

Hybrid method for protecting RTOS from failures and cyberattacks using compact Markov models

Piotr Gaj^{1,†}, Tomas Sochor^{2,†}, Maksym Korobchynskiy^{3,†}, Bohdan Savenko^{4,†} and Oleksandr Kozelskiy^{4,*}

¹ Silesian University of Technology, ul. Akademicka 2A, 44-100 Gliwice, Poland

² Prigo University College European Research University Vítězslava Nezvala 801/1 736 01 Havířov Czech Republic European Union

³ Yevhenii Bereznyak Military Academy, Kyiv, Ukraine

⁴ Khmelnytskyi National University, Khmelnytskyi, Instytutska street 11, 29016, Ukraine

Abstract

The paper presents a method of preventive restart of components in real-time operating systems, which combines a simplified Markov chain with a hybrid dual watchdog timer and is considered as a tool for enhancing cyber-resilience. Unlike traditional reactive strategies, which are activated only after a failure, the proposed approach provides a transition to a proactive model capable of counteracting both internal faults and external impacts, including denial-of-service (DoS) attacks, fault injections, or logic bombs. The use of compact Markov models makes it possible to quickly assess the risk of approaching a critical state without a significant increase in computational costs, which creates conditions for timely localization of the problem and initiation of the restart of a specific task, driver, or module at an early stage. This reduces the probability of losing the working context, minimizes the “window of vulnerability,” and decreases the risk of a complete reboot. The combination of software and hardware levels forms a multi-level protection mechanism: the software watchdog acts as the first line of defense, while the hardware one guarantees final recovery in case of deep lock-up or targeted attack. Such integration makes it possible to reduce the frequency of global restarts, limit downtime, and increase resilience to cyberattacks in resource-constrained conditions. The method does not require complex machine learning algorithms or cumbersome formal models, it is based on minimalist stochastic schemes and can be adapted to different platforms and threat scenarios. Experimental results demonstrated a 3.1 reduction in recovery time from 2.7 s to 0.84 s, approximately a 70% decrease in total system downtime from 4.5% to 1.37% over 5 minutes, and an average CPU load increase of only 1.4%, confirming the method’s high efficiency and low computational overhead. This ensures stability, predictability, and security of embedded and cyber-physical systems, where meeting timing constraints and continuity of task execution, as well as the ability to withstand modern cyber threats, are critically important. The obtained results demonstrate the practical effectiveness of the approach in industrial automation, robotics, automotive and medical systems, and confirm the prospects of the proposed solution for the development of next-generation cyber-defense systems.

Keywords

operating systems, RTOS, Markov chain, preventive restart, fault tolerance, cyber-physical systems anomaly detection, cyber resilience

1. Introduction

In real-time and embedded operating systems, there is a gradual increase in the number of competing tasks and the degree of component integration. This significantly complicates ensuring continuity of operation and compliance with strict timing constraints, known as deadlines. The problem becomes particularly critical in cyber-physical systems (CPS), where any error or delay in controlling physical processes can cause critical consequences — equipment failure, emergency

*AISSE-2025: The International Workshop on Applied Intelligent Security Systems in Law Enforcement, October, 30–31, 2025, Vinnytsia, Ukraine

^{1,†} Corresponding author.

[†] These authors contributed equally.

✉ piotr.gaj@polsl.pl (P. Gaj); tomas.sochor@osu.cz (T. Sochor); maks_kor@ukr.net (M. Korobchynskiy); bsavenko@khnmu.edu.ua (B. Savenko); oleksandr.kozelskiy@khnmu.edu.ua (O. Kozelskiy)

ORCID 0000-0002-2291-7341 (P. Gaj); 0000-0002-1704-1883 (T. Sochor); 0000-0001-8049-4730 (M. Korobchynskiy); 0000-0001-5647-9979 (B. Savenko); 0009-0002-7157-6499 (O. Kozelskiy)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

situations, or unacceptable delays in performing safety functions [1]. In the case of targeted cyberattacks, it creates an additional window of vulnerability that can be exploited by adversaries to destabilize or block the operation of the system. Modern reviews emphasize that with the growing complexity of RT environments, there is an increasing need for integrated software fault-tolerance mechanisms that take into account interactions with schedulers and mixed-criticality [34].

A key tool for improving reliability and security has traditionally been the watchdog timer. A standard hardware watchdog performs a full reset of the microcontroller after failures, thus implementing a purely reactive approach that often requires significant time and resource overhead [3] and does not take into account the factor of intentional impacts, for example, fault injections or logic attacks. The introduction of a combined hardware-software watchdog with the ability to locally (software-based) restart individual components before a fatal failure occurs [5,45] based on low-dimensional Markov chains potentially makes it possible to significantly increase fault tolerance, reduce overall system downtime [6], and decrease the effectiveness of denial-of-service (DoS) attacks or logic-based blocking of critical tasks.

The development of such solutions is complicated by limited computational resources, such as RAM capacity and clock frequency, and the requirement to meet real-time constraints [2]. In-depth system state analysis is usually based on computationally complex approaches, machine learning methods [7] or extended formal models, which are poorly suited for low-power microcontrollers. In contrast, simplified Markov chains with a small number of states make it possible to quickly, with minimal resource consumption [8], estimate the probability of the system transitioning into a critical state [4], including states that may be triggered by targeted attacks or abnormal behavior simulating failure. Within this approach, a predictive software watchdog can detect not only technical faults in advance but also suspicious activity and carry out a preventive restart of only the problematic task, driver, or module.

The aim of this work is the development and theoretical justification of a method for preventive restart of embedded RTOS components, based on a simplified Markov model of failure risk and integrated with a classical hardware watchdog timer. The proposed approach is intended to reduce the number of full system reboots, shorten downtime, and improve the overall reliability and cyber-resilience of resource-constrained CPS.

A comparative review of existing fault-tolerance methods in embedded real-time operating systems shows that the classical hardware watchdog guarantees recovery only post factum: the restart is initiated after an actual hang and is accompanied by a complete loss of operational context [9]. A software watchdog is potentially able to detect the failure of an individual process earlier, but its effectiveness entirely depends on the correct functioning of the RTOS kernel, and it can become a target for adversary bypass. Structural redundancy methods remain excessively resource-intensive for most low-power platforms, while the application of formal models or machine learning algorithms often exceeds the available computational capabilities of the microcontroller or requires significant verification efforts [10].

Under strict limitations on memory and clock frequency, it is advisable to combine a minimal hardware emergency restart mechanism with a software intelligence layer capable of proactively responding to symptoms of impending failure or signs of attack, and locally restoring the operation of the faulty component, minimizing the number of full restarts. The key task in such a hybrid scheme is the development of a lightweight criterion for evaluating the current system state sufficiently sensitive for timely detection of increasing execution delays, queue overflows, or other anomalies, including those caused by intentional actions, but not so coarse as to cause false positives. Simplified, low-dimensional Markov chains meet these requirements, since they allow real-time estimation of the probability of the system transitioning into a critical state with minimal computational overhead [11].

Accordingly, the scientific task is to develop a hybrid fault-tolerance and cyber-defense mechanism that integrates a hardware watchdog with a predictive software module based on a Markov model. This approach should ensure real-time assessment of the risk of a critical failure,

initiation of a soft restart of only the defective task or driver, maintenance of continuous and secure system operation in most non-fatal scenarios, and transfer of control to the hardware timer in cases of deep hang or successful attack. The expected result of implementing the proposed scheme is a significant reduction of downtime, a decrease in the number of full system reboots, and strengthened cyber-resilience in resource-constrained cyber-physical applications.

2. Similar works

In recent years, there has been a rapid development of approaches to failure prediction in real-time systems and embedded devices. Significant attention has been devoted to machine learning methods and sensor data analysis. In particular, Ukrainian researchers have explored the possibilities of using neural networks, decision trees, and other artificial intelligence algorithms for failure prediction in order to improve the reliability and efficiency of operation, for example, in smart homes [19]. Comprehensive reviews for real-time systems emphasize the appropriateness of combining model-based and data-driven approaches to failure prediction [34].

In such approaches, data are collected on system parameters, such as temperature, load, and response time, and hidden patterns that precede failure are sought. It has been proven that timely prediction of a fault before its actual occurrence makes it possible to initiate early self-diagnosis and protective measures, thereby preventing catastrophic consequences. For example, neural network-based methods are able to detect nonlinear dependencies in system behavior and predict overheating or other force majeure situations in advance [12,13]. At the same time, it is noted that purely reactive strategies, such as classical hardware resets in case of failure, or overly conservative approaches, when the system excessively reduces performance for the sake of reliability, no longer meet the needs of modern applications [14]. Additionally, emphasis is placed on the need for lightweight anomaly monitoring tools for resource-constrained CPS/IoT platforms [44]. A promising line for such constrained settings is deep echo state neural networks, a reservoir computing approach that offers fast training and efficient inference for cyber-threat detection in IoT-driven industrial control environments [51]. Complementary to runtime ML-based monitoring, evolutionary optimization, particularly genetic algorithms has proved effective for design-time architectural synthesis of security subsystems, including optimal sensor and coverage layout in perimeter protection, as demonstrated in PSCAD [47]. In SDN-IoT environments, systematic reviews document a spectrum of ML methods for detecting botnet anomalies and their practical limitations [38]. Beyond surveys, recent SDN work reports real-time ML-based DDoS detection integrated at the controller with automatic mitigation via flow-rule updates, achieving high detection rates while preserving throughput and acceptable latency [50]. In addition to detection, agent-based distributed defense mechanisms coordinate mitigation across nodes and demonstrate favorable performance against DDoS while maintaining acceptable throughput and response time [48].

Failure prediction is particularly relevant in the context of multitasking RTOS, where faults may manifest as task infinite loops, deadlocks, or missed critical deadlines. Research shows that the most common causes of embedded system hangs include memory corruption and being stuck in an infinite loop [15], hangs of external hardware components, mutual blocking of tasks due to incorrect use of mutexes [16], or situations where a high-priority task monopolizes the CPU and prevents other tasks from executing. At the network level, early detection of botnet activity is achieved by analyzing DNS traffic [31]. To counter evasion of detection, specialized DNS anti-evasion techniques are used [32]. In corporate networks, self-adaptive mechanisms increase resilience during botnet attacks by localizing and isolating nodes [33]. Reviews of botnet-driven DDoS attacks in IoT networks summarize the available datasets and methods, demonstrating the evolution of attack vectors [39].

Under such conditions, simply recording the fact of a failure is not enough, the system must be able to predict the approach of these undesirable states. Among modern approaches to failure prediction, model-based models [17] and data-driven models [18] can be distinguished. The first are

built on the use of analytical models of system components and the calculation of residuals, that is, the deviations between the measured parameters and the nominal model. Their advantage lies in the consideration of the physical properties of processes, but creating an accurate model of a complex RTOS is a non-trivial task. The second approach is based on the analysis of large volumes of sensor and telemetry data; statistical and ML algorithms study system behavior patterns and detect signs of impending failure without an explicit physical model. Data-driven methods work well for complex systems with many variables but require sufficiently informative data to train models and detect correlations that precede failures [20]. In security scenarios, this is accompanied by the transition to autonomous, privacy-oriented IDS with distributed decision-making [40]. In the context of SDN-IoT, a spectrum of ML methods and key challenges of deployment at the network edge have been outlined [41].

In the practice of failure prediction, both approaches are often combined: relying on expert models and machine learning algorithms to achieve maximum accuracy. Such integration is also useful for protection tasks, the combination of knowledge about the system architecture with telemetry data makes it possible to increase resilience to both accidental technical faults and deliberate cyberattacks.

The transition from reactive to proactive monitoring means that the system continuously tracks key parameters and the state of components in order to detect deviations in advance. This approach is implemented in the concept of Prognostics and Health Management (PHM), or prognostic health systems [21]. Within PHM, each node or process of a real-time system can have built-in sensors and control agents that evaluate its condition online. For example, response times of tasks, execution cycle frequency [22], resource consumption, temperature modes [23], supply voltage, and so on [24] can be monitored. Modern PHM surveys highlight the growing role of deep learning and the need to adapt methods to resource-constrained CPS/RTOS platforms [42]. Systematizations of ML-PHM emphasize the combination of condition monitoring with degradation prediction while taking into account real-time constraints [43].

Based on these data, algorithms perform diagnostics, that is, the detection of already existing deviations and prognostics, meaning the prediction of the time to failure or the probability of a fault. The literature provides examples where the implementation of comprehensive monitoring made it possible not only to promptly detect failures but also to predict their occurrence several cycles before the actual failure, which enabled the planning of recovery measures without stopping the system. This is particularly important for critical real-time systems, including avionics, automotive controllers, and medical devices, where unexpected failure is unacceptable [25]. Taking into account cyber threats, anomaly monitoring further limits the space for intentional impacts in critical domains [44].

Proactive monitoring minimizes downtime and ensures safe degraded functioning: the system can switch in advance to a backup module or perform recovery procedures if the prognostic module signals an increased risk of failure. One of the tools of proactive diagnostics is intelligent Watchdog controllers, special mechanisms that monitor system operability and initiate recovery in case of abnormal behavior [26]. Software-defined watchdog mechanisms make it possible to formalize deadlines and response policies at the level of event-control models [37]. “Smart” hardware detectors at the processor level increase coverage of transient and permanent faults without significant overhead [36]. Heterogeneous integration with FPGA reduces monitoring overhead without violating task deadlines [35].

The classical Watchdog timer is a hardware or software timer that is regularly reset by the executing program; if the program hangs and does not reload the timer in time, the timer generates a signal for reset or another emergency action. Thus, the Watchdog traditionally serves as the last line of defense of the system against hangs and unacceptable delays. However, modern trends go beyond simple hardware reset, researchers propose multi-level Watchdog architectures that include both software and hardware layers of control. In distributed IoT scenarios, decentralized IDS with alert correlation are advisable, which strengthens resilience to software supply chain attacks [40].

Additionally, in related work on the security of multi-computer systems, a similar principle of multi-level self-healing is proposed. In particular, systems for detecting metamorphic malware [27] and hidden steganographic modifications [28] demonstrate how intelligent agents can autonomously respond to anomalies without fully rebooting the server. Additional studies on the criteria of centralization of traps and honeypots in multi-computer networks [29], analysis of low-level DDoS attacks using the feature of traffic self-similarity [30], adaptive entropy metrics [46] and self-adaptive mechanisms of corporate network resilience against botnet attacks emphasize the effectiveness of the proactive, multi-level approach: first locally isolate or restart the affected node/process, and only then, if necessary, employ more drastic hardware measures. Complementary to runtime monitoring and recovery, evolutionary design-time optimization of security architectures has proved effective: genetic algorithms deliver near-Pareto-optimal sensor layouts and parameterizations for perimeter protection, as demonstrated in PSCAD [47], and have been generalized to distributed security systems with multi-criteria objectives and deployment constraints [49]. These results logically resonate with the development of intelligent Watchdog controllers in embedded systems, which combine selective recovery for fault or attack localization with the ultimate hardware line of defense.

3. Methodology and methods

3.1. Mathematical model and method description

The proposed method consists of combining the classical hardware watchdog timer as a hard layer of protection with a predictive software layer, referred to as the Software Watchdog, which uses a simplified Markov model to detect a probable failure before it occurs and to perform local recovery, that is, a soft reset, of problematic components. If the software restart does not help or the hang involves the kernel as a whole, the hardware watchdog timer, acting as the second stage, executes a full system reset. Such a two-stage mechanism reduces the number of full resets and shortens downtime, which is particularly important in CPS with strict timing requirements and limited resources. The method provides a discrete assessment of the system state at time moments $t = k \cdot \Delta t = k$, where Δt is the execution period of the Predictive Watchdog Task in the RTOS. At each step, the system is in one of a finite set of states, for example, $\{N, W, C, F\}$, corresponding respectively to “Normal”, “Warning”, “Critical”, “Fail”), and transitions between these states are described by a low-dimensional Markov chain.

Let $S = \{N, W, C, F\}$ be a finite set of states. At each tick $t^k = k\Delta t$ the system (or the “Predictive Watchdog Task,” which reflects its health) is in one of the states $x(k) \in S$. For simplicity, assume that, N (Normal) — the system is in the normal range (queues are not overfilled, heartbeat signals from all tasks arrive on time), W (Warning) — degradation of indicators is detected (increased load, delayed heartbeat signals), but not yet critical, C (Critical) — the risk of imminent failure is high (significant delays, frequent missed deadlines, a task stops sending heartbeat signals), F (Fail) — actual hang/failure (in the model, this is an absorbing state or an indicator that a global reset has occurred).

In implementation, more or fewer states may be defined, for example, “Recovery,” “Standby”, but 4–5 active states are already sufficient for most simplified scenarios.

We assume that $x(k+1)$ depends only on $x(k)$ and not on the states at earlier moments (the Markov property). This can be written as:

$$P = \begin{pmatrix} P_{NN} & P_{NW} & P_{NC} & P_{NF} \\ P_{WN} & P_{WW} & P_{WC} & P_{WF} \\ P_{CN} & P_{CW} & P_{CC} & P_{CF} \\ P_{FN} & P_{FW} & P_{FC} & P_{FF} \end{pmatrix}, \sum_{j \in S} P_{ij} = 1, \forall i \in S, \quad (1)$$

where the indices i and j denote the system states N (Normal), W (Warning), C (Critical), and F (Fail).

If F (Fail) is considered as an absorbing state, then the following holds $P_{FF} = 1$ i $P_{Fj} = 0$ for $j \neq F$. In this case, reaching F is interpreted as an irreversible hang. However, if in the physical system a hardware reset is executed after F (Fail), then it is logical to assume that the system state returns to N (Normal) — but only after reboot.

For estimating the elements p_{ij} , a simple frequency counting method is applied. Suppose that at the k -th step the system is in state $i = x(k)$, and at the $x(k+1)$ -th step it is in state $j = x(k+1)$. Then the counter $n_{(i \rightarrow j)}$ is incremented by 1:

$$n_{(i \rightarrow j)} := n_{(i \rightarrow j)} + 1 \quad (2)$$

Additionally, we track $n_i = \sum_{m \in S} n_{(i \rightarrow m)}$, that is, how many times the system was in state i . Thus, we have:

$$p_{ij} \approx \frac{n_{(i \rightarrow j)}}{\sum_{m \in S} n_{(i \rightarrow m)}} = \frac{n_{(i \rightarrow j)}}{n_i} \quad (3)$$

This ratio provides an approximation of the transition probability $i \rightarrow j$. It allows updating P , without significant additional CPU overhead. If the system runs for a long time, a finite sliding window [21] or exponential smoothing can be applied to avoid the accumulation of outdated statistics. For example, with a sliding window of size L , for instance, 50–100 steps, only the most recent L transition observations are retained.

At each time step Δt the Predictive Watchdog Task in the RTOS collects metrics as queue load, delays, and missed deadlines, and, according to threshold logic, assigns the system to one of the states:

1. N (Normal): for example, when the maximum queue fill is $\leq 50\%$ and all heartbeat signals from critical tasks are timely;
2. W (Warning): partial or periodic overload above permissible limits, but no sustained deadline misses; e.g., max fill in the 50–80% range or at least one delayed heartbeat signal;
3. C (Critical): queue overflow above 80% or systematic deadline misses by one or more critical tasks;
4. F (Fail): heartbeat signals are missing for one or more critical tasks over two–three consecutive cycles, or when a fatal error occurs.

Thus, the definitions of N, W, C, F depend on the application scenario and the established threshold values. Such thresholds are determined empirically or based on the specifics of the application.

The transition probabilities between the states in the Predictive Watchdog are dynamic and can be adjusted in real time depending on changes in key system metrics. The main influencing factors are queue occupancy, delays in system activity signals, memory usage, and other system performance parameters. For example, if the queue occupancy level exceeds a certain threshold, this may indicate an approach to a critical state, increasing the probability of transition:

$$p(W \rightarrow C) = f^1(Q, H, M) \quad (4)$$

Similarly, if the delay of system activity signals lasts longer than expected, the probability of transition to the failure state increases:

$$p(C \rightarrow D) = f^2(H, T) \quad (5)$$

If after a Soft Reset the system quickly stabilizes, the probability increases that:

$$p(C \rightarrow N) = f^3(R), \quad (6)$$

where Q is the queue occupancy. H is the delay of activity signals. M is the memory usage. T is the response waiting time. R is the number of successful software resets, f^1, f^2, f^3 is the empirical or statistically identified functions that map current system metrics into the probability of the corresponding transition, $p(\dots)$ is the the probability of the Predictive Watchdog making the given transition.

The time to failure is modeled as a random variable that follows an exponential distribution, since system failures in real time are usually independent events occurring at a constant rate. The exponential distribution is used to estimate the probability that the system will remain operational during a certain period, i.e., that a failure will not occur before a given moment. Its key property is memorylessness, which means that the probability of failure in the future does not depend on how long the system has already been operating without faults. The failure rate is described by parameter λ , which determines the average failure frequency, and the probability of system survival during time t is calculated by the formula:

$$P(T > t) = e^{-\lambda t}, \quad (7)$$

here T is the time to failure. λ is the failure rate. The probability of transition to state W or C is defined as:

$$p(N \rightarrow W) = 1 - P(T > t) \quad (8)$$

If the calculated probability value exceeds a certain threshold, the watchdog timer switches to proactive response, triggering a software reset or preparing a hardware reset.

Let us describe failure prediction and the two-stage protection. If it is known that the current state is $x(k) = i$, then $P(x(k+1) = Fail / x(k) = i) = p_{iF}$. If this probability p_{iF} exceeds the predefined threshold θ , we choose θ as the failure probability threshold. When:

$$p_{iF} > 0 [P^2]_{iF} > 0 \theta \in [0.5; 0.8] \text{ (typically)} \quad (9)$$

the system is considered close to a critical state, and the predictive layer decides to perform a preventive software reset of the corresponding component, for example, by deleting the problematic task and recreating it, without waiting for an actual hang.

If it is necessary to consider the probability of reaching F (Fail) after several steps, the product of the matrix P^r is used:

$$P^r = \underbrace{P \times P \times \dots \times P}_r, \quad (10)$$

where element $[P^r]_{iF}$ gives the probability of being in state F after r steps, given that currently $x(k) = i$. In practical microcontroller systems, usually 1–5 steps of prediction are sufficient due to resource constraints and short periods Δt .

In the proposed two-stage mechanism, if the Markov chain analysis indicates a high probability $P \geq \theta$ of rapid transition to F , the predictive layer locally recovers the problematic task, performing a soft reset: $TaskDel(taskXHandle) \rightarrow ReInit() \rightarrow TCreate()$.

If the local restart does not help, that is, when state C persists, the software layer does not call $feed(\dots)$ of the hardware watchdog timer. After time T_{HW} , which represents the hardware WDT timeout, a full system reset occurs. In the Markov model, this corresponds to the transition $C \rightarrow F$, which in practice means microcontroller reboot and return to the initial state N .

Thus, if the deviation is not too deep, the soft reset allows avoiding a global reboot and reduces downtime. The hardware WDT remains as the last safeguard if the system escapes software control. Formally: if at step k the soft reset procedure is launched, then at step $k+1$ we artificially set $x(k+1)=N$, provided that the reset was successful. For accuracy, we introduce a “Recovery” state (R) and modify the transition matrix.

Since we have $\Delta t \ll T_{HW}$, the software layer has several attempts, for example, 10–20 cycles with a step interval of Δt to perform a software reset before the hardware watchdog timeout expires. If the Predictive Watchdog Task concludes that the system has returned to Normal or to Warning without progress to Critical, it invokes HW_WDT_Feed(), continuing normal operation. If the local failure is not eliminated, the Predictive Watchdog Task deliberately does not send the control signal to the hardware watchdog, which causes it to trigger and initiates a full system reboot.

If $x(k)=C$, the soft reset is ineffective \Rightarrow no control signal is sent to the HW WDT \Rightarrow global reset.

In summary, the mathematical scheme is defined by the following relations. The system state $x(k)$ is a discrete variable in a finite space S . The transition laws are determined by the matrix P , which is updated incrementally or within a sliding window:

$$p_{ij} = \frac{n_{(i \rightarrow j)}}{n_i}, \quad (11)$$

where p_{ij} is the probability of the system transitioning from state i to state j in one step, $n_{(i \rightarrow j)}$ is the number of observed direct transitions from state i to state j within the current sampling window, n_i is the total number of times state i was recorded as the origin during that same window. The probability of failure in one step is $P(\text{failure})$, which represents transitions from the Critical state to the Fail state or from the Warning state to the Fail state. If the threshold θ is exceeded, preventive actions such as a soft reset are triggered. If necessary, a prediction for k steps ahead is made using the transition matrix P . A soft reset is equivalent to forcing the system back from a dangerous state to the Normal or Recovery state. A full reset via the hardware watchdog occurs if the system signals a critical problem but the software reset does not succeed.

3.2. Markov Chain Predictive Watchdog Task Algorithm

The algorithm of the Markov Chain Predictive Watchdog Task is presented in Figure 1 as a block diagram showing the sequence of actions from initialization to the choice of recovery strategy. It illustrates the monitoring task logic in FreeRTOS:

1. Start system monitoring.
2. Analyze the process.
3. Check, using the Markov model, whether the task shows prerequisites for a hang.
4. If yes – perform a task restart or recovery.
5. If after the restart the problem is resolved – terminate the watchdog task.
6. If after the restart the problem is not resolved – restart the system.
7. After the system restart, the monitoring cycle begins again.

Thus, the proposed method combines hardware and software levels of control, using a simplified Markov model for predicting failures and timely triggering of a local restart. This approach makes it possible to react in advance to symptoms of an impending critical state, reduce the number of global restarts, and shorten downtime. The two-stage strategy a preventive soft reset and a guaranteed hardware reset ensures a balance between proactivity and reliability, making the method an effective solution for embedded and cyber-physical systems with strict timing constraints and limited resources.

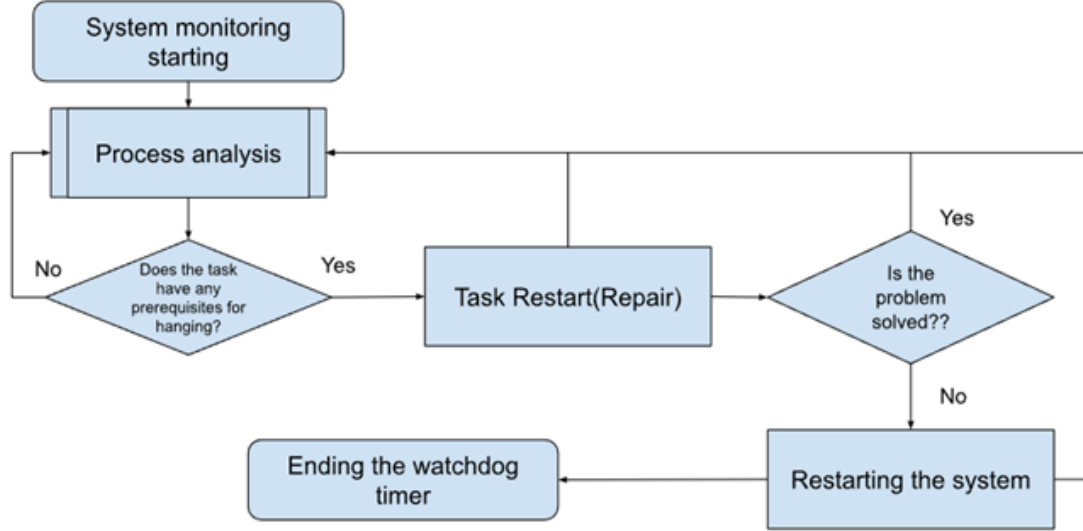


Figure 1: Algorithm of the Markov Chain Predictive Watchdog Task.

3.3. Integration with IDS and PHM for deeper cyber-resilience

The proposed PredictiveWDT can be augmented with external signals from intrusion detection systems (IDS) and from prognostics & health management (PHM) platforms to obtain a deeper assessment of cyber-resilience. IDS contributes anomaly scores derived from network or host telemetry, and PHM contributes health-risk scores derived from sensor and operational data. These signals are fused with the Markov estimator to form a unified risk used by the watchdog decision logic:

$$R_k = \alpha P_k(C \rightarrow F) \beta A_k^{IDS} + \gamma H_k^{PHM}, \quad (12)$$

where k is the sampling step with period Δt , $\alpha P_k(C \rightarrow F)$ is the estimated one-step probability of transition from Critical to Fail at step k , $A_k^{IDS} \in [0,1]$ is the normalized IDS anomaly score at step k , $H_k^{PHM} \in [0,1]$ is the normalized PHM health-risk score at step k , $\alpha, \beta, \gamma \geq 0$ are the weights with unit sum that set the contribution of each term. A local recovery is triggered when $R_k \geq \theta$, while the hardware watchdog remains the final recovery layer. Integration is lightweight: subscribe to IDS and PHM events, normalize incoming scores to the unit range, update R_k once per polling period, and reuse the existing queue, semaphore, timing, and task-control primitives of the PredictiveWDT. This fusion improves early detection of both technical faults and cyber anomalies without modifying the Markov core or the state machine.

4. Experiments

The research was conducted on a real hardware platform STM32F4 under the control of the FreeRTOS real-time operating system. The selected hardware platform belongs to the class of microcontrollers with the ARM Cortex-M4 architecture, widely used in embedded systems, which combines sufficient performance with strict limitations on power consumption and computational resources. FreeRTOS, in turn, provides typical multitasking mechanisms, such as the scheduler, message queues, and semaphores, which allow the reproduction of scenarios of critical application systems. Such a combination of hardware and software components makes it possible to model realistic operating conditions of resource-constrained cyber-physical systems and at the same time experimentally verify the effectiveness of the proposed fault-tolerance mechanisms.

The test CPS on which the experiments were conducted consisted of the following components:

1. The central processing unit, performing task scheduling;
2. The SPI peripheral, on which overload scenarios were modeled;
3. The UART interface for logging and diagnostics;
4. GPIO events, which simulated external sensor signals and generated asynchronous interrupts;
5. The PredictiveWDT software module, integrated into FreeRTOS for predictive diagnostics and recovery.

5. Results

In the first series of experiments, the operation of the system was studied using only the classical hardware watchdog. For this purpose, a hang of TaskA, which serviced the SPI peripheral, was simulated, as well as a mutual blocking or deadlock in TaskA. In the case when TaskA stopped responding, the hardware WDT was triggered only after the expiration of the preset timeout of 2.0 seconds. After that, a full microcontroller reset occurred with re-initialization of all system components, which took another 0.6–0.8 seconds. Thus, the total recovery time averaged about 2.7 seconds. The operation of the system in the first series of experiments, which involved only the hardware watchdog timer, is illustrated in Figure 2, left part, where the event logging is shown in simple_watchdog.log.

simple_watchdog.log	predictive_watchdog.log
1 [0.000] [LOGGER] System boot (HW WDT only)	1 [0.000] [LOGGER] System boot (HW WDT + PredictiveWDT)
2 [0.010] [TaskA] Init complete	2 [0.010] [TaskA] Init done
3 [0.020] [TaskB] Init complete	3 [0.020] [TaskB] Init done
4 [0.030] [TaskC] Init complete	4 [0.029] [TaskC] Init done
5 [0.050] [LOGGER] State: NORMAL, Queue usage=0%	5 [0.051] [LOGGER] Markov state = N, P(N)=1.0, Q usage=0%
6 [0.096] [TaskA] Heart-beat OK	6 [0.095] [TaskA] Heart-beat OK
7 [0.152] [TaskB] Sensor reading: 310	7 [0.202] [TaskC] Processing cycle, Q usage=5%
8 ...	8 [0.507] [TaskC] Done, CPU=25%
9 [36.237] [LOGGER] << CRITICAL >> No response, queue usage=95%	9 ...
10 [36.307] [HW WDT] Triggering HARD RESET	10 [53.617] [LOGGER] Queue usage=30%, State update => CRITICAL (P(C)>0.7)
11 --- DOWNTIME from 36.307s to 38.996s ---	11 [58.259] [PredictiveWDT] SoftReset(TaskA)
12 [38.996] [LOGGER] Reboot after WDT reset	12 --- partial downtime from 58.259s to 58.611s (~0.35s) ---
13 [39.268] [TaskA] Init complete	13 [58.259] [TaskA] Re-initialized
14 [39.686] [TaskB] Init complete	14 [58.612] [LOGGER] State => N, queue usage=10%
15 [40.453] [TaskC] Init complete	15 ...
16 [40.280] [LOGGER] State: NORMAL, Queue usage=0%	16 [66.366] [LOGGER] Queue usage=85%, State update => CRITICAL (P(C)>0.72)
17 [41.555] [TaskA] Heart-beat OK	17 [66.466] [PredictiveWDT] SoftReset(TaskA)
18 ...	18 --- partial downtime 66.466s to 66.817s (~0.35s) ---
19 [59.529] [LOGGER] << CRITICAL >> No response, queue usage=88%	19 [67.477] [TaskA] Re-initialized, drivers reinit
20 [59.641] [HW WDT] Triggering HARD RESET	20 [67.570] [LOGGER] State => NORMAL, queue usage=5%
21 --- DOWNTIME from 59.641s to 62.307s ---	21 ...
22 [62.307] [LOGGER] Reboot after WDT reset	22 [90.448] [TaskB] Sensor reading=599
23 [62.936] [TaskA] Init done	23 [90.510] [LOGGER] Queue usage=38%, State update => CRITICAL (P(C)>0.72)
24 [62.946] [TaskB] Init done	24 [90.609] [PredictiveWDT] SoftReset(TaskB) // this time we reset TaskB
25 [62.949] [TaskC] Init done	25 --- partial downtime 90.609 to 90.951 (~0.35s) ---
26 [63.030] [LOGGER] State: NORMAL, Queue usage=0%	26 [90.951] [TaskB] Re-initialized
27 ...	27 [90.961] [LOGGER] State => N, queue usage=10%
28 [102.732] [LOGGER] << CRITICAL >> No response, queue usage=93%	28 ...
29 [102.854] [HW WDT] Triggering HARD RESET	29 [91.448] [TaskA] Sensor reading=1002
30 --- DOWNTIME from 102.854s to 104.805s ---	30 [93.758] [LOGGER] Queue usage=82%, State update => CRITICAL (P(C)>0.71)
31 [104.805] [LOGGER] System reboot	31 [95.075] [PredictiveWDT] SoftReset(TaskA)
32 [105.520] [TaskA] Init done	32 --- downtime 95.075 to 95.421 (~0.35s) ---
33 [105.847] [TaskB] Init done	33 [95.175] [TaskA] Re-initialized
34 [105.534] [TaskC] Init done	34 [95.270] [LOGGER] Markov => N, queue usage=8%
35 [107.428] [LOGGER] State: NORMAL, Queue usage=0%	35 ...
36 ...	36 [210.882] [LOGGER] Queue usage=95, State update => CRITICAL (P(C)>0.78)
37 [198.940] [LOGGER] << CRITICAL >> No response, queue usage=90%	37 [215.999] [PredictiveWDT] CRITICAL lock-up, SoftReset fails
38 [198.951] [HW WDT] Triggering HARD RESET	38 [227.121] [HW WDT] Triggering HARD RESET
39 --- DOWNTIME from 198.951s to 201.931s ---	39 --- downtime from 227.121 to 229.833 (~2.7s) ---
40 [201.931] [LOGGER] System reboot	40 [227.241] [LOGGER] System reboot
41 [201.985] [TaskA] Init done	41 [229.833] [TaskA] Init done
42 [196.930] [TaskB] Init done	42 [229.859] [TaskB] Init done
43 [196.960] [TaskC] Init done	43 [229.861] [TaskC] Init done
44 [200.302] [LOGGER] State: NORMAL, Queue usage=0%	44 [230.476] [LOGGER] State => N, queue usage=0%
45 ...	45 [230.671] [TaskA] Heart-beat OK
46 [220.170] [LOGGER] << CRITICAL >> No response, queue usage=85%	46 ...
47 [222.675] [HW WDT] Triggering HARD RESET	47 [LOGGER] *** End of 5-min run ***
48 --- DOWNTIME from 222.675s to 225.383s (~2.7s) ---	48 [LOGGER] Soft Resets=4 (58.259,66.466,90.609,95.075) ~0.35s each => 1.4s total
49 [225.383] [LOGGER] System reboot	49 [LOGGER] Hard Resets=1 (227.121) => 2.7s
50 [225.680] [TaskA] Init done	50 [LOGGER] Total downtime=4.2s (~1.40%)
51 [225.708] [TaskB] Init done	
52 [225.722] [TaskC] Init done	
53 [226.558] [LOGGER] State: NORMAL, Queue usage=0%	
54 ...	
55 [LOGGER] *** End of 5-min run ***	
56 [LOGGER] Hard Resets=5(36,59,102,198,227,222) => 2.7s	
57 [LOGGER] Total downtime=5*2.7=13.5s (~4.5%)	

Figure 2: System logging with and without Soft Watchdog, and with it over a 5-minute period under random faults.

Total downtime:

$$5 \times 2.7 = 13.5 \text{ s} \quad (13)$$

Percentage of downtime over 5 minutes (300 s):

$$\frac{13.5}{300} \times 100 \% = 4.5 \% \quad (14)$$

In the second series of tests, the operation of the system with the software watchdog timer was studied. The same actions were carried out. TaskA could hang due to simulated SPI peripheral overload or simulated mutual blocking. The system with PredictiveWDT detected the critical state on average within 0.11–0.13 s, since the polling period was 100 ms. Usually, 1–2 cycles were sufficient to change the state estimate from Normal/Warning to Critical. After transitioning to Critical with a high probability of transition to F (Fail), in particular when the thresholds of critical load were exceeded, PredictiveWDT performed a software reset of TaskA, reloading the associated drivers. This operation took about 0.3–0.4 s, so the total return time to Normal did not exceed 0.5 s. There was also a simulation of a mutual blocking in TaskA, where the deadlock was so deep that even a software reset with task recreation did not solve the problem. In this case, the deep lock occurred due to an error in SPI peripheral interaction, when one of the tasks held a semaphore for SPI access while the DMA transmission remained stuck in a waiting state. As a result, other tasks waiting for this resource could not proceed, and the FreeRTOS kernel effectively lost the ability to perform context switching. Under such conditions, a software reset of an individual task is ineffective because the blocking occurs at the scheduler level. This defines the applicability limit of the PredictiveWDT method: when the scheduler or peripheral subsystem becomes unresponsive, full recovery can be achieved only through the activation of the hardware watchdog timer. The system operation can be seen in Figure 2, right part, where the event logging is shown in predictive_watchdog.log.

In a total of 5 attempts, only in one case did the system remain in a locally non-functional state. In this case, PredictiveWDT deliberately did not send a control signal to the hardware watchdog timer, which led to its triggering and a full system reset. The other 4 faults were corrected precisely through the local restart. Thus, the proportion of global restarts was $100\%/5 \times 1 = 20\%$, as shown in Figure 2 on the right. With a purely reactive approach, that is, in the absence of a software watchdog timer, the real-time scheduling method in similar scenarios always results in 100% global restarts, since the system waits only for the hardware watchdog timer.

Total downtime:

$$(4 \times 0.35) + (1 \times 3) = 1.4 + 2.7 = 4.1 \text{ s} \quad (15)$$

Percentage of downtime over 5 minutes (300 s):

$$\frac{4.1}{300} \times 100 \% = \sim 1.36666 \% \quad (16)$$

Thus, the reduction in downtime is:

$$4.5 \% - 1.36666 \% = 4.5 \% - 1.36666 \% = 3.13334 \% \quad (17)$$

Let us calculate in relative terms the reduction of downtime between the first and the second test:

$$\frac{4.5 - 1.36666}{4.5} \times 100 \% = 69.62978 \% \quad (18)$$

Regarding CPU load, the worst measurement in FreeRTOS+Trace recorded an additional 1.6% load when TaskA generated a large number of events and PredictiveWDT simultaneously processed transition counters in the matrix. On average, this load remained below 1.4%, i.e., it did not noticeably affect the execution of the remaining tasks considering that the processor core runs at 180 MHz.

Also the concise comparison Table 1 below summarizes response time and steady-state CPU load across representative fault-tolerance methods, providing context for the proposed hybrid approach [45].

Table 1

Response time vs CPU load across fault-tolerance methods

Fault-tolerant methods.	Steady-state CPU overhead (%)	Description
Hardware watchdog timer	0–0.2	Last-resort global reset, reactive, no early localization
Soft watchdog with Markov chain (proposed)	1–3	Predictive local restart, HW WDT as final safeguard
Redundancy	50–200	Immediate masking or fast takeover, high hardware and synchronization cost
Machine-learning–based methods	20–80	Feature extraction and inference on the target MCU, higher compute and energy demand
Formal methods and verification	~0	Proves absence of classes of defects before deployment, limited use for online detection

It should be noted that the proposed approach is OS-independent. Only the monitoring and recovery bindings are adapted, while the Markov-based decision logic, thresholds, sampling period, and transition matrix remain unchanged. On Zephyr, a watchdog task can selectively use kernel queues, semaphores, and thread synchronization primitives, as well as restart tasks at the task level using standard thread management services; on VxWorks, the same functionality is implemented through message queues, semaphore facilities, task management services, and a native watchdog library for eventual recovery. The necessary changes are limited to replacing the queue/semaphore checking, delay/synchronization, task management, and hardware watchdog interfaces with their platform equivalents. With a polling period of 100 ms, the computational overhead remains low, approximately one to two percent on Cortex-M-class target systems, and comparable behavior is expected on Zephyr and VxWorks under similar configurations.

The overall conclusions from the experiment show that the proposed strategy thanks to the regular soft restart of problematic components provides a significant reduction in average downtime five to eight times less compared to a full reboot. At the same time, the guaranteed hard protection method against fatal failures is preserved. The execution time itself did not increase significantly in percentage terms. Therefore, the combination of lightweight stochastic risk assessment based on the Markov model and the two-stage software–hardware mechanism contributes to a noticeable improvement in reliability and recovery speed in CPS, while not requiring significant computational or memory resources.

6. Conclusions

The paper presents a two-stage method of cyber-resilience and fault tolerance for embedded and cyber-physical systems, which combines a hardware watchdog timer as the last line of defense against fatal failures with a dedicated software watchdog module that operates on a simple Markov model. This approach makes it possible to proactively detect a probable fault or anomaly caused either by a technical malfunction or by malicious interference and to perform a local software reset of the affected components. The main idea is not to wait for an actual system hang, but to evaluate the risk of transition to F (Fail) by analyzing current operational conditions using a low-dimensional Markov chain, and to initiate recovery measures in advance. In cases where preventive actions are ineffective, the hardware watchdog is deliberately allowed to execute a full system reset, ensuring the last line of defense against deep lockups or kernel-level cyberattacks. The conducted experimental studies confirmed the following key advantages:

1. The system downtime was significantly reduced thanks to local soft restarts, and the recovery time decreased by a factor of 3.1 compared to a full system reboot from 2.7 to 0.84 seconds.
2. Reduction in the frequency of global restarts: in a typical scenario, about 80% of potential failures or anomalies were eliminated locally, without hardware resets.
3. The computational overhead remained low, as the CPU load did not exceed 1–2%, which makes the proposed approach suitable for resource-constrained microcontrollers, even in high-risk attack conditions.

With the same number of faults over 5 minutes, the use of the Predictive Watchdog reduced the total downtime percentage from 4.5% to 1.36666%. In relative terms, this represents approximately a 69.62978% reduction in downtime.

The proposed method demonstrates high practical feasibility in the domain of embedded and real-time cyber-physical systems, where strict timing constraints must be combined with guaranteed fault elimination and protection against malicious impacts. The use of a lightweight Markov model ensures fast analysis without complex formal or ML-based solutions, while the two-stage design minimizes the number of hardware reboots, still retaining the hardware watchdog timer as the ultimate protective mechanism. Thus, the combination of preventive software recovery based on low-dimensional Markov chains with hard hardware resets creates a balance between the reactive nature of classical watchdog timers and dynamic approaches to cyber defense.

The research results confirm the effectiveness of applying this scheme across diverse fields — from industrial automation and robotics to automotive and medical systems — where it is critically important not only to maintain continuity and reliability of operation with minimal downtime, but also to ensure the ability of systems to withstand modern cyber threats. Future work will focus on integrating the PredictiveWDT with intrusion detection systems (IDS) and prognostics and health management (PHM) frameworks to enable unified monitoring of both system faults and cyber anomalies, thereby providing a more comprehensive assessment of cyber-resilience.

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] J. Navarro, Introduction to system reliability theory, Springer, New York, NY, 2021, 181 pp. doi:10.1007/978-3-030-86953-3.
- [3] A. Gangolli, Q. H. Mahmoud, A. Azim, A systematic review of fault injection attacks on IoT systems, *Electronics* 11 (2022) 2023. doi:10.3390/electronics11132023 .
- [4] T. Zaytseva, L. Kravtsova, N. Kaminskaya, Markov processes in researching the probability of cyber attacks on marine vessels, *Journal of Information Technologies in Education* 52 (2023) 20–32. doi:10.14308/ite000763.
- [5] R. Mehalaine, et al., Watchdog timer for fault tolerance in embedded systems, *J. Intelligent Embedded Systems Applications* (2024). doi:10.18280/jesa.570619.
- [6] Y. Yigit, L. Maglaras, M. A. Ferrag, N. Moradpoor, G. Lambropoulos, Reliability analysis of fault-tolerant memory systems, in: *Proceedings of the 8th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, IEEE, Piraeus, Greece, 2023, pp. 1–6. doi:10.1109/SEEDA-CECNSM61561.2023.10470763.
- [7] A. Elhanashi, P. Dini, S. Saponara, Q. Zheng, Advancements in TinyML: applications, limitations, and impact on IoT devices, *Electronics* 13 (2024) 3562. doi:10.3390/electronics13173562.
- [8] F. Michel, M. Siegle, Formal error bounds for the state space reduction of Markov chains, *Performance Evaluation* 167 (2025) 102464. doi:10.1016/j.peva.2024.102464.
- [9] M. A. Solouki, S. Angizi, M. Violante, Dependability in embedded systems: a survey of fault-tolerance methods and software-based mitigation techniques, *IEEE Access* 12 (2024) 1–35. doi:10.1109/ACCESS.2024.3509633.
- [10] L. Bariah, Q. Zhao, H. Zou, et al., Tiny machine learning and on-device inference: a survey of applications, challenges, and future directions, *Sensors* 25 (2025) 3191. doi:10.3390/s25103191
- [11] M. Kobielnik, W. Kempa, On the time to buffer overflow in a queueing model with a general independent input stream and working vacations, *Sensors* 21 (2021) 5507. doi:10.3390/s21165507.
- [12] X. Fan, W. Zhang, H. Qi, X. Zhou, Accurate battery temperature prediction using self-training neural networks within embedded system, *Energy* 313 (2024) 134031. doi:10.1016/j.energy.2024.134031.
- [13] S. Maity, A. Majumder, R. Roy, A. Hota, S. Dey, Harnessing machine learning in dynamic thermal management in embedded CPU–GPU platforms, *ACM Transactions on Design Automation of Electronic Systems* 30 (2025) 19:1–19:32. doi:10.1145/3708890.
- [14] S. Rana, AI-driven fault detection and predictive maintenance in electrical power systems: a systematic review of data-driven approaches, digital twins, and self-healing grids, *American Journal of Advanced Technology and Engineering Solutions* 1 (2025) 258–289. doi:10.63125/4p25x993.
- [15] E. Magliano, A. Savino, S. Di Carlo, Real-time embedded system fault injector framework for micro-architectural state-based reliability assessment, *Journal of Electronic Testing* 41 (2025) 193–208. doi:10.1007/s10836-025-06170-w.
- [16] R. Aalund, V. P. Paglioni, Enhancing reliability in embedded systems hardware: a literature survey, *IEEE Access* (2025) 1–1. doi:10.1109/ACCESS.2025.3534138.
- [17] P. Mercorelli, Recent advances in intelligent algorithms for fault detection and diagnosis, *Sensors* 24 (2024) 2656. doi:10.3390/s24082656.
- [18] R. Mennilli, L. Mazza, A. Mura, Integrating machine learning for predictive maintenance on resource-constrained PLCs: a feasibility study, *Sensors* 25 (2025) 537. doi:10.3390/s25020537.
- [19] D. A. Santos, et al., Hybrid hardening approach for a fault-tolerant RISC-V system-on-chip, *IEEE Transactions on Nuclear Science* 71 (2024) 1722–1730. doi:10.1109/TNS.2024.3406021.
- [20] M. Almeida, E. Pereira, G. Gonçalves, HyPredictor: hybrid failure prognosis approach combining data-driven and knowledge-based methods, in: *Proceedings of the 21st International Conference on Informatics in Control, Automation and Robotics (ICINCO 2024)*, 2024, pp. 245–252. doi:10.5220/0012924300003822.

- [21] E. Zio, Challenges to IoT-Enabled Predictive Maintenance for Industry 4.0, *IEEE Internet of Things Journal* 7 (2020) 4585–4597. doi:10.1109/JIOT.2019.2957029.
- [22] S. A. Khajeh, M. Saberikamarposhti, A. M. Rahmani, Real-time scheduling in IoT applications: a systematic review, *Sensors* 23 (2023) 232. doi:10.3390/s23010232.
- [23] O. Ettahri, A. Oukaira, M. Ali, A. Hassan, M. Nabavi, Y. Savaria, A. Lakhssassi, Real-time thermal monitoring using embedded sensors interfaces, *Sensors* 20 (2020) 5657. doi:10.3390/s20195657.
- [24] W. A. Cruz Castañeda, P. Bertemes Filho, Improvement of an edge-IoT architecture driven by artificial intelligence for smart-health chronic disease management, *Sensors* 24 (2024) 7965. doi:10.3390/s24247965.
- [25] N. Jiang, C. Zhang, Y. Cao, R. Zhan, Prognostic and health management of critical aircraft systems and components: an overview, *Sensors* 23 (2023) 8124. doi:10.3390/s23198124.
- [26] R. Mehalaine, M. Djeddar, D. Nessah, Z. Saiad, A. Saidi, Watchdog timer for fault tolerance in embedded systems, *Journal Européen des Systèmes Automatisés* 57 (2024) 1713–1720. doi:10.18280/jesa.570619.
- [27] A. Kashtalian, S. Lysenko, O. Savenko, A. Nicheporuk, T. Sochor, V. Avsiyevych, Multi-computer malware detection systems with metamorphic functionality, *Radioelectronic and Computer Systems* 2024 (1) (2024) 152–175. doi:10.32620/reks.2024.1.13.
- [28] D. Denysiuk, O. Savenko, S. Lysenko, B. Savenko, A. Kashtalian, Method for detecting steganographic changes in images using machine learning, in: 2023 13th International Conference on Dependable Systems, Services and Technologies (DESSERT), IEEE, Athens, Greece, 2023, pp. 1–6. doi:10.1109/DESSERT61349.2023.10416453.
- [29] 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* 2025 (1) (2025) 264–297. doi:10.32620/reks.2025.1.18.
- [30] S. Lysenko, K. Bobrovnikova, S. Matiukh, I. Hurman, O. Savenko, Detection of the botnets' low-rate DDoS attacks based on self-similarity, *International Journal of Electrical and Computer Engineering* 10 (2020) 3651–3659. ISSN: 2088-8708. doi:10.11591/ijece.v10i4.pp3651-3659.
- [31] O. Pomorova, O. Savenko, S. Lysenko, A. Kryshchuk, K. Bobrovnikova, A technique for the botnet detection based on DNS-traffic analysis, in: *Communications in Computer and Information Science*, vol. 522, Springer, 2015, pp. 127–138. ISSN: 1865-0929. doi:10.1007/978-3-319-19419-6_12.
- [32] S. Lysenko, O. Pomorova, O. Savenko, A. Kryshchuk, K. Bobrovnikova, DNS-based anti-evasion technique for botnets detection, in: *Proceedings of the 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, IEEE, Warsaw, Poland, 2015, pp. 453–458. doi:10.1109/IDAACS.2015.7340777.
- [33] S. Lysenko, O. Savenko, K. Bobrovnikova, A. Kryshchuk, Self-adaptive system for the corporate area network resilience in the presence of botnet cyberattacks, in: *Communications in Computer and Information Science*, vol. 860, 2018, pp. 385–401. doi:10.1007/978-3-319-92459-5_31.
- [34] F. Reghenzani, Z. Guo, W. Fornaciari, Software fault tolerance in real-time systems: identifying the future research questions, *ACM Computing Surveys* 55 (2023) 306. doi:10.1145/3589950.
- [35] F. Ratti, HeterogeneousRTOS: a CPU-FPGA real-time OS for fault tolerance on COTS at near-zero timing cost, *ACM Transactions on Embedded Computing Systems* 24 (2025) 144. doi:10.1145/3712062.
- [36] D. Simpson, J. Harkin, M. McElholm, L. McDaid, Smart watchdog for RISC-V: a novel spiking neural network approach to fault detection, *IEEE Transactions on Circuits and Systems II: Express Briefs* 72 (2025) 1–5. doi:10.1109/TCSII.2025.3583042.

- [37] B. Asch, E. Jellum, M. Lohstroh, E. A. Lee, Software-defined watchdog timers for cyber-physical systems, *IEEE Embedded Systems Letters* 18 (2024) 115–118. doi:10.1109/LES.2024.3467332 .
- [38] A. Mehalaine, A. Ferroudji, M. Ziane, K. Benahmed, Watchdog timer for fault tolerance in embedded systems, *Journal Européen des Systèmes Automatisés* 57 (2024) 1713–1720. doi:10.18280/jesa.570619.
- [39] S. Lee, S.-H. Kwon, IndWatch: industrial watchdogs based on blockchain for securing software supply chains in IIoT, *Sensors* 21 (2021) 1170. doi:10.3390/s21041170.
- [40] H. de Medeiros Neto, A. M. d. R. Araújo, J. J. P. C. Rodrigues, A. L. L. Aquino, P. H. J. Nardelli, A survey on AI-based anomaly detection in IoT and sensor networks, *Sensors* 23 (2023) 6765. doi:10.3390/s23156765.
- [41] W. G. Negera, A. Abebe, Review of botnet attack detection in SDN-enabled IoT using machine learning, *Sensors* 22 (2022) 9837. doi:10.3390/s22249837.
- [42] I. Mutambik, An efficient flow-based anomaly detection system for enhanced security in IoT networks, *Sensors* 24 (2024) 7408. doi:10.3390/s24227408.
- [43] I. Apostol, M. Preda, C. Nila, I. Bica, IoT botnet anomaly detection using unsupervised deep learning, *Electronics* 10 (2021) 1876. doi:10.3390/electronics10161876.
- [44] M. Gelgi, Y. Guan, S. Arunachala, M. S. S. Rao, N. Dragoni, Systematic literature review of IoT botnet DDoS attacks and evaluation of detection techniques, *Sensors* 24 (2024) 3571. doi:10.3390/s24113571.
- [45] O. Kozelskyi, A. Drozd, B. Savenko, P. Gaj, A model for probabilistic monitoring and proactive restart of real-time operating systems under intensive state changes in cyber-physical systems, in: *CEUR Workshop Proceedings*, vol. 4013, 2025. URL: <https://ceur-ws.org/Vol-4013/paper16.pdf>.
- [46] J. Dalou', B. Al-Duwairi, M. Al-Jarrah, Adaptive entropy-based detection and mitigation of DDoS attacks in software defined networks, *International Journal of Computing* 19 (2020) 399–410. doi:10.47839/ijc.19.3.1889.
- [47] P. Bykovyy, V. Kochan, A. Sachenko, G. Markowsky, Genetic algorithm implementation for perimeter security systems CAD, in: *4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Dortmund, 2007, pp. 634–638. doi:10.1109/IDAACS.2007.4488498.
- [48] K. Singh, K. Singh Dhindsa, B. Bhushan, Performance analysis of agent-based distributed defense mechanisms against DDoS attacks, *International Journal of Computing* 17 (2018) 15–24. doi:10.47839/ijc.17.1.945.
- [49] P. Bykovyy, Y. Pigovsky, V. Kochan, A. Sachenko, G. Markowsky, S. Aksoy, Genetic algorithm implementation for distributed security systems optimization, in: *IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSAS)*, Istanbul, 2008, pp. 120–124. doi:10.1109/CIMSAS.2008.4595845.
- [50] S. R., A. Kanavalli, A. Gupta, A. Pattanaik, S. Agarwal, Real-time DDoS detection and mitigation in software defined networks using machine learning techniques, *International Journal of Computing* 21 (2022) 353–359. doi:10.47839/ijc.21.3.2691.
- [51] S. Singaravelan, P. Velayutha Perumal, R. Arun, V. Selvakumar, D. Murugan, Deep learning-based echo state neural network for cyber threat detection in IoT-driven IICS networks, *International Journal of Computing* 23 (2024) 205–210. doi:10.47839/ijc.23.2.3538.