algorithms

MDPI

Article

Workload- and Event-Adaptive Dual-Mode Scheduling with Recovery-Aware Service Control for Smart Home Controllers

Mysha Nishat Bidhu and Jalil Boudjadar *

Department of Electrical and Computer Engineering, Aarhus University, 8000 Aarhus, Denmark; bidhu.mysha@gmail.com
* Correspondence: jalil@ece.au.dk

Abstract

Smart home gateways execute routine automation and emergency workloads on centralized, resource-constrained controllers. Under overload or dynamic workloads, conventional static scheduling policies may allow routine jobs to delay high-criticality tasks. This article presents RT-DASH, a dual-mode scheduling framework for smart home controllers. In Normal Mode, the dispatch is primarily deadline-oriented, whereas Critical Mode, activated by semantic emergency events or sustained workload conditions, gives precedence to ready high-criticality jobs and permits reduction only for application-declared optional computation subject to configured service floors. Once the configured recovery conditions are satisfied, the controller returns to Normal Mode. RT-DASH is evaluated across 15,780 simulation runs against conventional and criticality-aware scheduling baselines. Across the held-out modeled emergency workloads, the recovery-aware RT-DASH variant missed 15.68% of high-criticality deadlines, compared to 62.72% for EDF and 91.67% for period-based fixed-priority scheduling. Furthermore, RT-DASH achieved the same high-criticality deadline miss ratio and mean completed-job response time as the static criticality-first baseline, while completing a larger fraction of routine work. The RT-DASH implementation software and scripts are made public for reproducibility purposes.

Keywords: real-time scheduling; smart home controller; criticality-aware scheduling; workload adaptation; graceful degradation; recovery scheduling; FreeRTOS

1. Introduction

A smart home gateway executes heterogeneous jobs, including sensor monitoring, climate control, appliance coordination, security processing, and user command handling. These jobs share limited processing resources, usually a centralized embedded or edge solution, and differ in their arrival patterns, timing requirements, and criticality levels. Delays in routine temperature or energy updates may be tolerable, whereas delayed responses to fire, gas leak, intrusion, or access control events can have serious consequences. The scheduling problem becomes particularly important when emergency work arrives while routine jobs are already queued, requiring the controller to select work according to both deadline urgency and application criticality.

Two established real-time scheduling policies are Earliest Deadline First (EDF) and Rate-Monotonic Scheduling (RMS) [1]. Neither policy on its own captures the application consequence of allowing routine work to run before a safety response when the processor is overloaded. Mixed-criticality research addresses a related concern by changing the service available to lower-criticality tasks after a higher-criticality execution budget has

Check for updates

Academic Editor: Carolina Del-Valle-Soto

Received: 30 August 2026
Revised: 7 October 2026
Accepted: 8 October 2026
Published: 9 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

been exceeded [2–7]. A smart home controller may therefore require a scheduling response before an execution budget overrun is observed. For example, a smoke event can provide sufficient semantic evidence to trigger an immediate scheduling state transition.

Dynamically increasing the priority of an emergency job alone is insufficient to address the timeliness criticality scheduling problem. The controller must distinguish sustained emergency or overload conditions from transient workload fluctuations, prevent excessive mode switching near decision thresholds, restrict service reduction to tasks for which degradation is explicitly permitted, and avoid executing stale work after its timing relevance has expired. Emergency-first IoT scheduling illustrates the importance of explicit criticality-aware prioritization [8]. In contrast, much of the smart home scheduling literature focuses primarily on energy optimization, appliance coordination, or distributed resource management [9,10]. Although these objectives are important, they do not directly address the scheduling problem considered here: how does a resource-constrained real-time controller arbitrate between high-criticality emergency work and routine tasks while adapting its scheduling state to changing operation conditions?

To address this scheduling problem, this article proposes RT-DASH (Real-Time Dynamic Workload Allocation in Smart Homes). Under routine operating conditions, RT-DASH executes in Normal Mode, where absolute deadlines provide the primary dispatch criterion. When a semantic emergency event is detected or sustained processor utilization exceeds the configured entry condition, the controller transitions to Critical Mode. In this mode, ready high-criticality jobs receive precedence, while optional computation associated with explicitly degradable low-criticality work may be reduced within its application-defined service floor. Separate entry and recovery conditions, including utilization hysteresis and dwell times, limit unnecessary mode transitions. During recovery, once no high-criticality job remains ready, the recovery-aware policy applies deadline-oriented ordering to the remaining low- and medium-criticality backlog while retaining immediate re-entry to Critical Mode if new high-criticality work appears. Compared with scheduling models that assume fixed task sets and regular task arrival patterns, RT-DASH evaluates dynamically varying workloads representative of smart home operation and combines adaptive mode switching with a recovery strategy intended to reduce the unnecessary loss of low-criticality service.

The evaluation is organized around four research questions. RQ1 examines how precedence for ready high-criticality jobs during emergency conditions affects the observed high-criticality deadline miss ratio and completed-job response time. RQ2 evaluates whether the adaptive activation of Critical Mode provides advantages over permanent criticality-first operation and whether hysteresis reduces unnecessary mode transitions. RQ3 examines whether deadline-oriented backlog processing during recovery improves low-criticality completion while preserving the observed high-criticality deadline performance. RQ4 evaluates the robustness of these behaviors under increased workload intensity, alternative trace mappings, timing and overhead perturbations, larger task sets, and an alternative dual-criticality scheduling model based on EDF-VD.

The contributions made in this article are summarized as follows:

- We develop a task model that can be applied to various types of smart home systems—including tasks that occur periodically, respond to events, or can be explicitly degraded if needed. To evaluate the model reproducibly, we use deterministic simulation, account explicitly for assessed and right-censored deadline outcomes, and quantify service performance using completion and utility measures.
- Our system uses a dual-trigger controller in which Critical Mode is activated either by an emergency event or when the workload remains above the configured entry

condition for the required duration. Hysteresis and temporal dwell conditions govern the return to Normal Mode and reduce unnecessary mode transitions.
- We evaluate RT-DASH through 15,780 simulation runs. We compare the results against multiple baselines, analyze sensitivity and scalability, and perform stress tests for the timing and execution overhead. We also use a separate EDF-VD benchmark to examine the service–control trade-offs under an alternative dual-criticality scheduling model.
- We deliver a portable C++ implementation and build-validated FreeRTOS-oriented prototypes for the QEMU LM3S6965EVB and STM32F103C8T6 targets.

Taken together, these contributions provide an application-specific scheduling framework together with a reproducible evaluation of its behavior under the modeled operating conditions. The rest of the article is structured as follows: Section 2 cites relevant related work; Section 3 presents the foundations and functionality of RT-DASH; and the software implementation and analysis metrics are described in Section 4, whereas the experimental results and validation are given in Section 5. Finally, Section 6 concludes the article.

2. Related Work

Real-time scheduling has been extensively studied across a wide range of application domains [2–6,9–13]. Among established uniprocessor scheduling policies, EDF assigns priority according to job deadlines, whereas RMS assigns fixed priorities according to task periods [1]. Vestal's mixed-criticality model introduced assurance-dependent execution budgets and linked them to system modes [2]. Subsequent research examined the schedulability of high-criticality tasks when execution budget overruns trigger a transition between system modes [3].

Subsequent work has examined whether low-criticality service must be suspended completely following a mode transition. Flexible mixed-criticality scheduling instead permits low-criticality service to be reduced or adapted rather than fully discarded [6]. Degraded-quality EDF-VD follows the same general service retention principle under an overrun-driven dual-criticality model [5]. Boudjadar et al. extended this concept further, combining changes at both the task level and the overall system level and allowing lower-criticality task periods to be stretched when the system is under greater strain [7]. Degraded-quality EDF-VD and adaptive low-criticality service control provide the closest comparisons to the service adaptation component of RT-DASH because they address the common trade-off between protecting high-criticality execution and retaining useful low-criticality service. Their mode transition semantics differ; however, the mixed-criticality approaches considered above are principally driven by execution budget overruns, whereas the primary RT-DASH smart home experiments use semantic emergency events and sampled workload conditions as mode entry signals. We therefore do not treat these approaches as directly interchangeable in the main smart home workload matrix. Instead, Section 5.4.3 evaluates degraded-quality EDF-VD and RT-DASH-style service control in a separate overrun-driven dual-criticality benchmark, allowing their service retention trade-offs to be examined under common mode transition semantics.

In the classical graceful degradation paradigm, the computation associated with a job can be separated into a mandatory part required to obtain an acceptable result and an optional part that improves result quality; under timing or resource pressure, optional work can therefore be reduced while the mandatory portion is retained. Esmaili et al. [14] studied task graph scheduling with imprecise computation under end-to-end deadline and energy constraints, while Stavrinides and Karatza examined imprecise computation for real-time task graphs in heterogeneous distributed systems [15] and later combined approximate computation with DVFS for real-time workflow scheduling under energy, cost, and QoS objectives [16]. Jiang et al. [17] further connected approximated computation

with mixed-criticality scheduling through the HIART-MCS architecture. These approaches share with RT-DASH the general principle that useful service can be retained by reducing selected computations under constrained conditions rather than indiscriminately discarding all lower-priority work. The mechanisms are nevertheless different: RT-DASH uses application-declared per-task service floors together with semantic event and sampled load mode control on a single smart home controller, and it does not adopt the task graph approximation, precision error, or formal mixed-criticality models.

For IoT applications, several scheduling approaches explicitly prioritize critical work during emergency or overload conditions [8]. In contrast, many smart home scheduling studies focus primarily on energy optimization, appliance coordination, prediction, or distributed resource management [9,10]. These approaches address important system-level objectives, but they generally do not focus on job-level arbitration between high-criticality emergency work and routine tasks sharing a resource-constrained embedded controller.

Huang et al. [18] and Ren et al. [19] associated execution dependencies between high- and low-criticality tasks, such that only low-criticality tasks linked to an activated high-criticality task are discarded. Gu et al. [20] introduced a hierarchical component-based scheduling technique in which multiple high-criticality tasks can be grouped into a single component, and each task can run different modes. If any critical task within a component switches to high mode, all low-criticality tasks belonging to the component are discarded. Erickson et al. [21] proposed a scheduling framework for mixed-criticality systems to recover from transient overload conditions. Their recovery mechanism scales task inter-release times to reduce the frequency of job releases.

RT-DASH builds on established concepts from adaptive scheduling, emergency-aware prioritization, and mixed-criticality service control but integrates them into an application-specific dual-mode framework for smart home controllers. Its distinguished integration combines semantic event and workload-based entry into Critical Mode, application-declared minimum service levels for degradable tasks, the explicit handling of stale or delayed work, and a recovery-aware backlog policy. Table 1 summarizes how these mechanisms relate to representative scheduling families considered in the literature.

Table 1. Position of RT-DASH relative to representative scheduling families.

Approach	Primary Setting	Adaptation	Emergency Treatment	Position Relative to RT-DASH
EDF/RMS [1]	Uniprocessor real-time tasks	Deadline- or period-based priority	No explicit semantic emergency mode	Reference baselines; no application service recovery
Vestal/response time models [2–4]	Assurance-based mixed criticality	Budget overrun mode change and response time analysis	Indirect through overrun	Formal assurance models; different trigger semantics
EDF-VD degraded quality [5]	Dual-criticality implicit deadline tasks	Virtual deadlines and reduced LO service	Overrun-driven	Direct service degradation comparator; evaluated separately under common overrun-driven semantics
Adaptive workload control [6,7]	Mixed-criticality embedded systems	Service reduction or period stretching	Overrun-driven	RT-DASH adds semantic event entry, sampled load, and recovery backlog policy
Imprecise/approx. computation [14–17]	Real-time and MC systems	Optional work reduction	Constraint-driven	Related degradation concept; different service and trigger semantics
Emergency-first IoT [8]	Embedded hard/soft real-time work	Emergency priority elevation	Explicit	Less emphasis on hysteresis, service floors, and recovery
Learning/energy scheduling [9,10]	Smart home energy and appliances	Optimization or reinforcement learning	Application-dependent	Different objective, resources, and runtime overhead
RT-DASH	Single smart home controller	Dual-trigger mode control and recovery-aware service	Explicit high-criticality event	Application-specific integration evaluated with matched controls

Note: EDF, Earliest Deadline First; RMS, Rate-Monotonic Scheduling; EDF-VD, Earliest Deadline First with Virtual Deadlines; MC, mixed criticality; RT-DASH, the proposed dual-mode smart home scheduling controller.

3. Foundations and Functionality Specification

This section describes the RT-DASH workload specifications and its mode-dependent scheduling behavior.

3.1. Task, Job, and Service Model

A task i is the basic schedulable unit and is represented by the tuple

$$\tau_i = (C_i, D_i, T_i, \ell_i, g_i, \gamma_{i,\min}) \tag{1}$$

where C_i denotes the nominal execution demand, D_i is the relative deadline, and T_i is the period for periodic tasks; one-shot event tasks have no associated T_i. The label $\ell_i \in \{1, 2, 3\}$ denotes low, medium, or high application criticality. The flag g_i indicates whether optional computation associated with a task may be reduced under the service control policy, while $\gamma_{i,\min} \in [0, 1]$ specifies the minimum fraction of nominal computation that the application permits the controller to retain. Task names, resource annotations, and history labels are implementation metadata and do not affect dispatch ordering. The k-th release of task i creates job $J_{i,k}$ at release time $r_{i,k}$, with absolute deadline $d_{i,k} = r_{i,k} + D_i$.

$$d_{i,k} = r_{i,k} + D_i \tag{2}$$

The simulator applies strict deadline handling: a job that is not completed by its absolute deadline is classified as missed and is removed rather than being retained in the backlog. If a job's absolute deadline occurs after the simulation horizon, the job is treated as right-censored and is excluded from both the completed and missed counts. Only tasks marked as low-criticality can be degraded; high-criticality tasks always have $\gamma_{i,\min} = 1.0$, so their modeled execution demand is not reduced by the degradation policy. When the optional work is reduced, it only lessens the simulated workload by the parts that are actually skipped. Table 2 gives an overview of the specification symbols.

Table 2. Principal notation.

Symbol	Definition
τ_i, $J_{i,k}$	Task i and its k-th released job
C_i	Nominal execution budget
D_i, $d_{i,k}$	Relative and absolute deadline
T_i	Period; absent for one-shot event jobs
ℓ_i	Application criticality: low, medium, high
$\gamma_{i,\min}$	Minimum approved service fraction
$\widehat{U}(t)$, $\overline{U}(t)$	Windowed and smoothed processor utilization
U_{high}, U_{low}	Critical entry and normal return thresholds
H_{entry}, H_{recovery}	Entry and recovery dwell times
$R_{i,k}$	Completion time minus release time
$M_{i,k}$	Integer mandatory service floor for job $J_{i,k}$
$x_{i,k}(t)$, $e_{i,k}(t)$	Executed and remaining work at time t
$O_{i,k}(t)$	Currently removable optional work

3.2. Runtime Monitoring and Mode Control

Let B(t-W,t) denote the processor time used during the latest observation window W. Window utilization is estimated as

$$\widehat{U}(t) = \frac{B(t - W, t)}{W} \tag{3}$$

To prevent one sample from triggering a transition, the controller averages the most recent h observations:

$$\overline{U}(t) = \frac{1}{h} \sum_{j=0}^{h-1} \widehat{U}(t - j\Delta) \tag{4}$$

Samples are taken every Δ time duration. The Boolean signal *H(t)* becomes true when a high-criticality event is released or high-criticality work remains ready. In either case, the controller switches to Critical Mode immediately. The system only switches to Critical Mode based on the workload if the average utilization $\overline{U}(t)$ stays at or above U_{high} for the entire period H_{entry}. The return to Normal Mode is intentionally more conservative: the high-criticality ready set must be empty, and the average utilization $U(t)$ must remain at or below U_{low} throughout the recovery interval H_{recovery}. Because $U_{\text{low}} < U_{\text{high}}$, the controller incorporates both amplitude hysteresis and a temporal dwell requirement before completing the transition. The arrival of new high-criticality work cancels the recovery process and causes immediate re-entry to Critical Mode. Figure 1 illustrates how these mode transitions work.

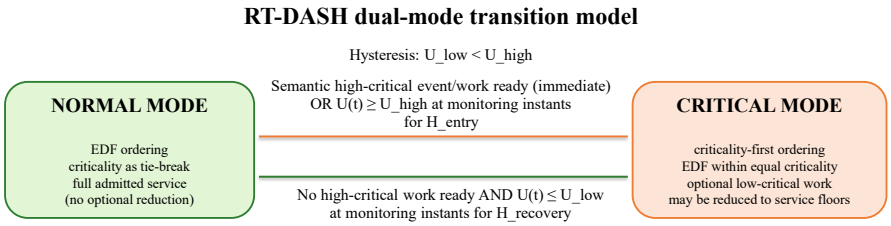


Figure 1. RT-DASH dual-mode transition model. Semantic high-criticality events cause immediate entry; load-based entry and normal return require configured dwell conditions.

3.3. Mode-Dependent Dispatch and Recovery-Aware Refinement

When the system runs in Normal Mode, it decides which job to execute using a set of rules (a lexicographic key):

$$K^N(J_{i,k}) = (d_{i,k}, -\ell_i, r_{i,k}, \text{id}_i) \tag{5}$$

Here, the absolute deadline $d_{i,k}$ is the primary ordering criterion. If two jobs have the same absolute deadline, the job with the higher criticality level ℓ_i is selected first, followed by the earlier release time $r_{i,k}$ and then the job identifier id_i as deterministic tie-breakers. Therefore, Normal Mode implements EDF with deterministic criticality-aware tie-breaking. Task periods are not used in this dispatch rule; period-based fixed-priority scheduling using RMS is evaluated only as a separate baseline. In Critical Mode, the system switches to a different ordering rule:

$$K^C(J_{i,k}) = (-\ell_i, d_{i,k}, r_{i,k}, \text{id}_i) \tag{6}$$

The recovery-aware policy selects the dispatch rule according to the ready job set and controller state. While at least one high-criticality job is ready, Critical Mode ordering is applied. If no high-criticality job is ready but the controller remains within the recovery interval, the remaining low- and medium-criticality jobs are ordered according to the Normal Mode policy. The arrival of new high-criticality work immediately restores Critical Mode ordering. This refinement affects only backlog processing and does not reduce the precedence assigned to high-criticality jobs.

For a degradable job $J_{i,k}$, $C_{i,k}$ is the original integer execution budget, $x_{i,k}(t)$ is the work already executed, and $e_{i,k}(t)$ is the work still pending. The mandatory integer service $M_{i,k}$ and the optional work that remains removable at time t, $O_{i,k}(t)$, are defined as

$$\begin{aligned} M_{i,k} &= \text{round}(\gamma_{i,\min} C_{i,k}), \\ O_{i,k}(t) &= \max\{0,\ e_{i,k}(t) - \max[0,\ M_{i,k} - x_{i,k}(t)]\}. \end{aligned} \tag{7}$$

Equation (7) follows the reference implementation's round-to-nearest, ties-to-even rule. When the controller is in Critical Mode and a scheduling decision is required, it estimates the near-term excess demand over the observation horizon W as $E(t) = \max\{0, \sum e_{i,k}(t) - U_{\text{high}} W\}$, considering only ready jobs with deadlines no later than $t + W$. If $E(t) > 0$, optional computation is reduced only to the extent required by this estimate and never by more than the currently removable amount $O_{i,k}(t)$ for any job. For degradable low-criticality jobs, candidates are ordered in terms of increasing minimum service fraction $\gamma_{i,\min}$, with ties resolved by later absolute deadlines and then by job identifier. This ordering prioritizes tasks with greater declared service flexibility while preserving already executed work and the mandatory service floor. The complete removal of a job is permitted only when its application-declared minimum service fraction is $\gamma_{i,\min} = 0$. Equation (7) does not represent frequency reduction, which changes T_i and must be encoded as another task configuration. Table 3 summarizes the complete rule set.

Table 3. Mode-dependent dispatch and service behavior.

Controller State	Dispatch	Service Rule	Purpose
Normal mode	EDF	Full admitted service	Routine urgency ordering
Critical mode, high ready	Criticality-first EDF	High work is not degraded; optional low-criticality work may be reduced only under estimated excess demand	Prioritizes modeled high-criticality work
Critical mode, no high ready (default)	Criticality-first EDF	Controlled reduced-service state retained	Can prolong low/medium backlog
Critical mode, no high ready (recovery-aware)	EDF over low/medium backlog	Controlled reduced-service state retained until the exit guard completes	Improves backlog clearance while preserving immediate high precedence
Return to normal	EDF	Full service restored for new work; stale jobs remain dropped	Avoids replaying expired automation

Algorithm 1 consolidates RT-DASH's event-triggered and load-triggered mode transitions, service floor-preserving degradation, and recovery-aware dispatch rules. The utilization guards are evaluated only at the monitoring instants separated by Δ. In contrast, the release of a high-criticality event causes immediate entry into Critical Mode.

3.4. Policy Properties and Complexity

The following propositions characterize deterministic properties of the implemented RT-DASH policy rather than formal schedulability guarantees. Specifically, they establish the precedence of ready high-criticality jobs in Critical Mode, the preservation of the configured computational service floor during degradation, and the sampled condition required for workload-triggered mode entry. They do not prove that a given task set is schedulable, derive worst-case response time bounds, or guarantee that all deadlines will be met. Accordingly, these properties should be interpreted as behavioral invariants of the scheduling and service control rules, while deadline feasibility and timing performance are evaluated empirically and through the diagnostic analyses reported later in the article.

Algorithm 1: RT-DASH recovery-aware scheduling decision

Input: Time t, ready set $\mathcal{R}(t)$, mode $m \in \{\mathrm{N}, \mathrm{C}\}$, high-criticality signal $H(t)$, and guard onset times s_{E} and s_{R}

Output: Selected job $J^{\star}$, updated mode, guards, and service levels

Insert all jobs released at t into $\mathcal{R}(t)$;
if $m = \mathrm{N}$ *and* $H(t) = 1$ **then**
  $m \leftarrow \mathrm{C}$; $s_{\mathrm{E}}, s_{\mathrm{R}} \leftarrow \perp$;
if t *is a monitoring instant* **then**
  Update $\widehat{U}(t)$ and $\overline{U}(t)$ using Equations (3) and (4);
  if $m = \mathrm{N}$ *and* $H(t) = 0$ **then**
    if $\overline{U}(t) \geq U_{\mathrm{high}}$ **then**
      if $s_{\mathrm{E}} = \perp$ **then**
        $s_{\mathrm{E}} \leftarrow t$;
      if $t - s_{\mathrm{E}} \geq H_{\mathrm{entry}}$ **then**
        $m \leftarrow \mathrm{C}$; $s_{\mathrm{E}}, s_{\mathrm{R}} \leftarrow \perp$;
    else
      $s_{\mathrm{E}} \leftarrow \perp$;
  if $m = \mathrm{C}$ **then**
    if $H(t) = 0$ *and* $\overline{U}(t) \leq U_{\mathrm{low}}$ **then**
      if $s_{\mathrm{R}} = \perp$ **then**
        $s_{\mathrm{R}} \leftarrow t$;
      if $t - s_{\mathrm{R}} \geq H_{\mathrm{recovery}}$ **then**
        $m \leftarrow \mathrm{N}$; $s_{\mathrm{E}}, s_{\mathrm{R}} \leftarrow \perp$;
    else
      $s_{\mathrm{R}} \leftarrow \perp$;
if $m = \mathrm{C}$ *and controlled degradation is enabled* **then**
  $\mathcal{D}(t) \leftarrow \{J_{i,k} \in \mathcal{R}(t) : d_{i,k} \leq t + W\}$;
  $E \leftarrow \max\{0, \sum_{J_{i,k} \in \mathcal{D}(t)} e_{i,k}(t) - U_{\mathrm{high}} W\}$;
  Form the set $\mathcal{G}(t)$ of degradable, low-criticality ready jobs with $O_{i,k}(t) > 0$;
  Sort $\mathcal{G}(t)$ by $(\gamma_{i,\min}, -d_{i,k}, \mathrm{id}_i)$;
  foreach $J_{i,k} \in \mathcal{G}(t)$ *while* $E > 0$ **do**
    $\delta \leftarrow \min\{O_{i,k}(t), \lceil E \rceil\}$;
    Remove δ units of optional work from $J_{i,k}$;
    $E \leftarrow \max\{0, E - \delta\}$;
if $\mathcal{R}(t) = \varnothing$ **then**
  return idle;
if $m = \mathrm{C}$ *and a high-criticality job is pending* **then**
  $J^{\star} \leftarrow \arg\min_{J \in \mathcal{R}(t)} K^{\mathrm{C}}(J)$;
else
  $J^{\star} \leftarrow \arg\min_{J \in \mathcal{R}(t)} K^{\mathrm{N}}(J)$;
return $J^{\star}$, updated mode, guards, and service levels

Proposition 1 (Criticality dominance). *If RT-DASH is in Critical Mode and at least one high-criticality job is ready, the scheduler selects a high-criticality job before every ready job of lower criticality.*

Proof. Let J_{H} be any ready high-criticality job and let J_{L} be any ready low- or medium-criticality job. Their Critical Mode ordering keys are given by

$$K^{\mathrm{C}}(J_{i,k}) = (-\ell_i, d_{i,k}, r_{i,k}, \mathrm{id}_i).$$

Given that $\ell_{\mathrm{H}} > \ell_{\mathrm{L}}$, their first key components satisfy $-\ell_{\mathrm{H}} < -\ell_{\mathrm{L}}$. Lexicographic minimization therefore ranks J_{H} ahead of J_{L}, independently of their deadlines, release times, or identifiers. Consequently, no lower-criticality job can be selected while a high-criticality job is ready. This ordering property alone does not guarantee that every high-criticality deadline is feasible or will be satisfied. □

Proposition 2 (Integer service floor preservation)**.** *For every degradable job $J_{i,k}$, every sequence of optional work reductions permitted by RT-DASH preserves the mandatory integer service floor*

$$M_{i,k} = \mathrm{round}(\gamma_{i,\min} C_{i,k}),$$

where the implementation uses round-to-nearest with ties to even. More precisely, if $x_{i,k}(t)$ is the executed work and $e_{i,k}(t)$ is the remaining work, then $x_{i,k}(t) + e_{i,k}(t) \geq M_{i,k}$ after every permitted execution or degradation step.

Proof. Initially, $x_{i,k}(0) = 0$ and $e_{i,k}(0) = C_{i,k}$. Since $0 \leq \gamma_{i,\min} \leq 1$ and $C_{i,k}$ is a positive integer, $0 \leq M_{i,k} \leq C_{i,k}$. Therefore, $x_{i,k}(0) + e_{i,k}(0) = C_{i,k} \geq M_{i,k}$.

Assuming that the inequality holds immediately before a degradation step at time t, the removable optional work is $O_{i,k}(t) = \max\{0,\, e_{i,k}(t) - \max[0, M_{i,k} - x_{i,k}(t)]\}$. Let $\delta_{i,k}(t)$ denote the work removed during this step. Algorithm 1 requires that

$$0 \leq \delta_{i,k}(t) \leq O_{i,k}(t).$$

The induction hypothesis implies $e_{i,k}(t) \geq \max[0, M_{i,k} - x_{i,k}(t)]$. Consequently, after the reduction,

$$\begin{aligned} e_{i,k}^{\mathrm{new}}(t) &= e_{i,k}(t) - \delta_{i,k}(t) \\ &\geq \max[0, M_{i,k} - x_{i,k}(t)]. \end{aligned}$$

It follows that

$$x_{i,k}(t) + e_{i,k}^{\mathrm{new}}(t) \geq M_{i,k}.$$

An ordinary execution step transfers an equal amount of work from $e_{i,k}$ to $x_{i,k}$, so it does not change their sum. By induction over all execution and degradation steps, the retained computational service never falls below $M_{i,k}$. □

Proposition 2 establishes the preservation of the configured computational service floor. The application designer remains responsible for selecting $\gamma_{i,\min}$ so that the corresponding retained computation provides acceptable application-level service.

Proposition 3 (Sampled load entry guard)**.** *Suppose that RT-DASH has returned to Normal Mode and that a subsequent load-triggered transition to Critical Mode occurs at monitoring instant t_q. Then, there exists an earlier monitoring instant $t_p \leq t_q$ such that $t_q - t_p \geq H_{\mathrm{entry}}$ and $\overline{U}(t_j) \geq U_{\mathrm{high}}$ for every monitoring instant $t_j \in [t_p, t_q]$. A high-criticality event can nevertheless cause immediate entry into Critical Mode without satisfying this load-based condition.*

Proof. After entry into Normal Mode, the load entry guard onset s_{E} is reset to $\perp$. At a monitoring instant t_j, the guard can be started or retained only when $\overline{U}(t_j) \geq U_{\mathrm{high}}$. The first qualifying sample sets $s_{\mathrm{E}} = t_p$. If a later sampled value satisfies $\overline{U}(t_j) < U_{\mathrm{high}}$,

Algorithm 1 resets s_E to $\perp$. Therefore, a load-triggered transition at t_q is possible only if every sampled value from t_p through t_q satisfies the high-utilization condition and

$$t_q - s_E = t_q - t_p \geq H_{entry}.$$

The high-criticality signal $H(t)$ is evaluated independently of this load guard. Hence, when $H(t) = 1$, the controller may enter Critical Mode immediately, even if the load entry guard is inactive. □

Proposition 3 describes the sampled controller implemented in the simulator; it does not claim that utilization is observed continuously between monitoring instants. It also does not provide a global stability guarantee or bound transitions caused by repeated high-criticality events.

Taken together, Propositions 1–3 establish only these policy-level invariants of the implemented controller. They are neither necessary nor sufficient schedulability conditions and do not imply worst-case response time or deadline feasibility guarantees for arbitrary task sets.

Let $n = |\mathcal{R}(t)|$, let b be the number of busy history entries retained for the utilization window, and let h be the number of utilization samples used for smoothing. Selecting a job requires an $O(n)$ scan. Constructing the degradation candidates requires $O(n)$ time, sorting them requires $O(n \log n)$ time, and applying the reductions requires at most $O(n)$ time. A monitoring update requires $O(b + h)$ time in the reference implementation. Thus, the worst-case cost of a scheduling instant that includes monitoring and degradation is

$$O(n \log n + b + h),$$

whereas an ordinary dispatch without degradation requires $O(n)$ time. The corresponding working memory requirement is $O(n + b + h)$. When W, Δ, and h are fixed configuration parameters, the job-dependent worst-case time reduces to $O(n \log n)$. These bounds characterize the reference software implementation and do not constitute physical hardware timing measurements. Figure 2 summarizes the RT-DASH architecture and separates the workload generation, runtime monitoring, mode control, dispatch, service adaptation, simulation instrumentation, and implementation validation paths.

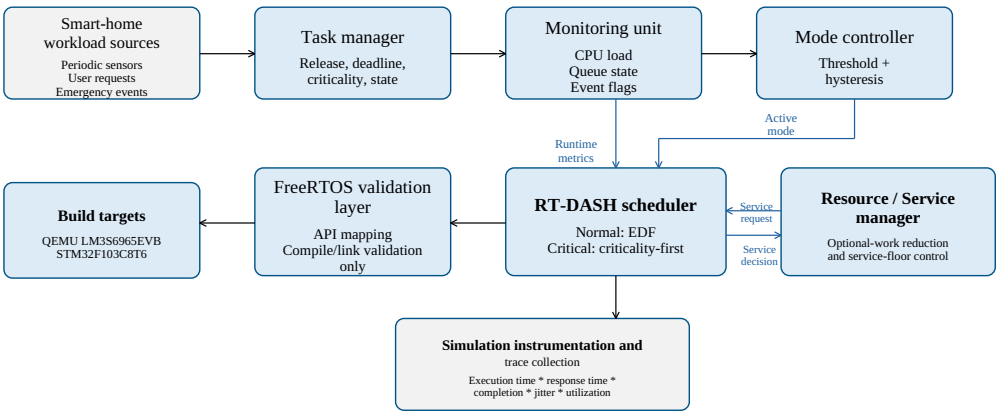


Figure 2. RT-DASH architecture and workflow. Workload generation, monitoring, mode control, dispatch, service adaptation, metrics, and implementation targets are separated to preserve traceability.

4. Implementation and Experiment Configuration

This section presents the implementation, dataset, experiment configuration, and metrics used for the RT-DASH performance analysis.

4.1. FreeRTOS-Oriented Implementation of RT-DASH

Every numerical claim in this article comes from the deterministic software simulation of a single preemptive processor. Python 3.13.5 simulator is the primary experimental engine. A separately compiled C++17 core checks whether the policy semantics carry over to another implementation language, but its executions are not pooled with the Python runs for statistical analysis. FreeRTOS projects were also compiled and linked for QEMU LM3S6965EVB and STM32F103C8T6 targets. Since neither target was executed on physical hardware, the study reports no measured WCETs, energy traces, or hardware transition time distributions. Accordingly, the FreeRTOS artifacts are used only to demonstrate and build the implementation portability; they do not provide the measured scheduler overhead, the context switch cost, runtime memory or stack consumption, platform-specific WCET values, mode transition latency, or hardware-level deadline feasibility evidence.

For a fixed configuration and seed, the simulator is deterministic and advances in 10 ms steps, reallocating unused time within a step when a job is completed early. Its logs record releases, starts, completions, deadline outcomes, preemptions, delivered service, mode changes, queue states, and recovery episodes. A separate C++17 core implements the same policy selection, mode control, and resource-accounting semantics; its five policy executions assess build integrity and semantic portability rather than providing additional statistical replications.

FreeRTOS Kernel V11.1.0 prototypes model jobs as worker tasks coordinated by a release and scheduling manager [22]. The QEMU LM3S6965EVB and STM32F103C8T6 targets include the required Cortex-M3 startup, linker, and SVC/PendSV/SysTick support and were compiled and linked successfully; as stated above, this constitutes build validation only. Reproducibility checks included all 25 passing Python 3.13.5 tests, the C++ Release build and CTest target, and exact regeneration of the 2520 configuration training and 1680 optimized matrix runs. Supplementary File S1 provides the complete versioned software environment, source code, configurations, run-level outputs, statistical analyses, inference scripts, figures, manifests, licensing information, and SHA-256 integrity checks.

4.2. Workload Construction and OpenSHS Trace Mapping

The archived Smart Home Dataset generated using OpenSHS [23] contains 1,048,575 rows and 31 fields: 29 binary sensor or device states, one activity label, and a timestamp. Inspection found three duplicate rows and no missing values. Execution budgets, deadlines, criticality labels, resource annotations, and fire/gas/intrusion semantics were assigned as part of the experimental model. These quantities are therefore treated as controlled modeling parameters rather than measurements obtained from OpenSHS or from a deployed embedded platform.

Six periodic templates form the base controlled workload. In milliseconds, their (execution budget, relative deadline, period) values are temperature (12, 100, 100), motion (8, 80, 80), energy (18, 250, 250), lighting (15, 60, 120), display (20, 50, 100), and comfort (30, 150, 200). Temperature, motion, and energy are of medium criticality. Lighting, display, and comfort are of low criticality and degradable, with service floors of 0.50, 0.25, and 0.40. The selected template values are intended to span heterogeneous real-time behaviors rather than reproduce measured timing from a specific device platform. Criticality assignments similarly encode the experimental distinction between functions that should retain full service and functions for which controlled degradation is permitted. These assignments are therefore used to exercise the scheduler across contrasting timing and service conditions, not to claim that the listed values constitute canonical smart home timing parameters. The service floor values likewise represent heterogeneous degrees of permitted degradation, chosen for a controlled comparative evaluation; they are not

measurements of user-perceived quality or validated minimum service requirements for specific commercial devices. For a requested nominal utilization, all budgets are scaled together, perturbed by a seeded factor drawn uniformly from 0.92 to 1.08, and corrected after integer rounding so that the target load is restored.

The controlled matrix contains 28 scenario/load combinations: routine {0.50, 0.65, 0.80, 0.90}; overload {0.90, 1.00, 1.10, 1.20}; mild emergency {0.60, 0.75, 0.90}; moderate emergency {0.70, 0.85, 1.00}; severe emergency {0.80, 0.95, 1.10}; prolonged emergency {0.80, 0.95}; threshold chatter {0.72, 0.82, 0.88}; recovery backlog {0.75, 0.90, 1.05}; and optional service stress {0.70, 0.85, 1.00}. Emergency bursts occur at 15 and 40 s. Mild, moderate, and severe bursts contain 1–2, 2–3, and 3–4 high-criticality jobs, respectively. Their execution budgets span 45–70, 60–90, and 80–115 ms; deadlines span 260–380, 220–320, and 170–240 ms; and within-burst spacing is 20, 20, and 10 ms. Every policy receives the same releases and budgets in a matched scenario/load/seed cell. The mild, moderate, and severe emergency profiles are constructed to impose progressively stronger timing pressure: the burst size and execution demand increase while the deadline ranges and within-burst spacing become tighter. The emergency releases at 15 s and 40 s are controlled injection points used to evaluate the scheduler behavior before and during repeated high-criticality bursts within a fixed 60 s run. Likewise, the fire/gas/intrusion interpretation denotes modeled high-criticality event semantics rather than measured execution times or certified response requirements for these applications.

Trace robustness was examined with five contiguous OpenSHS windows. Each window was converted to tasks under relaxed (budget $\times$0.75, deadline $\times$1.25), nominal ($\times$1.00, $\times$1.00), and stressed (budget $\times$1.25, deadline $\times$0.85) profiles. Activity rows release event jobs, while changes in sensor state release additional medium-criticality jobs. The mapping can be reproduced from the supplementary code, but it remains a constructed bridge from trace activity to real-time tasks rather than a measurement of embedded timing. Figure 3 makes this distinction explicit.

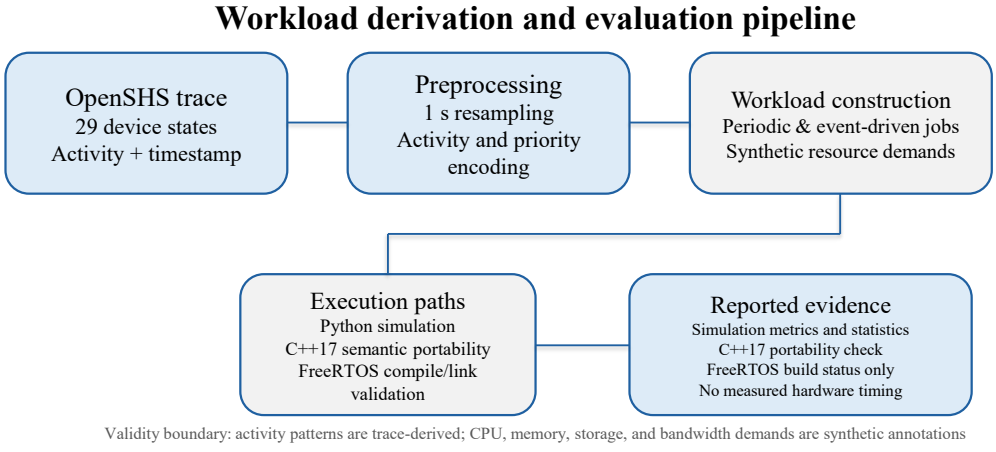


Figure 3. Evaluation provenance. Observed OpenSHS activity/device patterns are separated from modeled execution budgets, deadlines, criticality labels, resource annotations, and injected emergency scenarios.

4.3. Experimental Matrix, Configuration Selection, and Baselines

Each principal run covers 60,000 ms. Table 4 provides the complete experimental inventory, comprising 15,780 simulation executions across the controlled, sensitivity, scalability, trace mapping, configuration selection, held-out, robustness, EDF-VD, and optimized matrix studies. The 70 demand screening records and five C++ policy runs are not included in this simulation.

Table 4. Complete experimental inventory.

Study Component	Executions	Design
Controlled scenario matrix	7560	28 cells × 30 seeds × 9 policies
Parameter sensitivity	880	One factor at a time around the nominal controller
Scalability	450	5 task counts × 3 loads × 3 policies × 10 seeds
Trace mapping robustness	90	5 windows × 3 mappings × 6 policies
Configuration training	2520	36 configurations × 7 cells × 10 seeds
Configuration validation	420	6 shortlisted configurations × 7 cells × 10 seeds
Locked held-out test	420	7 cells × 10 seeds × 6 variants
Overhead and jitter	960	2 scenarios × 4 overheads × 3 jitters × 4 variants × 10 seeds
External EDF-VD benchmark	800	16 parameter cells × 5 policies × 10 seeds
Optimized full matrix	1680	28 cells × 30 seeds × 2 optimized variants
Total simulations	15,780	Demand screening and five C++ runs excluded

Configuration selection was completed before the held-out partition was opened. Seeds 0–9 were used for training, seeds 10–19 for validation, and seeds 20–29 for the locked test. The search covered 36 combinations of $U_{\text{high}} \in \{0.80, 0.85, 0.90\}$, hysteresis gap $\in \{0.10, 0.20\}$, entry hold $\in \{0, 200\}$ ms, and recovery hold $\in \{500, 1000, 2000\}$ ms. The six leading training candidates were then evaluated on validation data using the following predeclared safety-dominant score:

$$\begin{aligned} S = {} & 0.50\,\mathrm{DMR}^H + 0.15\,\min\left\{2, \frac{\overline{R}^H}{500\,\mathrm{ms}}\right\} + 0.15\left(1 - U^L\right) \\ & + 0.10\left(1 - \mathrm{CR}^L\right) + 0.05\,\min\left\{2, \frac{M}{50}\right\} + 0.05\,\min\left\{2, \frac{W_{95}^L}{2000\,\mathrm{ms}}\right\}. \end{aligned} \tag{8}$$

In Equation (8), DMR_H and R^H denote the mean high-criticality deadline miss ratio and completed-job response time across the emergency scenarios, while U^L and CR_L denote the mean low-criticality linear work-weighted utility and completion rate across the non-routine scenarios. M is the mean mode transition count in the threshold chatter scenario, and W_{95}^L is the mean low-criticality 95th-percentile waiting time in the recovery backlog scenario. The weights encode a pre-declared safety-dominant engineering preference rather than statistically fitted or theoretically optimized coefficients: the high-criticality DMR receives 0.50 as the primary safety-oriented criterion; the high-criticality response time and retained low-criticality utility receive 0.15 each; low-criticality completion receives 0.10; and the transition count and backlog waiting time receive 0.05 each as secondary stability and recovery criteria. The coefficients sum to one and are fixed before held-out evaluation. The response time, transition count, and waiting time are normalized by fixed reference scales of 500 ms, 50 transitions per 60 s run, and 2000 ms, respectively, with each normalized term capped at 2. The 500 ms scale provides a sub-second reference above the modeled emergency deadline range, 50 transitions distinguishes persistent chatter from occasional switching, and 2000 ms corresponds to the longest recovery-hold timescale considered during the configuration search. These heuristic normalizers and caps make the heterogeneous criteria comparable and limit domination by extreme values; they are neither deadline feasibility guarantees nor application-level safety limits, and different application priorities or weights could select a different configuration.

The complete training and validation ranking of the shortlisted RT-DASH configurations is provided in Supplementary File S1; Table 5 reports the configuration retained for the held-out and optimized matrix evaluations.

Table 5. Controller defaults and locked configuration used for held-out and optimized matrix evaluations.

Parameter	Controlled Default	Locked Selected
Upper utilization threshold	0.85	0.90
Lower utilization threshold	0.65	0.80
Entry hold	200 ms	200 ms
Recovery hold	1000 ms	500 ms
Observation window	1000 ms	1000 ms
Monitoring interval	100 ms	100 ms
Smoothing samples	5	5
Simulation quantum	10 ms	10 ms
Run duration	60,000 ms	60,000 ms

RT-DASH is compared with three established scheduling baselines. The first is Earliest Deadline First (EDF), with deterministic criticality-aware tie-breaking for jobs with equal absolute deadlines. The second is period-based fixed-priority scheduling: RMS is applied to periodic tasks, while one-shot event tasks use their relative deadlines as surrogate periods for priority assignment. The third is static criticality-first EDF, which orders jobs by criticality before deadline urgency.

The conventional EDF and period-based fixed-priority baselines are evaluated without using the semantic emergency signal $H(t)$; their dispatch rules therefore remain unchanged when a modeled high-criticality emergency is released. Equivalent event information could, in principle, be supplied to either baseline and used to activate a different, criticality-aware dispatch rule. Such a modification, however, would introduce an event-triggered adaptation mechanism and would no longer represent conventional EDF or the fixed-priority baseline evaluated here. Accordingly, the direct EDF/RMS comparisons measure the performance of the complete RT-DASH control strategy and should not be interpreted as isolating the benefit of semantic emergency information alone. This contribution is examined separately through the "no semantic event entry" ablation, which retains the remaining RT-DASH mechanisms while removing immediate event-based activation. The static criticality-first EDF baseline provides an additional control by applying criticality-first dispatch continuously without requiring semantic event activation. Together, these comparisons distinguish the contribution of emergency information from that of criticality-first dispatch and the remaining adaptive mechanisms.

We additionally perform ablation experiments in which individual RT-DASH mechanisms, including hysteresis, semantic event entry, service degradation, and the recovery hold, are removed to quantify their respective effects. For the held-out evaluation, we compare the standard RT-DASH configuration, the configuration selected during validation, and the recovery-aware extension. To complement the primary EDF, RMS, and static criticality-first baselines with a more directly related adaptive mixed-criticality and service degradation comparison, we separately evaluate EDF-VD in both its classical drop and degraded-quality forms [5]. EDF-VD assumes dual-criticality task sets, virtual deadlines, and execution budget overrun-driven mode changes, which differ from the semantic event and sampled workload triggers used in the primary smart home experiments. We therefore use a dedicated budget overrun benchmark in which EDF-VD and RT-DASH-style service control operate under common overrun-driven mode transition semantics. This enables a comparison of high-criticality deadline outcomes and retained low-criticality service without giving the compared approaches different mode trigger information.

4.4. Metrics and Statistical Analysis

Metrics are reported separately by criticality class and include released and completed jobs, assessed and right-censored deadlines, the deadline miss ratio (DMR), the completion rate (CR), the response and waiting time percentiles, delivered service, preemptions, and queue behavior. The DMR includes only jobs whose deadlines occur within the experimental horizon, whereas the CR includes all released jobs:

$$\mathrm{DMR} = \frac{N_{\mathrm{miss}}}{N_{\mathrm{assessed}}}, \qquad \mathrm{CR} = \frac{N_{\mathrm{completed}}}{N_{\mathrm{released}}}.$$

The delivered service fraction is defined as the executed work divided by the offered work. For job j, $x_j \in [0,1]$ denotes the delivered work fraction, and the overall work-weighted utility is $U_w = \sum_j C_j u(x_j) / \sum_j C_j$. We evaluate concave $u(x) = \sqrt{x}$, linear $u(x) = x$, and convex $u(x) = x^2$ utility functions to represent different service distribution preferences, together with Jain's index for cross-task service fairness.

The response time is measured only for jobs that are completed. It is therefore interpreted jointly with the deadline miss ratio (DMR) and completion rate to avoid favoring a scheduling policy merely because jobs with longer or more difficult execution trajectories fail to complete. For the waiting time, we count every job that started running, regardless of whether it finished, and we separately track jobs that were considered but never actually started.

Policy comparisons are matched by scenario, load, and random seed. Because multiple design cells share a seed, inference is performed on seed-level mean paired differences rather than treating cells as independent observations. Percentile-bootstrap 95% confidence intervals use 10,000 resamples of these seed-level means with bootstrap seed 20260709 [24]; Wilcoxon signed rank tests use the same means when at least one paired difference is nonzero [25]. Controlled and full-matrix analyses contain 30 independent seed clusters, whereas held-out testing contains 10. Raw cell counts and descriptive means are also reported, and all-zero contrasts are stated directly. Jain's index is used where cross-task fairness is relevant [26]. Cell-level Holm-adjusted tests and effect size estimates are retained in Supplementary File S1 as sensitivity analyses; the principal conclusions rely on effect magnitudes, matched seed-level comparisons, and seed-clustered confidence intervals.

5. Experimental Results

The six leading training configurations shared the same utilization thresholds and differed mainly in the recovery hold duration. Although training favored a 2000 ms hold, validation selected 500 ms: the corresponding scores were 0.158375 and 0.157355, respectively. The shorter hold improved low-criticality utility and completion without changing the validation high-criticality DMR or response time and was therefore fixed before evaluation on the held-out partition.

5.1. RQ1: Held-Out Emergency Performance

Table 6 summarizes the held-out emergency performance for seeds 20–29. Static criticality-first EDF and all three RT-DASH variants produced the same mean high-criticality DMR (0.15675) and completed-job response time (106.60 ms), whereas EDF and the period-based fixed-priority baseline produced higher DMRs of 0.62722 and 0.91667, respectively. This agreement shows that the principal observed high-criticality benefit is attributable to criticality-first dispatch rather than to mode switching itself. Hysteresis, service degradation, and recovery-aware backlog handling are therefore evaluated through the mechanism-specific and low-criticality service analyses in Sections 5.2 and 5.3 rather than being credited

with additional high-criticality improvements. Figure 4 shows the corresponding variation across emergency profiles and seeds.

Table 6. Held-out emergency results averaged over three emergency families and ten seeds per family.

Variant	High DMR	High Mean Response (ms)	Low Completion	Low Linear Utility
Static criticality-first EDF	0.15675	106.60	0.76771	0.90278
EDF	0.62722	250.59	0.99736	0.99847
Period-based fixed priority (RMS for periodic tasks)	0.91667	274.90	0.85790	0.97314
RT-DASH default	0.15675	106.60	0.78712	0.90958
RT-DASH tuned	0.15675	106.60	0.82095	0.92059
RT-DASH recovery-aware	0.15675	106.60	0.99307	0.99413

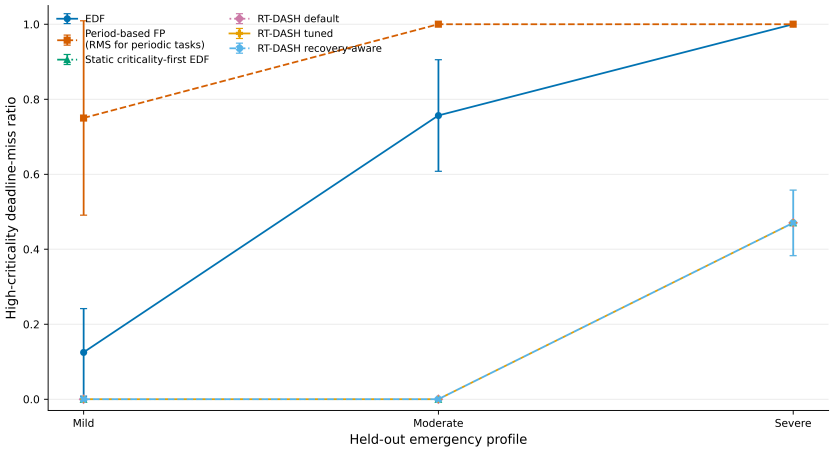


Figure 4. Held-out high-criticality deadline miss ratio by emergency profile and policy. Points show means over ten seeds; error bars are 95% t intervals. Curves with identical values overlap.

The reported mean high-criticality response includes completed jobs only; the DMR includes every high-criticality job for which the deadline was assessed.

The matched seed-level analysis is consistent with the aggregate held-out comparison. Recovery-aware RT-DASH lowered the high-criticality deadline miss ratio (DMR) by 0.4705 compared to EDF, with a seed-clustered 95% confidence interval (CI) for this reduction of between 0.4267 and 0.5208 ($p = 0.002$). It also reduced the high-criticality DMR by 0.7599 compared to the period-based fixed-priority baseline (95% CI: 0.6921–0.8179; $p = 0.002$). Both the tuned and recovery-aware versions of RT-DASH had exactly the same high-criticality DMR in all 30 paired emergency scenarios. Across all seven held-out test cells, tuning increased low-criticality linear utility by 0.00831 relative to the default configuration, with a seed-clustered 95% CI of 0.00581 to 0.01148 ($p = 0.002$).

5.2. RQ2: Mechanism Ablations

The mechanism-specific ablations separate the effects of dispatch ordering, semantic event entry, hysteresis, optional service degradation, and adaptive activation; recovery-aware backlog handling is evaluated separately in Section 5.3. Static criticality-first EDF serves as the control for dispatch ordering. The results show that criticality-first dispatch accounts for the principal observed high-criticality benefit once this ordering is active, whereas semantic event entry determines how promptly it is activated, hysteresis suppresses transition chatter, and adaptive activation avoids imposing criticality-first ordering continuously outside emergency conditions. Optional degradation increases low-criticality

completion under service stress by redistributing the available computation across more jobs rather than increasing the total delivered service fraction. Table 7 reports the corresponding paired effects and confidence intervals; the transition chatter visualization is provided in Supplementary File S1.

Table 7. Mechanism-specific paired comparisons from the controlled study.

Mechanism	Paired Contrast	Metric	Mean Difference	95% CI	Interpretation
Event trigger	RT-DASH-no event trigger	High DMR (emergency)	−0.1233	[−0.1334, −0.1135]	Semantic event entry reduces the observed high-criticality DMR before load-based evidence accumulates
Hysteresis	No hysteresis-RT-DASH	Transitions (chatter)	+44.7	[+44.11, +45.41]	Dual thresholds/hold suppress oscillation
Optional degradation	RT-DASH-no degradation	Low completion (stress)	+0.02188	[+0.02148, +0.02230]	More jobs complete, but service amount is redistributed
Adaptive activation	RT-DASH-static criticality EDF	Low completion (adaptive cells)	+0.02009	[+0.01960, +0.02059]	Avoids permanent criticality-first ordering when not needed

5.3. RQ3: Recovery-Aware Backlog Scheduling

Across the seven held-out test cases, recovery-aware ordering increased low-criticality completion by 0.22886 relative to the tuned RT-DASH (95% CI: 0.21844–0.23869; $p = 0.002$), while the high-criticality DMR remained unchanged in every matched pair. The low-criticality p95 waiting time decreased by 0.98 ms (95% CI: 0.39–1.47 ms; $p = 0.020$), a small absolute effect. Relative to permanent static criticality-first EDF, recovery-aware RT-DASH increased low-criticality linear utility by 0.12591 (95% CI: 0.12290–0.12874; $p = 0.002$). Figure 5 shows the corresponding held-out utility comparison under the three utility assumptions.

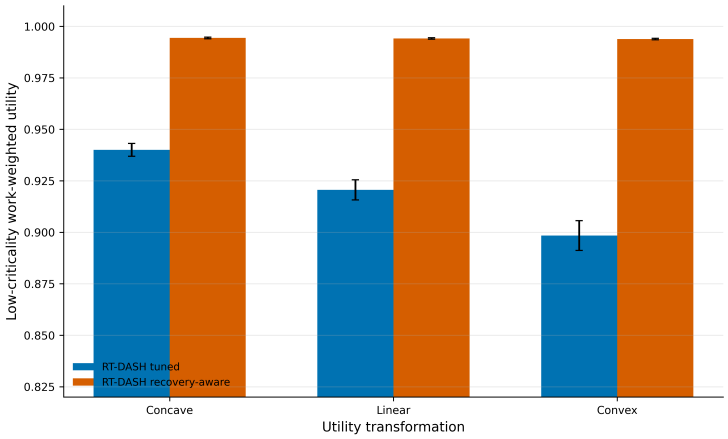


Figure 5. Held-out low-criticality work-weighted utility under concave, linear, and convex value assumptions. Bars show means over ten seed-level means after averaging the three held-out emergency profiles within each seed; error bars are 95% t intervals across seeds.

Across the optimized 30-seed matrix, recovery-aware scheduling increased low-criticality completion by 0.18755 and work-weighted utility by 0.07842, 0.09784, and 0.12025 under the concave, linear, and convex utility functions, respectively, while the high-criticality DMR and completed-job response time remained unchanged. Jain's fairness index increased by 0.01129 and the mean number of mode transitions decreased by 0.80714; the trade-offs were a 2.81607 ms increase in the low-criticality p95 waiting time and a

0.04682 increase in the Critical Mode fraction. Table 8 reports the corresponding raw-pair counts, seed-clustered confidence intervals, and statistical tests.

Table 8. Paired comparison across the full optimized 30-seed matrix.

Metric (Recovery-Aware-Tuned)	Raw Pairs (Seed Clusters)	Mean Difference	95% Seed-Cluster Bootstrap CI	Seed-Level Wilcoxon
High DMR	510 (30)	0.00000	[0.00000, 0.00000]	Not applicable—all seed-level differences were zero
Low completion	840 (30)	0.18755	[0.18210, 0.19299]	$p < 0.001$
Low utility (concave)	840 (30)	0.07842	[0.07664, 0.08023]	$p < 0.001$
Low utility (linear)	840 (30)	0.09784	[0.09543, 0.10024]	$p < 0.001$
Low utility (convex)	840 (30)	0.12025	[0.11711, 0.12338]	$p < 0.001$
Jain fairness	840 (30)	0.01129	[0.01075, 0.01186]	$p < 0.001$
Low p95 wait (ms)	840 (30)	2.81607	[2.54809, 3.08530]	$p < 0.001$
Mode transitions	840 (30)	−0.80714	[−0.85000, −0.76190]	$p < 0.001$
Critical mode fraction	840 (30)	0.04682	[0.03975, 0.05381]	$p < 0.001$

Note: The high-criticality DMR is based on 510 raw pairs because it is defined only for cells containing high-criticality releases. The other metrics use 840 raw pairs. Inferential results use 30 seed-level means after averaging the eligible fixed design cells within each seed. Since every high-DMR difference was zero, no Wilcoxon test is applicable to this contrast.

5.4. RQ4: Robustness and Boundary Analyses

The threshold and hysteresis settings primarily affect how often RT-DASH enters Critical Mode and how long it remains there; once Critical Mode is activated, high-criticality ordering is governed by the dispatch key. The sensitivity study used a 50 ms monitoring interval, whereas the main controlled and selected-configuration experiments used 100 ms. Longer recovery holds reduced rapid exits from Critical Mode but prolonged the reduced-service state; validation selected the 500 ms hold over the 1000 and 2000 ms alternatives without changing the high-criticality deadline miss ratio. Scalability tests increased the task count from 8 to 128 at nominal utilizations of 0.75, 0.95, and 1.15, with scheduling decisions and the maximum ready-queue length increasing with the task count and offered load. The host runtime is reported only as a simulator diagnostic and is not interpreted as the embedded scheduler overhead or as evidence of general asymptotic performance.

The trace mapping study produced an important caution. Five contiguous OpenSHS windows were evaluated under relaxed, nominal, and stressed budget/deadline assumptions, and the outcomes changed with these assumptions. An activity trace does not reveal the WCET or deadline of an embedded task. Figure 6 therefore treats this dependence as a construct validity result rather than unexplained variation.

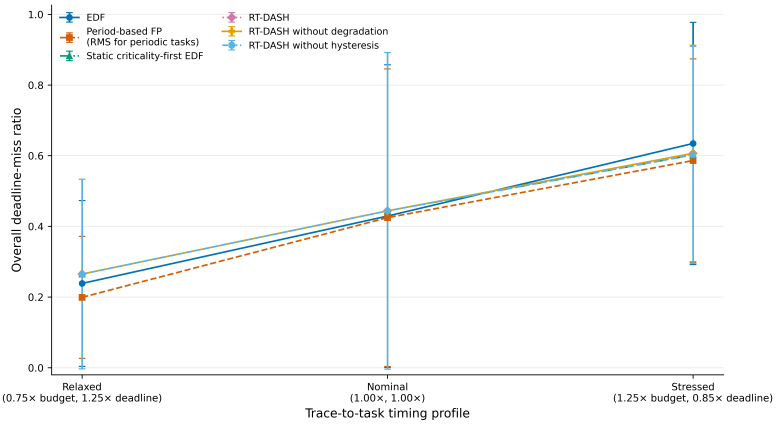


Figure 6. Overall deadline miss ratio under relaxed, nominal, and stressed trace-to-task timing profiles. Points average five contiguous OpenSHS-derived windows; error bars are 95% t intervals across windows.

5.4.1. Demand Bound Screening

The demand bound screen combines periodic-task demand bound functions with bounded event-driven arrivals over tightly packed intervals and event-synchronized release points. It is used only as a diagnostic workload feasibility screen, not as a comprehensive schedulability proof. Table 9 shows that some moderate emergency, severe emergency, recovery backlog, and optional service cases remain infeasible even after minimum service adjustment; deadline failures in these cases therefore cannot be attributed solely to the dispatch policy. In contrast, all held-out overload and threshold chatter cases pass the minimum service screen, indicating that their modeled demand does not exceed the tested diagnostic bound under the minimum service assumptions; passing this screen does not establish schedulability or guarantee deadline satisfaction.

Table 9. Held-out demand bound diagnostic screening.

Scenario	Full Feasible Fraction	Minimum Service Feasible Fraction	Max. Full Demand Ratio	Max. Minimum Service Ratio
Emergency mild	0.70	1.00	1.062	0.767
Emergency moderate	0.00	0.10	1.567	1.246
Emergency severe	0.00	0.00	2.576	2.177
Optional service stress	0.00	0.50	2.538	1.236
Overload	0.10	1.00	1.014	0.618
Recovery backlog	0.00	0.50	1.597	1.267
Threshold chatter	0.00	1.00	1.184	0.875

5.4.2. Execution Overhead and Release Jitter Robustness

Robustness was evaluated with modeled execution overhead increments of 0%, 5%, 10%, and 20% and release jitter of 0, 20, and 50 ms. At 20 ms jitter, recovery-aware RT-DASH and static criticality-first EDF retained a high-criticality DMR of zero through 20% modeled overhead in both tested scenarios, whereas that of EDF increased from 0.7275 to 1.0000; low-criticality utility decreased with overhead for all policies. Table 10 reports the detailed results, and Figure 7 summarizes the DMR trend. These overhead values are sensitivity perturbations, not measurements of the FreeRTOS scheduling cost, context switch latency, instrumentation overhead, or other platform-specific runtime expenses.

Table 10. Robustness results at 20 ms release jitter, averaged over the configured robustness scenarios.

Variant	Modeled Overhead Fraction	High DMR	Low Linear Utility
Static criticality-first EDF	0.00	0.00000	0.91409
Static criticality-first EDF	0.05	0.00000	0.86958
Static criticality-first EDF	0.10	0.00000	0.83375
Static criticality-first EDF	0.20	0.00000	0.72318
EDF	0.00	0.72750	0.97764
EDF	0.05	0.83333	0.97540
EDF	0.10	0.92417	0.97118
EDF	0.20	1.00000	0.89976
RT-DASH tuned	0.00	0.00000	0.92705
RT-DASH tuned	0.05	0.00000	0.87962
RT-DASH tuned	0.10	0.00000	0.83733
RT-DASH tuned	0.20	0.00000	0.72728
RT-DASH recovery-aware	0.00	0.00000	0.97443
RT-DASH recovery-aware	0.05	0.00000	0.97219
RT-DASH recovery-aware	0.10	0.00000	0.96780
RT-DASH recovery-aware	0.20	0.00000	0.89542

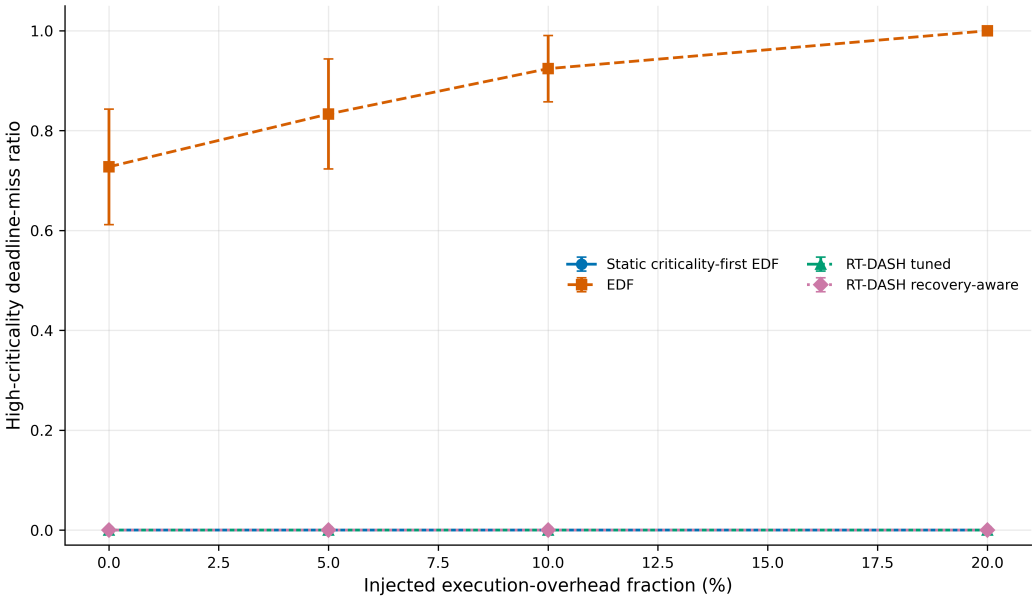


Figure 7. High-criticality deadline miss ratio versus injected execution overhead at 20 ms release jitter, averaged across the emergency moderate and recovery backlog scenarios. Error bars are 95% t intervals across ten seed-level means after averaging the two robustness scenarios within each seed. The x-axis represents the modeled overhead, not a measured firmware cost.

5.4.3. Separate EDF-VD Benchmark

The separate EDF-VD benchmark provides the directly related adaptive mixed-criticality and service degradation comparison described in Section 4.3. It uses dual-criticality implicit-deadline task sets with low- and high-mode execution budgets, probabilistic budget overruns, and virtual deadlines in low mode. Classical EDF-VD drops low-criticality work after a mode switch, whereas the degraded-quality variant retains a prescribed service fraction [5]. For fairness, RT-DASH-style control is also activated by execution budget overruns in this benchmark, so the adaptive policies receive the same type of mode transition information. The comparison therefore evaluates high-criticality deadline behavior and low-criticality service retention under common dual-criticality semantics without implying identical application models.

Table 11 shows that static criticality-first EDF, EDF-VD drop, and RT-DASH-style control each produced an aggregate high-criticality deadline miss ratio (DMR) of 0 in this benchmark, whereas degraded-quality EDF-VD produced a high-criticality DMR of 0.00012. The degraded-quality EDF-VD variant achieved a mean completed high-criticality response time of 14.39 ms, a low-criticality completion rate of 0.87485, and a retained low-criticality service fraction of 0.64766. The corresponding RT-DASH-style values were 14.02 ms, 0.87087, and 0.66068, respectively. Thus, both service degradation approaches achieved near-zero high-criticality deadline miss rates under the common benchmark while exhibiting slightly different low-criticality service trade-offs: degraded-quality EDF-VD produced a marginally higher low-criticality completion rate, whereas RT-DASH-style control retained a larger fraction of the offered low-criticality service. These results are interpreted as evidence of different service control characteristics under the benchmark assumptions, rather than as establishing the universal superiority of either scheduling approach; the corresponding response time and service retention visualization is provided in Supplementary File S1.

Table 11. Aggregate results from the separate EDF-VD dual-criticality benchmark.

Policy	High DMR	High Mean Response (ms)	Low Completion	Low Service
Static criticality-first EDF	0.00000	9.88	0.95655	0.96129
EDF	0.01602	38.87	0.98775	0.98696
EDF-VD drop	0.00000	12.95	0.64365	0.56106
EDF-VD degraded quality	0.00012	14.39	0.87485	0.64766
RT-DASH-style MC	0.00000	14.02	0.87087	0.66068

5.5. Software Validation and Evidence Boundary

The validation included 25 passing Python 3.13.5 tests, the passing C++ test target, 15,780 simulation executions, and five C++ host policy runs. The regeneration of the deterministic training and optimized matrix outputs reproduced the stored results exactly; host-dependent wall clock timing fields were excluded from these reproducibility checks.

The firmware images were compiled and linked successfully, and the required SVC, PendSV, and SysTick handlers are present. This establishes build integrity for the evaluated targets but does not constitute runtime validation. Because neither QEMU execution nor physical STM32 deployment was performed, the study did not measure the mode transition latency, UART data loss, runtime stack consumption, context switch overhead, or hardware deadline performance. These properties therefore remain to be evaluated through emulator-based or physical platform experiments.

6. Conclusions

This article presented RT-DASH, a dual-mode scheduling framework that combines semantic emergency signals with sampled load monitoring to transition between deadline-oriented Normal Mode dispatch and criticality-first Critical Mode dispatch. Across 15,780 simulation runs, criticality-first ordering reduced the observed high-criticality deadline miss ratio relative to conventional EDF and period-based fixed-priority scheduling under the modeled emergency workloads. RT-DASH activates this stricter ordering adaptively rather than applying it continuously during routine operation. The results therefore attribute the principal observed high-criticality deadline benefit to criticality-first dispatch, while semantic event entry, hysteresis, service degradation, and recovery-aware backlog handling primarily affect the activation timing, mode stability, and retained low-criticality service. The recovery-aware refinement improved low-criticality completion and work-weighted utility in both the held-out and full-matrix evaluations while preserving the observed high-criticality deadline performance. The ablation studies further showed that immediate event entry and hysteresis make distinct contributions to the controller behavior. Additionally, demand screening identified the cases that remained infeasible under the tested diagnostic bound even after minimum service adjustment, highlighting the limits of the evaluated workload conditions and policies.

Overall, the results provide reproducible simulation-based evidence for the behavior and performance of RT-DASH under the modeled workloads considered in this study. They should not be interpreted as proof of schedulability, a worst-case response time guarantee, or evidence that all deadlines will be met for arbitrary task sets or physical smart home deployments. Establishing such real-time guarantees would require formal schedulability analyses together with platform-specific execution time and blocking assumptions.

As future work, we aim to evaluate RT-DASH on real Cortex-M hardware using measured or conservatively estimated worst-case execution times (WCET), detailed blocking and interrupt analysis, additional trace sources, and service value assessments tailored to

the target application. Furthermore, we will consider identifying and demonstrating the sufficient conditions for schedulability feasibility in RT-DASH.

Supplementary Materials: The following supporting information can be downloaded at https://e.pcloud.link/publink/show?code=XZDLk77Zw3rgRgogO3zEIA2BJ9vQhRYBHtXX (accessed on 29 August 2026). Supplementary File S1 provides the source code, configurations, run-level outputs, statistical analyses, representative traces, figure generation materials, validation logs, reproduction instructions, software environment information, manifests, licensing information, and SHA-256 integrity checks used to reproduce and verify the reported simulation results. It contains no physical-board timing measurements, measured WCET values, energy measurements, or hardware deadline performance data.

Author Contributions: The software implementation and testing were conducted primarily by M.N.B. The corresponding author is J.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable. The study reports a software simulation and analysis of a publicly distributed simulated smart home dataset and involved no human participants or animals.

Informed Consent Statement: Not applicable.

Data Availability Statement: The OpenSHS-derived Smart Home Dataset used in this study is publicly available through Mendeley Data, Version 1, at https://doi.org/10.17632/zgsw84b2ff.1 (accessed on 14 August 2026). The code, configurations, run-level outputs, statistical tables, representative traces, figures, firmware build artifacts, manifests, and SHA-256 checksums generated for this study are provided in Supplementary File S1. The original software and supporting materials are distributed under the licenses specified in Supplementary File S1, while the third-party dataset remains subject to its original CC BY 4.0 license. No physical-board timing data were generated in this study.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CI	Confidence interval
DMR	Deadline miss ratio
EDF	Earliest Deadline First
EDF-VD	EDF with Virtual Deadlines
IoT	Internet of Things
RMS	Rate-Monotonic Scheduling
RTOS	Real-time operating system
WCET	Worst-case execution time

References

1. Liu, C.L.; Layland, J.W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM* **1973**, *20*, 46–61. [CrossRef]
2. Vestal, S. Preemptive Scheduling of Multi-Criticality Systems with Varying Degrees of Execution Time Assurance. In Proceedings of the 28th IEEE International Real-Time Systems Symposium, Tucson, AZ, USA, 3–6 December 2007. [CrossRef]
3. Baruah, S.; Burns, A.; Davis, R.I. Response-Time Analysis for Mixed Criticality Systems. In Proceedings of the 32nd IEEE Real-Time Systems Symposium, Vienna, Austria, 29 November–2 December 2011. [CrossRef]
4. Burns, A.; Davis, R.I. A Survey of Research into Mixed Criticality Systems. *ACM Comput. Surv.* **2017**, *50*, 82. [CrossRef]
5. Liu, D.; Spasic, J.; Guan, N.; Chen, G.; Liu, S.; Stefanov, T.; Yi, W. EDF-VD Scheduling of Mixed-Criticality Systems with Degraded Quality Guarantees. In Proceedings of the IEEE Real-Time Systems Symposium, Porto, Portugal, 29 November–2 December 2016. [CrossRef]

6. Chen, G.; Guan, N.; Liu, D.; He, Q.; Huang, K.; Stefanov, T.; Yi, W. Utilization-Based Scheduling of Flexible Mixed-Criticality Real-Time Tasks. *IEEE Trans. Comput.* **2018**, *67*, 543–558. [CrossRef]
7. Boudjadar, J.; Ramanathan, S.; Easwaran, A.; Nyman, U. Combining Task-Level and System-Level Scheduling Modes for Mixed Criticality Systems. In Proceedings of the 2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT), Cosenza, Italy, 7–9 October 2019; pp. 1–10. [CrossRef]
8. Malik, S.; Ahmad, S.; Ullah, I.; Park, D.H.; Kim, D. An Adaptive Emergency First Intelligent Scheduling Algorithm for Efficient Task Management and Scheduling in Hybrid of Hard Real-Time and Soft Real-Time Embedded IoT Systems. *Sustainability* **2019**, *11*, 2192. [CrossRef]
9. Yu, L.; Xie, W.; Xie, D.; Zou, Y.; Zhang, D.; Sun, Z.; Zhang, L.; Zhang, Y.; Jiang, T. Deep Reinforcement Learning for Smart Home Energy Management. *IEEE Internet Things J.* **2020**, *7*, 2751–2762. [CrossRef]
10. Chung, H.-M.; Maharjan, S.; Zhang, Y.; Eliassen, F. Distributed Deep Reinforcement Learning for Intelligent Load Scheduling in Residential Smart Grids. *IEEE Trans. Ind. Inform.* **2021**, *17*, 2752–2763. [CrossRef]
11. Boudjadar, J.; David, A.; Kim, J.H.; Larsen, K.G.; Mikučionis, M.; Nyman, U.M.; Skou, A.J. Statistical and Exact Schedulability Analysis of Hierarchical Scheduling Systems. *Sci. Comput. Program.* **2016**, *127*, 103–130. [CrossRef]
12. Boudjadar, J.; Larsen, K.G.; Kim, J.H.; Nyman, U. Compositional Schedulability Analysis of an Avionics System Using UPPAAL. In Proceedings of the 1st International Conference on Advanced Aspects of Software Engineering (ICAASE 2014), Constantine, Algeria, 2–4 November 2014; CEUR-WS; Volume 1294, pp. 140–147. Available online: https://ceur-ws.org/Vol-1294/paper16.pdf (accessed on 29 August 2026).
13. Picard, S.; Paolillo, A.; Geeraerts, G.; Goossens, J. Exact Schedulability Test for Sporadic Mixed-Criticality Real-Time Systems Using Antichains and Oracles. In Proceedings of the 32nd International Conference on Real-Time Networks and Systems (RTNS), Porto, Portugal, 6–8 November 2024. [CrossRef]
14. Esmaili, A.; Nazemi, M.; Pedram, M. Energy-Aware Scheduling of Task Graphs with Imprecise Computations and End-to-End Deadlines. *ACM Trans. Des. Autom. Electron. Syst.* **2020**, *25*, 11:1–11:21. [CrossRef]
15. Stavrinides, G.L.; Karatza, H.D. The Impact of Input Error on the Scheduling of Task Graphs with Imprecise Computations in Heterogeneous Distributed Real-Time Systems. In Proceedings of the International Conference on Analytical and Stochastic Modeling Techniques and Applications (ASMTA), Venice, Italy, 20–22 June 2011; pp. 273–287. [CrossRef]
16. Stavrinides, G.L.; Karatza, H.D. An Energy-Efficient, QoS-Aware and Cost-Effective Scheduling Approach for Real-Time Workflow Applications in Cloud Computing Systems Utilizing DVFS and Approximate Computations. *Future Gener. Comput. Syst.* **2019**, *96*, 216–226. [CrossRef]
17. Jiang, Z.; Dai, X.; Audsley, N.C. HIART-MCS: High Resilience and Approximated Computing Architecture for Imprecise Mixed-Criticality Systems. In Proceedings of the 42nd IEEE Real-Time Systems Symposium (RTSS), Virtual Conference, 7–10 December 2021; pp. 290–303. [CrossRef]
18. Huang, P.; Kumar, P.; Stoimenov, N.; Thiele, L. Interference Constraint Graph—A New Specification for Mixed-Criticality Systems. In Proceedings of the 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), Cagliari, Italy, 10–13 September 2013; pp. 1–8. [CrossRef]
19. Ren, J.; Phan, L.T.X. Mixed-Criticality Scheduling on Multiprocessors Using Task Grouping. In Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems (ECRTS), Lund, Sweden, 8–10 July 2015; pp. 25–34. [CrossRef]
20. Gu, X.; Easwaran, A.; Phan, K.-M.; Shin, I. Resource Efficient Isolation Mechanisms in Mixed-Criticality Scheduling. In Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems (ECRTS), Lund, Sweden, 7–10 July 2015; pp. 13–24. [CrossRef]
21. Erickson, J.P.; Kim, N.; Anderson, J.H. Recovering from Overload in Multicore Mixed-Criticality Systems. In Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Hyderabad, India, 25–29 May 2015; pp. 775–785. [CrossRef]
22. FreeRTOS. FreeRTOS Kernel Book. Available online: https://github.com/FreeRTOS/FreeRTOS-Kernel-Book (accessed on 14 August 2026).
23. Lutta, P. Smart Home Dataset; Mendeley Data, Version 1, 2024. Available online: https://data.mendeley.com/datasets/zgsw84b2ff/1 (accessed on 14 August 2026).
24. Efron, B.; Tibshirani, R.J. *An Introduction to the Bootstrap*; Chapman & Hall/CRC: New York, NY, USA, 1993. [CrossRef]
25. Wilcoxon, F. Individual Comparisons by Ranking Methods. *Biom. Bull.* **1945**, *1*, 80–83. [CrossRef]
26. Jain, R.; Chiu, D.-M.; Hawe, W.R. *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems*; DEC Research Report TR-301; Digital Equipment Corporation: Hudson, MA, USA, 1984. Available online: https://classes.engineering.wustl.edu/~jain/papers/fairness.htm (accessed on 14 August 2026).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.