, C, F\}$: "Normal," "Critical," or "Fail"). The transition between states is based on the analysis of current performance indicators, in particular, task processing time, queue load or activity signal update frequency. The model uses a probabilistic approach, which allows predicting possible failures and initiating preventive recovery before the transition to a critical state.

Let $S = \{N, C, F\}$ be a finite set of states. At each time step $t^k = k\Delta t$, the system (or PredictiveWDT, reflecting system health monitoring) is in one of the states $x(k) \in S$.

For simplicity, let us assume that:

1. N (Normal) – the system is in the normal range (queues are not crowded, signals about the activity of all tasks arrive on time).
2. C (Critical) – the risk of a quick failure is high (significant delays, frequent missed deadlines, the task stops sending activity signals).
3. F (Fail) – actual hang/failure (in the model, this is an absorbing state or an indicator that a global reset has occurred).

Transitions between states are governed by a probability matrix $P \in R^{3\times3}$:

$$P = \begin{bmatrix} P_{NN} & P_{NC} & P_{NF} \\ P_{CN} & P_{CC} & P_{CF} \\ 0 & 0 & 1 \end{bmatrix}, \quad (1)$$

where $p_{ij} = P(x+1) = j | x(k) = i)$, and $x(k)$ – denotes the system state at time step k. The state F is absorbing, implying that once reached, the system initiates a full hardware reset. At each time step k, the predictive monitoring component estimates the probability of failure at the next step:

$$P_F(k) = P\big(x(k+1)\big) = F\big|x(k)\big), \tag{2}$$

this can be computed as:

$$P_F(k) = P_{CF} \cdot I_{x(k)=C} + P_{NF} \cdot I_{x(k)=N} = N, \tag{3}$$

where I is the indicator function. Let $\theta \in [0,1]$ denote the failure risk threshold. If the condition

$$P_F(k) \geq \theta \tag{4}$$

is satisfied, a software restart of the task considered the cause of the critical state is performed. Let $R(T_i)$ denote the restart or reinitialization operation:

$$R(T_i) := D(T_i) \circ C(T_i), \tag{5}$$

where T_i is the identifier of the task to be restarted, $D(T_i)$ denotes the deletion of the task from the RTOS scheduler, $C(T_i)$ represents the recreation of the task with its initial configuration, $\circ$ is the composition operator (sequential execution of actions).

Let Δt be the monitoring interval, and T_{HW} the timeout of the HW WDT. Let f denote the hardware watchdog feed function (feed_HW_WDT). If the system remains in the critical state C for m, consecutive monitoring steps and the local recovery attempt fails, the Predictive Watchdog deliberately suppresses the hardware watchdog feed signal at time step $k + m$:

$$f(k+m) = 0 \tag{6}$$

This action triggers a full system reset. To maintain adaptivity, the transition probabilities p_{ij} can be updated using an exponentially weighted moving average:

$$P_{ij}^{(k)} = \alpha \cdot I_{x(k-1)=i \wedge x(k)=j} + (1-a) \cdot P_{ij}^{(k-1)}, \tag{7}$$

where $\alpha \in (0,1)$ is the smoothing coefficient, $I_{x(k-1)=i \wedge x(k)=j}$ is the indicator function that equals 1 if the system was in state i at time $k-1$ and in state j at time k , and 0 otherwise.

Thus, the definition of N, C, F depends on the application scenario and the thresholds set. Such thresholds are determined empirically or based on the specifics of the application.

The probabilities of transitions between states in Predictive Watchdog are dynamic and can be adjusted in real time depending on changes in key system metrics. The main influencing factors are queue fullness, system activity signal latency, memory usage, and other system performance parameters.

The proposed model implements a two-stage approach to system recovery. If the analysis indicates a high probability ($\geq\theta$) of a transition to the Fail (F) state, the PredictiveWDT initiates a local recovery procedure by performing a soft reset of the problematic component. If the system fails to stabilize after the soft reset and remains in the Critical (C) state, the software layer deliberately refrains from updating the HW WDT (i.e., the feed() function is not called). As a result, after the expiration of the hardware watchdog timeout T_{HW}, a full microcontroller reset is triggered, restoring the system to the Normal (N) state.

Thus, when deviations remain within tolerable limits, a soft reset obviates the need for a global restart and thereby reduces system downtime. The HW WDT is retained as a final safeguard should the software layer lose control. Formally, if at step k the soft reset procedure is started, then at step $k+1$ we artificially set $x(k+1) = N$ (if the restart was successful).

Since we have $\Delta t \ll T_{HW}$, then the software layer has several attempts (for example, 10–20 cycles with a step of Δt) to perform a soft reset before the HW WDT timeout expires. If the PredictiveWDT determines that the system state has returned to Normal (or remains in Warning without progressing to Critical), it calls HW_WDT_Feed(), continuing normal operation. Otherwise, it deliberately withholds the feed signal to the hardware watchdog, allowing the timer to expire and trigger a full system reset (see Figure 4, right).

As a result, the mathematical scheme has the following steps:

1. The state of the system is denoted as $x(k)$ – a discrete moment in time in a finite space S.
2. The transition between states is determined by probabilities.
3. The probability of failure in one step is defined as p_{iF} (from Critical to Fail). When the threshold θ is exceeded – we launch preventive actions (soft reset).
4. If necessary, we consider the forecast for r steps through P^r.
5. If local recovery is ineffective, the system goes into state C (Critical) and continues to operate until the HW WDT timeout.

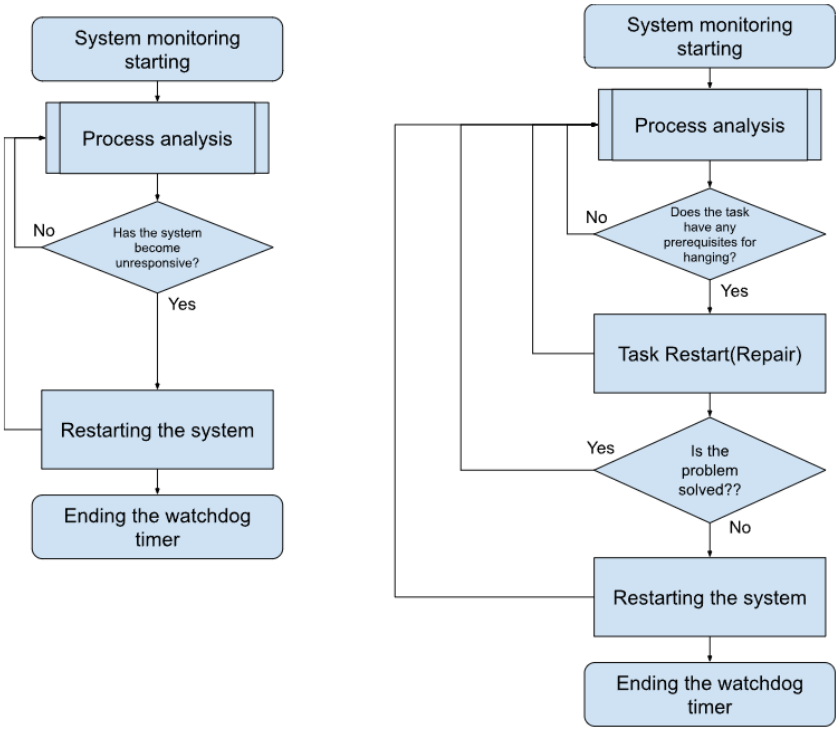

Figure 4: Schematic diagram of the standard watchdog timer workflow (left) and of the enhanced predictive software-layer watchdog timer (right).

The proposed model is based on a discrete analysis of changes in the system state with the possibility of dynamically updating the probability matrix P in real time, which ensures effective detection of potential failures without significant computational costs. The basic concept of the model combines a probabilistic approach to failure risk assessment with a two-stage recovery strategy.

The first stage involves local preventive recovery through a soft reset, which is triggered when an increased risk of failure is detected; this allows for restoring the operability of individual tasks or drivers without requiring a full system reboot. If the software-based recovery proves ineffective, the second stage is activated—resulting in a global restart initiated by the HW WDT, which ensures restoration of system functionality in critical scenarios.

Thus, the model allows to effectively avoid minor failures, reducing the need for a full reset, while ensuring resistance to critical failures when software methods do not work.

5. Experimental research

5.1. Prototype preparation

To assess the effectiveness of the two-stage fault tolerance mechanism (combination of hardware watchdog timer and software layer with adaptive analysis), a series of experiments were conducted

on the basis of the STM32F407 microcontroller platform (Cortex-M4 core, clock frequency 168 MHz). The test environment used the FreeRTOS real-time operating system with five application tasks:

1. TaskA – the main task that works with the periphery (SPI/UART) and periodically creates high loads on the system.
2. TaskB, TaskC – auxiliary tasks for signal processing and interaction with sensors.
3. Logger – a service for recording diagnostic messages and event timestamps.
4. PredictiveWDT – a separate task that implements intelligent watchdog control with a frequency of 100 ms.

The HW WDT had a timeout of 2 seconds. PredictiveWDT checked activity signals from other tasks on each cycle and analyzed the level of queue filling. A dynamic probabilistic model was used to assess the system state, taking into account the history of transitions over the last 50 observations.

The system estimated the probability of transition to the Fail state (F). If this indicator exceeded the set threshold, PredictiveWDT performed a selective soft reset of the problematic module (deletion and re-creation of the task or driver restart). In the case when local measures had no effect, the control signal to the watchdog timer was deliberately not transmitted, which caused a global system restart after 2.7 seconds.

To unify the experiments, special failure scenarios were defined when TaskA was artificially blocked (frozen) or generated excessive amounts of data, which caused queues to grow by more than 80%, or delayed activity signals. We checked how quickly the software Watchdog Timer would detect a critical state, whether it would have time to perform a soft reset, and whether a hardware reset would be required at all. Timestamps were recorded via Logger, as well as via the event tracking mechanisms in FreeRTOS+Trace. For each scenario, a series of runs (at least 10) were repeated and the average values of the main indicators were calculated.

The key parameter for assessing the system's effectiveness was the failure detection time (Mean Time to Detect, MTTD), which determined how much time passed from the moment anomalous behavior appeared (for example, the absence of an activity signal from TaskA) before the PredictiveWDT transitioned to the Critical (C) state and increased the probability of failure, initiating a soft reset. In parallel, the system recovery time (Mean Time to Repair, MTTR) was measured, which determined how much time it took to return to Normal (N) after a local or global restart. The time required for a soft reset (recovery of a specific task) was compared with the time required for a full restart via the HW WDT.

The Global Reset Rate (GRR) was also analyzed, which reflects the percentage of cases when a local recovery was not successful and a full system reset occurred. It was important to assess how effectively the PredictiveWDT prevents fatal failures, whether it is too optimistic in its predictions and misses critical situations, or, conversely, whether it generates an excessive number of false soft resets when the system could recover on its own.

Additionally, FreeRTOS was profiled using integrated trace tools to analyze the average additional CPU load caused by calculating the parameters of the predictive model. This made it possible to verify that the PredictiveWDT does not create an excessive load on the system and does not negatively affect the performance of other critical tasks.

5.2. Conducting experiments and comparing efficiency

During testing, PredictiveWDT detected TaskA hangs due to SPI overload in an average of 0.11–0.13 s, since the check period was 100 ms. In the case of a critical condition, it performed a restart of TaskA, which took up to 0.35 s. In comparison, the HW WDT required almost 3 s to fully recover the system, which resulted in a 4.5% downtime in 3 minutes (Figure 5). In tests with TaskA deadlock, PredictiveWDT solved the problem locally in 100% of cases, and in 0% of cases the system restarted via the HW WDT. The total downtime in this case decreased to 1.88%, which is 42% less than in the first test. Experimental results confirm that the proposed model, which implements regular soft

restart of problematic components, allows to significantly reduce the average downtime compared to a full system restart. At the same time, the system maintains a strict level of protection in cases where software recovery is ineffective. The execution time of the algorithm remains practically unchanged, since the state estimation process uses a lightweight stochastic model with minimal computational costs.

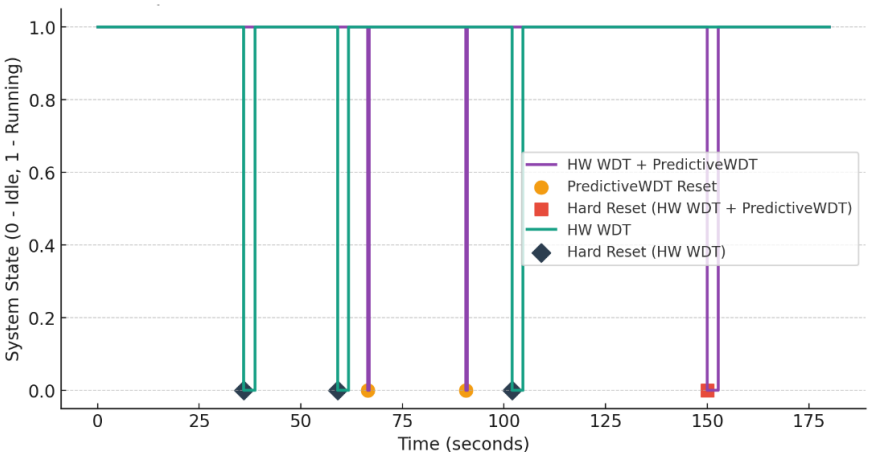

Figure 5: Comparison of the system with and without a software watchdog timer (3 minutes) with random failures.

In Figure 5, two system configurations are compared: the combined PredictiveWDT + HW WDT scheme (purple line) and the traditional HW WDT-only scheme (turquoise line). In the PredictiveWDT configuration, two proactive software resets were initiated upon detecting elevated failure risk, and one hardware reset occurred as a result of a complete system freeze and the subsequent withholding of the HW WDT feed. By contrast, the HW WDT-only configuration experienced three hardware resets. The relative thickness of the bars illustrates that the duration of each software reset is substantially shorter than that of a hardware reset, which collectively leads to a marked reduction in overall system downtime.

The results obtained indicate the feasibility of using a two-stage (software-hardware) mechanism to increase the reliability of embedded and cyber-physical systems. The proposed approach is especially important in areas where simple systems can have critical consequences – industrial automation, robotics, transportation, medical devices – but there are strict limitations on hardware resources, which makes it impossible to use complex ML models or redundant hardware circuits.

Thus, the proposed model provides a compromise between adaptive response to failures and guaranteed system stability, minimizing the number of global restarts, reducing downtime, and not creating excessive load on computing resources.

6. Conclusions

The paper presents a two-stage model for increasing fault tolerance for embedded and cyber-physical systems, which combines a hardware watchdog as the last level of protection against critical failures and an intelligent software layer (Software Watchdog), which proactively detects the probability of failure and performs local recovery of individual components. The main idea of the approach is not to wait for the actual system hang, but to use dynamic analysis of the current state to assess the risk of transition to a critical mode and initiate preventive actions in advance. If local recovery is ineffective, the system intentionally allows the hardware watchdog to operate, which provides a full restart in the event of a deep freeze.

The proposed model demonstrates practical effectiveness for real-time embedded and cyber-physical systems, where it is necessary to ensure reliable recovery of operability under strict time constraints. The use of a lightweight probabilistic model allows for quick analysis of the system state without complex formal or ML methods, and the two-level mechanism minimizes the number of

hardware resets, leaving HW WDT only as a backup protection. The combination of software preventive recovery with a hardware protection mechanism creates an effective balance between reactive and proactive strategies for increasing fault tolerance. The results obtained confirm the feasibility of applying the approach in various industries - from industrial automation and robotics to automotive and medical systems, where it is critically important to reduce downtime and ensure stable continuous operation, and also open up opportunities for improving the model and building new methods. It is worth noting, however, that the current model requires manual tuning of the failure risk threshold (θ), which may limit its generalizability across diverse hardware platforms and workloads. Addressing this challenge through adaptive or self-learning threshold mechanisms remains a promising direction for future research.

The experimental results show that the average fault detection time is only 0.11–0.13 s, and a local soft reset finishes in approximately 0.35 s—about 8–9 times faster than a full reboot (≈3 s). Among the main limitations of the method are its sensitivity to probability-threshold selection, the need for fine-grained tuning on each hardware platform, and a potential increase in power consumption at high checking frequencies. Future research will focus on adaptive auto-tuning of thresholds, integrating lightweight ML models into the PredictiveWDT, and extending the approach to distributed RTOS clusters and digital twins.

Declaration on Generative AI

AI tools were used exclusively for translation and proofreading purposes. All content was originally written by the author.

References

[1] S. Saraeian, B. Shirazi, Digital twin-based fault tolerance approach for Cyber–Physical Production System, ISA Transactions 130 (2022) 35–50. DOI:10.1016/j.isatra.2022.03.007.

[2] P. Alho, Service-based Fault Tolerance for Cyber-Physical Systems: A Systems Engineering Approach, December 2015. DOI:10.13140/RG.2.1.2862.6003.

[3] J. Navarro, Introduction to System Reliability Theory, Springer, New York, 2022, 181 p. DOI:10.1007/978-3-030-86953-3.

[4] M. Rausand, A. Barros, A. Hoyland, System Reliability Theory: Models, Statistical Methods, and Applications, 3rd edn., Wiley, 2021, 864 p. DOI:10.1002/9781119373940.

[5] Y. Chou, J. Yang, C. Wu, An energy-aware scheduling algorithm under maximum power consumption constraints, Journal of Manufacturing Systems 57 (2020) 182–197. DOI:10.1016/j.jmsy.2020.09.004.

[6] K. Zagalo, Stochastic analysis of stationary real-time systems, Ph.D. thesis, Sorbonne Université, 2023. URL: https://theses.hal.science/tel-04378907v1/file/ZAGALO_Kevin_these_2023.pdf.

[7] K. Zagalo, Y. Abdeddaïm, A. Bar-Hen, L. Cucu-Grosjean, Response Time Stochastic Analysis for Fixed-Priority Stable Real-Time Systems, IEEE Transactions on Computers 72(1) (2023) 3–14. DOI:10.1109/TC.2022.3211421.

[8] M. Rahnamania, F. Ashtiani, A New Analytical Approach for Delay Analysis in the Presence of Correlated Arrivals, in: Proc. 12th Iran Workshop on Communication and Information Theory (IWCIT), Tehran, Iran, 2024, pp. 1–6. DOI:10.1109/IWCIT62550.2024.10553217.

[9] F. Marković, P. Roux, S. Bozhko, A. V. Papadopoulos, B. B. Brandenburg, CTA: A Correlation-Tolerant Analysis of the Deadline-Failure Probability of Dependent Tasks, in: Proc. IEEE Real-Time Systems Symposium (RTSS), Taipei, Taiwan, 2023, pp. 317–330. DOI:10.1109/RTSS59052.2023.00035.

[10] N. Ueter, M. Günzel, J.-J. Chen, Response-Time Analysis and Optimization for Probabilistic Conditional Parallel DAG Tasks, in: Proc. IEEE Real-Time Systems Symposium (RTSS), Dortmund, Germany, 2021, pp. 380–392. DOI:10.1109/RTSS52674.2021.00042.

[11] E. Cabral, F. Tofoli, R. F. Sampaio, R. P. Leão, Reliability assessment applied in the design of an industrial substation in the context of Industry 4.0, Electric Power Systems Research 231 (2024) 110365. DOI:10.1016/j.epsr.2024.110365.
[12] S. Safari et al., A Survey of Fault-Tolerance Techniques for Embedded Systems From the Perspective of Power, Energy, and Thermal Issues, IEEE Access 10 (2022) 12229–12251. DOI:10.1109/ACCESS.2022.3144217.
[13] M. Saramud, V. Losev, M. Karaseva, Implementation of the t/(n−1) decision-making algorithm by RTOS and FPGA tools, AIP Conference Proceedings 2700(1) (2023) 070005. DOI:10.1063/5.0126813.
[14] S. Ramani, R. H. Jhaveri, ML-Based Delay Attack Detection and Isolation for Fault-Tolerant Software-Defined Industrial Networks, Sensors 22(18) (2022) 6958. DOI:10.3390/s22186958.
[15] D. Bouhata, H. Moumen, J. A. Mazari, A. Bounceur, Byzantine fault tolerance in distributed machine learning: a survey, Journal of Experimental & Theoretical Artificial Intelligence (2024) 1–59. DOI:10.1080/0952813X.2024.2391778.
[16] A. Singh, M. D'Souza, A. Ebrahim, Conformance testing of ARINC 653 compliance for a safety critical RTOS using UPPAAL model checker, in: Proc. 36th Annual ACM Symposium on Applied Computing (SAC), Association for Computing Machinery, New York, NY, USA, 2021, pp. 1807–1814. DOI:10.1145/3412841.3442053.
[17] A. Bosio, S. Di Carlo, M. Rebaudengo, A. Savino, Toward the hardening of real-time operating systems, in: Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Austin, TX, USA, 2022, pp. 1–6. DOI:10.1109/DFT56152.2022.9962356.
[18] D. A. Santos et al., Hybrid Hardening Approach for a Fault-Tolerant RISC-V System-On-Chip, IEEE Transactions on Nuclear Science 71(8) (2024) 1722–1730. DOI:10.1109/TNS.2024.3406021.
[19] A. Kashtalian, S. Lysenko, O. Savenko, A. Nicheporuk, T. Sochor, V. Avsiyevych, Multi-computer malware detection systems with metamorphic functionality, Radioelectronic and Computer Systems 1 (2024) 152–175. DOI:10.32620/reks.2024.1.13.
[20] M. A. Shajahan, Fault Tolerance and Reliability in AUTOSAR Stack Development: Redundancy and Error Handling Strategies, Technology & Management Review 3(1) (2018) 27–45. URL: https://hal.science/hal-04561763v1/file/2018_4.pdf.
[21] N. K. Al-Salihi, Improvement of the Fault Tolerance in IoT Based Positioning Systems by Applying Redundancy in the Controller Layer. Department of Computer Science and Engineering, College of Engineering, University of Kurdistan Hewlêr (UKH), Erbil, Iraq, 2021. DOI:10.21123/bsj.2021.18.4.1303.
[22] O. Savenko, A. Sachenko, S. Lysenko, G. Markowsky, N. Vasylkiv, Botnet detection approach based on the distributed systems. International Journal of Computing 19(2) (2020) 190–198. DOI:10.47839/ijc.19.2.1761.
[23] A. Kashtalian, S. Lysenko, A. Sachenko, B. Savenko, O. Savenko, A. Nicheporuk. Evaluation criteria of centralization options in the architecture of multicomputer systems with traps and baits. Radioelectronic and Computer Systems, 1 (2025), 264-297. DOI:10.32620/reks.2025.1.18
[24] A. Kashtalian, S. Lysenko, T. Kysil, A. Sachenko, O. Savenko, B. Savenko, Method and Rules for Determining the Next Centralization Option in Multicomputer System Architecture, International Journal of Computing 24(1) (2025) 35-51. DOI:10.47839/ijc.24.1.3875
[25] K. Bobrovnikova, M. Kapustian, D. Denysiuk, Research of machine learning based methods for cyberattacks detection in the internet of things infrastructure, Computer Systems and Information Technologies 3 (2022) 110–115. DOI:10.31891/CSIT-2021-5-15
[26] D. Chicco, G. Jurman, The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification, BioData Mining 16 (1) (2023) 1-23. DOI:10.1186/s13040-023-00322-4
[27] D. Martiniuk, O. Lyhun, A. Drozd, O. Besedovskyi, Task optimisation in multiprocessor embedded systems. Computer Systems and Information Technologies 1 (2025) 124–135. DOI: 10.31891/csit-2025-1-14