On Probabilistic Schedulability Analysis

Jaeyoung Lee

Yonsei University

March 31, 2026

Part I

Based on the Survey

Probabilistic Real-Time Systems

A Survey of Probabilistic Schedulability Analysis Techniques for Real-Time Systems
(LITES'19: Leibniz Transactions on Embedded Systems)

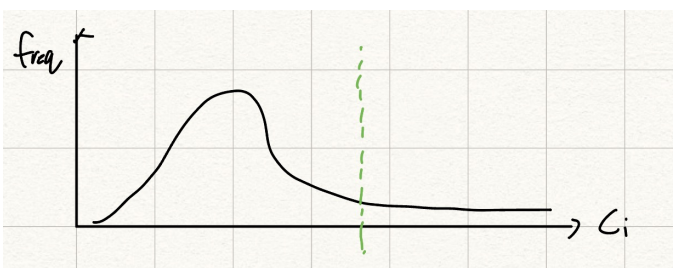
This presentation starts from the survey on probabilistic real-time systems and then returns to:

- **Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems** (ECRTS'18)
- **Efficiently Approximating the Worst-Case Deadline Failure Probability Under EDF** (RTSS'21)
- **Reducing Worst-Case Deadline Failure Probability for EDF Scheduling** (RTSS'25)

Problem Setting

What is Probabilistic Timing Guarantee?

- Classical real-time systems rely on **deterministic timing guarantees**
- Execution times exhibit **significant variability** in practice
- Therefore, timing guarantees are reformulated in a **probabilistic manner**
- $P(\text{deadline miss}) \leq \rho$



Classical vs Probabilistic Real-Time Systems

Classical Real-Time Systems

- **Timing Analysis**
  - Execution time is characterised by a **single upper bound (WCET)**
- **Schedulability Analysis**
  - Computes **Worst-Case Response Time (WCRT)**
  - Checks if: $WCRT \leq D$
  - Guarantees must hold for **all possible scenarios**

Probabilistic Real-Time Systems

- **Execution Time Variability**
  - Execution time is modeled as a **probability distribution**
- **Flexible Timing Requirements**
  - Deadlines are associated with a **maximum acceptable miss probability**
  - $P(\text{deadline miss}) \leq \rho$

Analysis Pipeline

From Execution Time to Deadline Miss Probability

- Step 1: **Probabilistic Timing Analysis**
  - Compute **pWCET distribution**
- Step 2: **Probabilistic Schedulability Analysis**
  - Compute **pWCRT distribution**
- Step 3: **Deadline Analysis**
  - Compute **WCDFP**

Pipeline

$$\text{Execution Time} \rightarrow \text{pWCET} \rightarrow \text{pWCRT} \rightarrow \text{WCDFP}$$

Probabilistic Timing Analysis (pWCET)

- **pWCET (Probabilistic Worst-Case Execution Time)**
  - Upper bound on execution time distribution across all scenarios
- Methodologies:
  - **SPTA** (Static Probabilistic Timing Analysis): an analytical method that uses static code analysis and abstract hardware models to construct pWCET bounds
  - **MBPTA** (Measurement-Based Probabilistic Timing Analysis): a statistical method that uses execution traces and **Extreme Value Theory (EVT)** to estimate the distribution of extreme execution times

Probabilistic Schedulability Analysis (pWCRT)

- pWCRT distribution computes probabilistic worst-case response time
- pWCRT is compared against the deadline to calculate:
  - **Deadline Miss Probability (DMP)**
  - **Worst-Case Deadline Failure Probability (WCDFP)**
- Accounts for scheduling and interference between tasks

From Single Task to Multiple Tasks

Probabilistic Timing Requirement

- Unlike classical systems, probabilistic systems specify:

$$P(\text{deadline miss}) \leq \rho$$

- Timing guarantees are defined in terms of **acceptable miss probability**

Single Task Case

- Response time distribution = execution time distribution
- Therefore, response time distribution is **trivial**

Multiple Tasks Case

- Execution-time distributions need to be combined to form a probabilistic response-time distribution
- This is where the issue of independence between execution times emerges

Fundamental Challenge

Independence of execution times of two jobs from a task

Equation-wise:

$$P(\{X = x\} \cap \{Y = y\}) = P(X = x) \cdot P(Y = y)$$

Idea-wise:

- The event that the execution time of the first job takes a particular value x has no effect on the probability that the execution time of the second job will take some value y

Combining Execution-Time Distributions

- Response time requires combining multiple execution-time distributions
- Typically done using **convolution**
- This needs to assume that the execution times are independent

$$Z = X \otimes Y$$

$$P(Z = z) = \sum_{k=-\infty}^{\infty} P(X = k) \cdot P(Y = z - k)$$

Independence Assumption

In practice, execution times are **not independent** due to:

- **Software state** (e.g., static local variable)
- **Hardware state** (e.g., shared cache line)
- **Input correlation**

Consequence

- Convolution under the assumption of independence may lead to **inaccurate probability estimation**

Definition of pWCET

- pWCET is defined over **all valid scenarios of operation**
- A scenario includes:
  - input sequences
  - software states
  - hardware states

Property

- pWCET distribution is an **upper bound** over all scenarios

Probabilistic Deadline

Classical

$$D_i$$

Probabilistic

$$(D_i, \rho_i)$$

$$P(R_i > D_i) \leq \rho_i$$

- ρ_i shows the maximum acceptable probability that the deadline may be exceeded
- There also exist formulations that look at multiple-deadline probabilistic views instead of a single deadline

Deadline Miss Probability

Deadline miss probability

- In probabilistic view, it can be defined as:
  1. Long-run frequency
  2. Probability of a randomly selected job
  3. Probability bound for a specific job
- It can also be defined in terms of:
  1. a specific task
  2. a group of tasks
  3. the whole system

Deadline Miss Metrics: DMP vs WCDFP

Deadline Miss Probability (DMP)

- **Average behavior over time**
- Interpreted as **long-run frequency**
- Equivalent to the fraction of missed deadlines over time
- Typically applied to **strictly periodic task sets**
- Assumes system behavior repeats over the **hyperperiod (LCM)**

Worst-Case Deadline Failure Probability (WCDFP)

- **Worst-case guarantee for any job**
- Interpreted as the probability that **any specific job** misses its deadline
- Defined using pWCRT:

$$WCDFP_i = P(R_i > D_i)$$

pWCRT and WCDFP: Key Insight

Probabilistic Worst-Case Response Time

- R_i: pWCRT of task τ_i
- Upper bound on response-time distribution for **any job**
- Computed by considering the **worst-case arrival pattern**

Why WCDFP?

- For **sporadic tasks**:
  - exact release times are unknown
  - only the minimum inter-arrival time T_i is given
- Exact analysis requires considering **all possible arrival sequences**
- This is **intractable**
- Therefore, analysis uses a worst-case arrival pattern

Limitation

- Provides a **safe bound**, but introduces **pessimism**

Research Directions

Main Approaches

- **Response Time Analysis**: directly compute response-time distribution
- **Server-based Analysis**: isolate tasks via bandwidth allocation
- **Queueing-based Methods**: model deadline satisfaction using queue dynamics

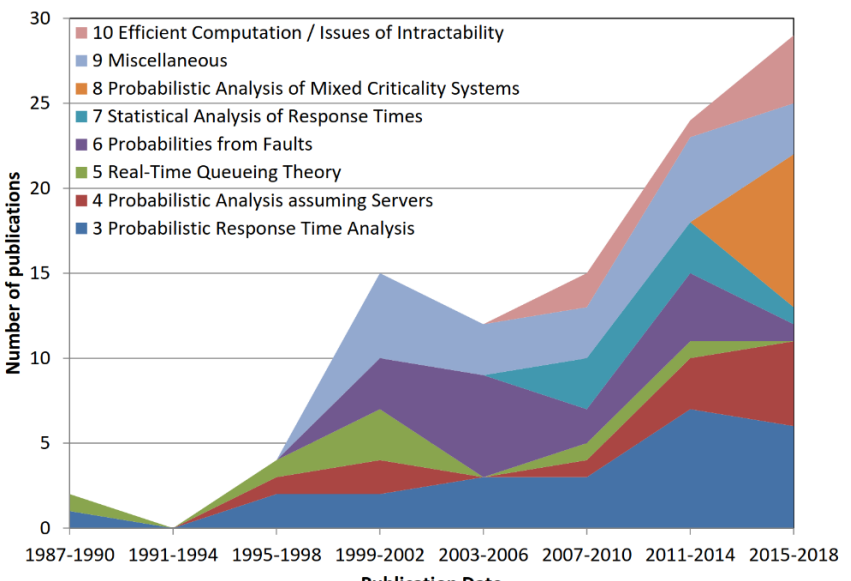
Alternative Modeling Perspectives

- **Fault-aware Analysis**: incorporate probabilistic fault-recovery overhead
- **Statistical Methods**: estimate deadline miss probability from observations
- **Mixed-Criticality Systems**: use probabilistic execution models across criticality levels

Extended Topics

- **Advanced Scheduling Models**: precedence constraints, priority assignment, multiprocessor scheduling
- **Scalability** / **Intractability**: reduce computational complexity of analysis

RESEARCH DIRECTIONS



Part II

Prior Works

From Prior Work to RTSS'25

We now return to the line of papers leading to RTSS'25:

1. **Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems** (ECRTS'18)
2. **Efficiently Approximating the Worst-Case Deadline Failure Probability Under EDF** (RTSS'21)
3. **Reducing Worst-Case Deadline Failure Probability for EDF Scheduling** (RTSS'25)

ECRTS'18: Critical Flaw – State Space Explosion

Job-level Convolution (Before This Work)

- Standard method for computing DMP was to use **job-level convolution**
- If a task τ_i has h execution modes and releases J jobs,

$$\text{number of execution scenarios} = h^J$$

- As J increases, computational cost grows **exponentially**
- This becomes especially severe for systems with long hyperperiods

So the main issue was a **scalability limitation**: computation quickly becomes intractable.

ECRTS'18: Key Idea – Breaking Exponential Complexity

To solve complexity issue, the paper introduces **three main strategies**.

1. Multinomial Distribution (Task-level Grouping)

- Shift from job-level to **task-level analysis**
- The order of execution modes **does not change total workload**
- Use a **Multinomial Distribution** to compute the probability of mode counts
- Instead of convolving each job individually

2. Analytical State Pruning

- Do not branch when a state is always safe or always misses
- If max workload $<$ deadline $\Rightarrow$ 0% miss
- If min workload $>$ deadline $\Rightarrow$ 100% miss

Benefit

- **Mathematically exact pruning**
- No loss of precision

3. Equivalence Class Merging

- Merge:
  - similar workload states
  - low-probability states

Safe Approximation

- Always select a **pessimistic representative**
- Guarantees that a **safe upper bound** is preserved

Part III

Reducing Worst-Case Deadline Failure Probability for EDF Scheduling

AGAIN ON RTSS'25

Will make a recap on the paper using last week's slides

Part IV

INTRODUCTION

Motivation: Deterministic vs Probabilistic Scheduling

Deterministic scheduling analysis

- Examines whether tasks can *strictly* meet their execution deadlines
- Deadline failure is treated as a binary event
- Ensuring zero deadline misses often leads to significant resource waste

Probabilistic scheduling analysis

- Bounds the probability of deadline failure
- Allows better resource utilization while still providing timing guarantees

Why Probabilistic Guarantees Matter

Industry standards also accept system failures with **bounded probabilities**.

- **IEC-61508**: functional safety standard for electrical, electronic, and programmable electronic safety-related systems
- **ISO 26262**: functional safety standard for electrical and electronic systems in road vehicles

Implication:

- In many practical systems, *zero* deadline misses are not strictly required
- A bounded failure probability is often sufficient and more resource-efficient

Paper Goals

This paper aims to reduce the **Worst-Case Deadline Failure Probability (WCDFP)** under EDF scheduling.

Main contributions

1. **Improve WCDFP analysis**
   - Propose a more accurate estimation method
   - Reduce pessimism in the analytical WCDFP
2. **Active-dropping policy**
   - Integrate active dropping into EDF
   - Enable bounded WCDFPs even when the maximum total utilization significantly exceeds 1

Part V

Related Work

Related Work: Big Picture

Most prior work on WCDFP has focused on:

- **Fixed-priority scheduling**

This line of work gradually transitions toward:

- **EDF scheduling**

This paper is positioned in that transition:

- improving WCDFP analysis specifically for EDF
- and further reducing failure probability through policy design

Related Work: Periodic Tasks

Existing WCDFP analysis on periodic tasks

1. **ECRTS 2002**
   - WCDFP analysis method combined with an overrun-handling mechanism that randomized drop jobs that execute more than certain threshold
     => higher utilization expectation but higher error ratio compared to non-dropping system
2. **RTSS 2002**
   - cover both static and dynamic priority scheduling
     => base idea : deadline failure prob converges to stable value as the task set is executed over many hyper periods.
     => prob : can not provide upper bound on WCDFP before the convergence
3. **ECRTS 2015**
   - WCDFP analysis for both static and dynamic priority tasks
   - Limitation: exponential computation complexity with respect to the number of jobs

Related Work: Sporadic Tasks

Sporadic tasks are characterized by a minimum inter-arrival time.

RTSS 2021

- First WCDFP analysis under EDF scheduling for sporadic tasks
- Core idea:
  - estimate overload probability for each necessary interval
  - sum these probabilities as the WCDFP of a task

Two efficiency tricks used in the baseline

1. Limit the number of intervals by bounding failure probability with the probability that the processor does not idle
2. Reduce convolution cost for each interval through state pruning

Part VI

Task and System Model

System Model

We consider a **uniprocessor** system with a task set

$$\Gamma = \{\tau_1, \tau_2, \ldots, \tau_\xi\}$$

of independent sporadic tasks scheduled under **preemptive EDF**.

Each task τ_i is defined by

$$\tau_i = \langle C_i, T_i, D_i \rangle$$

where:

- C_i: execution time random variable
- T_i: minimum inter-arrival time
- D_i: relative deadline

We assume **constrained deadlines**:

$$D_i \leq T_i$$

for all tasks.

System Model

The execution time C_i is upper-bounded by a **probabilistic Worst-Case Execution Time (pWCET)** distribution:

$$\mathcal{P}_{C_i} = \{(C_i^l, P_i^l)\}$$

- Each job $J_{i,j}$ is i.i.d. according to the distribution of task τ_i
- The model captures probabilistic execution-time behavior instead of using only a single deterministic WCET

Extra Assumption:

- Any job that misses its deadline is **immediately aborted**

Job Abort VS Job Continue

$\rightarrow$ makes the job independent

$\rightarrow$ also WCDFP talks about system level failure

IMPORTANT BUSY-INTERVAL DEFINITIONS

Definition 1. *A processor busy interval is a time interval during which the processor is continually occupied by pending jobs. Given a job of τ_i released at t_a with absolute deadline $d = t_a + D_i$, a **deadline-d busy interval** is a processor busy interval containing only jobs whose deadlines do not exceed d.*

Processor busy interval

- A time interval during which the processor is continually occupied by pending jobs
- In other words, the processor has **no idle time** in that interval

Deadline-*d* busy interval

- A processor busy interval containing only jobs whose deadlines do not exceed *d*
- Why this matters:
  - only jobs with deadlines $\leq d$ can interfere with the target job whose absolute deadline is *d*

WCDFP Definitions

WCDFP of a job

- The maximum probability that the job misses its deadline over all possible arrival sequences

WCDFP of a task

- The WCDFP among any job released by that task

WCDFP of the system

- The maximum WCDFP among all tasks in the task set

Part VII

Review of the Existing Analysis

Baseline Analysis under EDF

The existing RTSS 2021 approach consists of two main steps:

1. Identify a specific **worst-case release pattern** that leads to the WCDFP
2. Calculate the deadline failure probability under that release pattern

Worst-Case Release Pattern

The baseline is built on the **overload condition**.

An interval is overloaded when:

$$\text{cumulative processor demand} > \text{interval length}$$

Thus, the worst-case release pattern is the one that:

- generates the largest number of jobs in a deadline-*d* busy interval
- and hence maximizes cumulative demand

Definition 2. *The **demand** within the deadline-d busy interval $[t_s, d]$ is the cumulative execution time of all jobs released at or after t_s and having deadlines no later than d.*

Alignment to Deadline d

Under the worst-case release pattern,

- tasks are aligned so that one job from each task has absolute deadline d
- before that point, jobs are released periodically with minimum inter-arrival time

Intuition:

- among all sporadic arrivals, periodic release at minimum spacing maximizes the number of released jobs

Definition 3. *A task τ_i is said to be **aligned** to d if there exists a job $J_{i,j}$ with an absolute deadline equal to d, and τ_i has been released periodically before $J_{i,j}$.*

Lemma 1 (Based on Lemma 9 and Corollary 1 in [17])**.** *For each fixed interval $[t, d]$, the probability of overload, defined as the probability that an overload condition occurs within the interval, is maximized when all tasks are aligned to d.*

Key Observation: Overload Does Not Always Mean Miss at d

An overload in a deadline-d busy interval does **not** always imply a deadline miss exactly at d.

Reason:

- a job may miss its deadline *before* d
- once that happens, the job is aborted
- therefore, its remaining interference disappears

Nevertheless:

- the worst-case release pattern still provides an **upper bound** on deadline failure probability
- this is why the baseline analysis remains safe but can be pessimistic

Two Core Questions in the Baseline

To bound deadline failure probability, the baseline must answer:

1. What are the possible start points of deadline-d busy intervals?
2. How do overload probabilities in those intervals determine WCDFP?

Start Points of Deadline-d Busy Intervals

Any deadline-d busy interval $[t_s, d]$ must start at

$$t_s \in A(0, d - D_{\min})$$

where

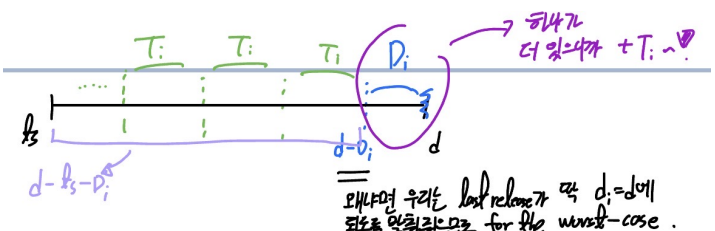
$$D_{\min} = \min_{\tau_i \in \Gamma} D_i$$

So the analysis only needs to consider these candidate start points.

Demand Computation: Number of Jobs

Let $N_i(t_s, d)$ denote the number of jobs of τ_i that contribute to the demand in the deadline-d busy interval $[t_s, d]$.

$$N_i(t_s, d) = \left\lfloor \frac{d - t_s + T_i - D_i}{T_i} \right\rfloor$$



Let $S(t_s, d)$ denote the **total demand** generated by all jobs in this interval.

Both $N_i(t_s, d)$ and $S(t_s, d)$ are defined under the **worst-case release pattern**.

Demand Computation via Convolution

Definition 4. *For two independent random variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ with probability mass functions $\mathcal{P}_X$ and $\mathcal{P}_Y$, the probability of their sum $\boldsymbol{Z} = \boldsymbol{X} + \boldsymbol{Y}$ is the* ***convolution*** *$\mathcal{P}_X \otimes \mathcal{P}_Y$, where $\mathbb{P}\{\boldsymbol{Z} = z\} = \sum_{k=-\infty}^{k=+\infty} \mathbb{P}\{\boldsymbol{X} = k\}\mathbb{P}\{\boldsymbol{Y} = z - k\}$.*

Definition 5. *The sum of k i.i.d random variables, each following the distribution $\mathcal{P}_X$, follows the distribution $\mathcal{P}_X^{\otimes k}$, defined as $\mathcal{P}_X^{\otimes k} \coloneqq \mathcal{P}_X \otimes \cdots \otimes \mathcal{P}_X$ (with k occurrences of $\mathcal{P}_X$).*

Since all jobs of a task are i.i.d., the total demand can be computed by **convolution**.

- Sum of independent random variables
- Exact demand distribution for each interval

Lemma 2 (Task level convolution proposed in [34])**.** *For any deadline-d busy interval $[t_s, d]$,*

$$\boldsymbol{S}(t_s, d) \sim \mathcal{P}_{\boldsymbol{S}(t_s,d)} \coloneqq \bigotimes_{i=1}^{\xi} \mathcal{P}_{C_i}^{\otimes \mathrm{N}_i(t_s,d)} \tag{1}$$

FINAL BASELINE WCDFP BOUND

Theorem 1 (Theorem 1 in [17]). *Consider any job J with an absolute deadline d. The deadline failure probability of J is bounded as* $\sum_{t_s \in \mathsf{A}(d-H, d-D_{\min})} \mathbb{P}\{\boldsymbol{S}(t_s, d) > d - t_s\}$.

The baseline computes overload probability for every deadline-d busy interval starting after $d - H$ and before $d - D_{\min}$.

Then the WCDFP is bounded by:

$$\sum_{t_s \in A(d-H,\, d-D_{\min})} \mathbb{P}\{S(t_s, d) > d - t_s\}$$

That is, the analysis **sums interval-wise overload probabilities**.

Why the Existing Analysis is Pessimistic

The key issue is **duplicate counting**.

The analysis should conceptually do three things:

1. determine which jobs can affect the job with deadline d
2. identify all execution-time patterns that can cause a miss at d
3. add the probabilities of those patterns

Problem:

- the same execution-time pattern can appear as an overload in *multiple* intervals
- the baseline sums them independently
- this causes overestimation

EXAMPLE 1: TASK SET

Consider the following task set:

$$\langle C_1 \sim \begin{pmatrix} 5 & 15 \\ 0.8 & 0.2 \end{pmatrix}, T_1 = D_1 = 20 \rangle$$

$$\langle C_2 \sim \begin{pmatrix} 9 \\ 1 \end{pmatrix}, T_2 = D_2 = 20 \rangle, \qquad \langle C_3 \sim \begin{pmatrix} 1 \\ 1 \end{pmatrix}, T_3 = D_3 = 40 \rangle$$

Here, the interval length is $H = 40$.

EXAMPLE 1: EXECUTION-TIME PATTERNS

Definition 6. *Given a set of jobs, where each job follows the same pWCET distribution as its corresponding task, an* ***execution time pattern*** *for the set is the collection of execution times by selecting a fixed execution time for each job.*

Within $[d - H, d]$, there are four possible execution-time patterns:

$$\begin{aligned} \{15, 15, 9, 9, 1\} &: \ 0.04 \\ \{5, 5, 9, 9, 1\} &: \ 0.64 \\ \{15, 5, 9, 9, 1\} &: \ 0.16 \\ \{5, 15, 9, 9, 1\} &: \ 0.16 \end{aligned}$$

Candidate interval start points:

$$A(d - H, d - D_{\min}) = \{d - 40, \ d - 20\}$$

EXAMPLE 1: DUPLICATE COUNTING

The deadline miss occurs in cases (a) and (d).

However, pattern (a) contributes overload in *both*

$$[d-20, d] \quad \text{and} \quad [d-40, d]$$

so it is counted twice by the baseline.

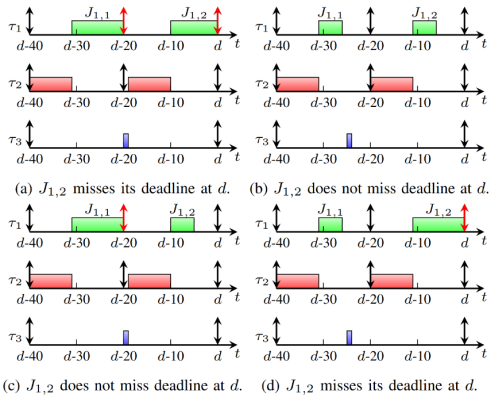
$$\boldsymbol{S}(d-20,d) \sim \mathcal{P}_{C_1} \otimes \mathcal{P}_{C_2} = \begin{pmatrix} 14 & 24 \\ 0.8 & 0.2 \end{pmatrix}$$

$$\boldsymbol{S}(d-40,d) \sim \mathcal{P}_{C_1}^{\otimes 2} \otimes \mathcal{P}_{C_2}^{\otimes 2} \otimes \mathcal{P}_{C_3} = \begin{pmatrix} 29 & 39 & 49 \\ 0.64 & 0.32 & 0.04 \end{pmatrix}$$

By Theorem 1, the WCDFP of τ_1 is

$$\sum_{t_s \in \mathsf{A}(d-40, d-\mathrm{D}_{\min})} \mathbb{P}\{\boldsymbol{S}(t_s, d) > d - t_s\}$$

$$= \mathbb{P}\{\boldsymbol{S}(d-20,d) > 20\} + \mathbb{P}\{\boldsymbol{S}(d-40,d) \geq 40\} = 0.24$$



(a) $J_{1,2}$ misses its deadline at d. (b) $J_{1,2}$ does not miss deadline at d.

(c) $J_{1,2}$ does not miss deadline at d. (d) $J_{1,2}$ misses its deadline at d.

Fig. 1: Execution time patterns of Example 1 in the worst-case arrival sequence. The execution times of $\{J_{1,1}, J_{1,2}\}$ are (a) $\{15, 15\}$, (b) $\{5, 5\}$, (c) $\{15, 5\}$, or (d) $\{5, 15\}$.

EXAMPLE 1: OVERESTIMATION

The true deadline failure probability is:

$$0.8 \cdot 0.2 + 0.2 \cdot 0.2 = 0.2$$

But the baseline analysis gives:

$$0.24$$

So the same execution-time pattern is counted more than once, leading to an overestimation of:

$$0.04$$

Example 2: A More Severe Pathology

Now add one more task:

$$\langle C_4 \sim \begin{pmatrix} 0 \\ 1 \end{pmatrix}, T_4 = D_4 = 1 \rangle$$

Your intended point here is:

- adding a task that should not materially worsen the failure probability
- nevertheless creates many more candidate intervals
- which amplifies duplication in the baseline

As a result, the baseline can produce a highly inflated value such as:

$$0.84$$

Root Cause of Pessimism

Root cause:

- the baseline sums overload probabilities interval by interval
- but execution-time patterns are **not disjoint** across intervals

Therefore:

- identical failure-causing patterns are repeatedly counted
- the resulting WCDFP bound is safe, but unnecessarily pessimistic

This is exactly the problem the new analysis aims to fix.

Part VIII

New Analysis

New Analysis: Main Idea

Goal:

- solve the **duplication problem** in the existing analysis

Instead of identifying execution-time patterns that **cause** deadline failure, we focus on the patterns that **never** encounter a deadline failure at d.

That is, rather than summing overlapping failure-causing events, we characterize the set of **safe execution patterns**.

Why?

$\rightarrow$ failure event from small to big interval (O)

$\rightarrow$ suceed event from small to big interval (X)

Reformulating the Probability

The event

$$\mathbb{P}\left\{\bigcap_{t_s \in A(0,\, d-D_{\min})} S(t_s, d) \leq d - t_s\right\}$$

provides a **lower bound** on the probability that no deadline miss occurs at d.

Therefore,

$$1 - \mathbb{P}\left\{\bigcap_{t_s \in A(0,\, d-D_{\min})} S(t_s, d) \leq d - t_s\right\}$$

provides an **upper bound** on the probability that a job misses its deadline at d.

Lemma 3. *Consider any job J that has its absolute deadline equal to d. The deadline failure probability of J at d, is bounded by P_W, where*

$$P_W = 1 - \mathbb{P}\left\{\bigcap_{t_s \in \mathsf{A}(0,d-D_{\min})} \boldsymbol{S}(t_s, d) \leq d - t_s\right\} \quad (2)$$

Why This Avoids Duplication

Recall the issue in the baseline:

- the same execution-time pattern can overload multiple intervals
- summing interval-wise overload probabilities counts such patterns repeatedly

New perspective:

- directly keep only the execution-time patterns that remain safe
- once a pattern causes overload in some shorter interval, discard it
- never count it again later

So each execution-time pattern is handled **once**, instead of being repeatedly added across intervals.

Incremental Evaluation over Intervals

We evaluate candidate deadline-d intervals in **ascending order of interval length**.

For each interval:

1. check the cumulative demand
2. discard execution-time patterns that lead to overload
3. continue until reaching the maximum interval length

Thus, the remaining patterns are exactly those that have not failed in any shorter interval considered so far.

Conditional Safe Demand

We define

$$S^h(t,d) := S(t,d)\,|_{t_s \in A(t,d-D_{\min})}\ \ S(t_s,d) \le d - t_s$$

as the total demand in $[t,d]$ **conditioned on no deadline miss in all shorter candidate intervals**.

Interpretation:

- $S(t,d)$: total demand in interval $[t,d]$
- $S^h(t,d)$: total demand only over execution-time patterns that have **survived** all previous shorter intervals

This is the key object that lets the new analysis remove duplicated failure-causing patterns.

Incremental Extension from Shorter to Longer Intervals

Instead of recomputing each interval from scratch:

- start from a shorter safe interval
- extend it to a larger interval
- incrementally add only the **additional demand**

Lemma 4. *For any* $t_{s1}, t_{s2} \in \mathsf{A}(0, d - D_{\min})$ *and* $t_{s2} < t_{s1}$, *let the additional demand in* $[t_{s2}, d]$ *than in* $[t_{s1}, d]$ *in the worst-case arrival sequence be* $\boldsymbol{S}(t_{s2}, t_{s1})$. *Then,*

$$\boldsymbol{S}(t_{s2}, t_{s1}) \sim \mathcal{P}_{\boldsymbol{S}(t_{s2}, t_{s1})} \coloneqq \bigotimes_{i=1}^{\xi} \mathcal{P}_{C_i}^{\otimes \mathrm{N}_i(t_{s2}, t_{s1})}, \qquad (3)$$

where $\mathrm{N}_i(t_{s2}, t_{s1}) = \left\lfloor \frac{d - t_{s2} - D_i}{T_i} \right\rfloor - \left\lfloor \frac{d - t_{s1} - D_i}{T_i} \right\rfloor$.

So the analysis progresses from short intervals to long intervals while preserving only the safe execution-time patterns.

Need for a Finite Analysis Horizon

A practical issue remains:

- in principle, deadline-d busy intervals can extend arbitrarily far into the past
- so the number of candidate intervals is infinite

Hence, we need a way to make the analysis **finite** by bounding how far back we need to consider candidate intervals.

BUSY-INTERVAL BASED BOUNDING IDEA

The key idea is to use the probability that a longer interval is **busy**.

Lemma 5 (Lemma 12 in [17]). *For the arrival sequence under the worst-case release pattern, consider any interval* $[t_1, d]$ *with* $t_1 \in \mathsf{A}(0, d - D_{\min})$. *The deadline failure probability contributed by any interval* $[t_s, d]$ *with* $t_s \in \mathsf{A}(0, d - D_{\min})$ *and* $t_s < t_1$ *is upper-bounded by* $\mathbb{P}_{busy}\{[t_1, d]\}$. *Here,* $\mathbb{P}_{busy}\{[t_1, d]\}$ *denotes the probability the that the processor is continuously busy executing jobs with deadlines* $\leq d$ *throughout* $[t_1, d]$.

Intuition:

- if a shorter interval near d is not busy, then a longer interval extending further back cannot be a deadline-d busy interval
- therefore, the probability that $[t_1, d]$ is busy upper-bounds the probability that there exists some earlier start time $t_s < t_1$

Bounding the Remaining Failure Probability

Using the busy-interval argument, WCDFP can be bounded as

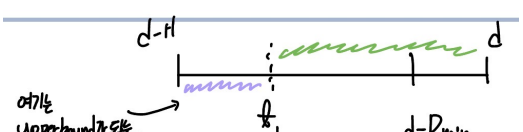
$$\mathbb{P}_{\text{busy}}\{[t_1, d]\} + \sum_{t_s \in A(t_1, d - D_{\min})} \mathbb{P}\{S(t_s, d) > d - t_s\}.$$

Interpretation:

- the summation handles the candidate intervals we still analyze explicitly
- the busy probability term upper-bounds the contribution of all earlier intervals

So this converts the infinite set of intervals into:

- a finite explicit part
- plus one residual upper bound



Bounding the Busy Probability

The busy probability can itself be upper-bounded as

$$\mathbb{P}_{\text{busy}}\{[t_1, d]\} \leq \mathbb{P}\left\{ S(t_1, d) + \sum_{i=1}^{\xi} C_i \geq d - t_1 \right\}.$$

Reason:

- one additional job per task may have been released before t_1
- but still execute inside the interval $[t_1, d]$

So we conservatively add one extra execution-time contribution from each task.

Problems with the Original Busy Bound

However, this original busy bound still has two issues:

1. it may involve **more jobs than necessary** when bounding the maximum execution request in the interval
2. it does **not exclude** execution-time patterns that have already caused overload in shorter intervals

So although it makes the analysis finite, it still retains unnecessary pessimism.

Refined Busy Probability

To solve this, the paper again proceeds from shorter intervals to longer intervals.

Define

$$\mathbb{P}^{h}_{\text{busy}}\{[t_1, d]\}$$

using the **safe-demand** random variable $S^h(t, d)$ instead of the original $S(t, d)$.

This means:

- we only consider execution-time patterns that have survived all shorter intervals
- the residual busy probability is computed on this refined set

Lemma 6.

$$\mathbb{P}^{h}_{busy}\{[t_1, d]\} \leq \mathbb{P}\{\boldsymbol{S}^h(t_1, d) + \sum_{i \in \mathcal{I}} \boldsymbol{C}_i \geq d - t_1\}, \quad (5)$$

where $\mathcal{I} = \left\{ i \in \{1, 2, \ldots, \xi\} \middle| \left\lceil \frac{d-t_1}{T_i} \right\rceil - \left\lfloor \frac{d-t_1+(T_i-D_i)}{T_i} \right\rfloor = 1 \right\}.$

TIGHTER ADDITIONAL-JOB BOUND

The paper also tightens the crude bound

$$\sum_{i=1}^{\xi} C_i$$

to

$$\sum_{i \in \mathcal{I}} C_i,$$

where $\mathcal{I}$ contains only tasks that can actually contribute one additional carry-in job.

The two counting terms are:

$$\left\lceil \frac{d - t_1}{T_i} \right\rceil \qquad \text{and} \qquad \left\lfloor \frac{d - t_1 + (T_i - D_i)}{T_i} \right\rfloor.$$

Interpretation:

- $\left\lceil \frac{d-t_1}{T_i} \right\rceil$: maximum number of jobs that may execute in $[t_1, d]$
- $\left\lfloor \frac{d-t_1+(T_i-D_i)}{T_i} \right\rfloor$: jobs already counted as arriving after t_1 and having deadlines no later than d

Only tasks with a difference of 1 need an extra carry-in term.

Final Refined Bound

The refined idea is exactly the same as before:

- move from shorter intervals to longer intervals
- preserve only the safe execution-time patterns
- add only the necessary extra demand

Theorem 2. *Consider any job J that has its absolute deadline equal to d. The deadline failure probability of J at d, is bounded by P_W, where*

$$P_W = 1 - \mathbb{P}\left\{ \bigcap_{t_s \in \mathsf{A}(t_1, d - D_{\min})} \boldsymbol{S}(t_s, d) \leq d - t_s \right\} + \mathbb{P}^h_{busy}\{[t_1, d]\}$$

As a result:

- duplication is avoided
- the residual busy-interval bound is tighter
- the overall WCDFP bound becomes less pessimistic

Part IX

Active Dropping Policy

Active Dropping Policy: Motivation

The paper argues that, in addition to improving the analysis, **intentionally dropping jobs** can also reduce the overall failure probability.

Key idea:

- analysis alone cannot eliminate all pessimism
- modifying the scheduling policy itself can further reduce WCDFP

Motivating Example

Consider the following task set:

$$\left\langle C_1 \sim \begin{pmatrix} 10 & 19 \\ 0.9 & 0.1 \end{pmatrix}, T_1 = D_1 = 20 \right\rangle, \quad \left\langle C_2 \sim \begin{pmatrix} 1 \\ 1 \end{pmatrix}, T_2 = D_2 = 20 \right\rangle,$$

$$\left\langle C_3 \sim \begin{pmatrix} 10 \\ 1 \end{pmatrix}, T_3 = D_3 = 40 \right\rangle.$$

Using the new analysis, the WCDFP is:

$$0.19$$

Why Intentional Dropping Helps

Suppose we intentionally drop every job of τ_1 when its execution demand reaches 19.

Then the effective execution-time model becomes:

$$C_1 \sim \begin{pmatrix} 10 \\ 1 \end{pmatrix}$$

As a result:

- no deadline miss occurs among the jobs that are allowed to continue
- the only failures are the jobs that we intentionally dropped

So the system job failure probability becomes:

$$0.1$$

Hence, active dropping can nearly halve the WCDFP:

$$0.19 \rightarrow 0.1$$

Why Analysis Alone Is Not Enough

This example shows an important limitation of pure analysis:

- even with a tighter bound, the analysis still only estimates failure probability
- it does not change the underlying system behavior

Active dropping changes the actual execution pattern:

- high-demand jobs are proactively removed
- overall interference is reduced
- deadline failures of other jobs can be prevented

ACTIVE DROPPING UNDER EDF

The paper integrates active dropping into EDF using two thresholds:

$$TH_i^1 \quad \text{and} \quad TH_i^2 \qquad (TH_i^1 > TH_i^2)$$

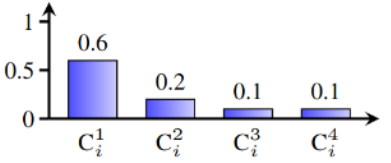
R1. For each task τ_i, any job executing beyond certain predefined execution time thresholds will be dropped with a specific probability. Each task τ_i has at most two thresholds, TH_i^1 and TH_i^2 with $\mathrm{TH}_i^1 > \mathrm{TH}_i^2$.

R2. The first threshold, TH_i^1, is set to C_i^{A-1}, where A is the smallest index that satisfies $\sum_{k=A}^{|\mathcal{P}_{C_i}|} \mathrm{P}_i^k \leq \mathrm{DP}$. The drop probability at TH_i^1, denoted as $\mathrm{DP}_i(\mathrm{TH}_i^1)$, is set to 1.

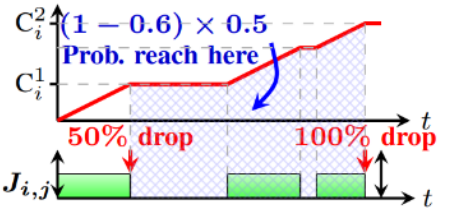
R3. Let B be the largest index such that $\sum_{k=B}^{|\mathcal{P}_{C_i}|} \mathrm{P}_i^k > \mathrm{DP}$. The second threshold, TH_i^2, is set to C_i^{B-1} if $B \geq 2$ and to 0 otherwise. The drop probability at TH_i^2, denoted as $\mathrm{DP}_i(\mathrm{TH}_i^2)$, is set to $(\mathrm{DP} - \sum_{k=A}^{|\mathcal{P}_{C_i}|} \mathrm{P}_i^k)/\mathrm{P}_i^B$, if A exists as above, or $\mathrm{DP}/\mathrm{P}_i^B$ otherwise.

EXAMPLE OF THRESHOLD-BASED DROPPING

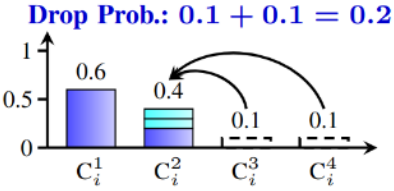
We can understand the active-dropping idea through the following example.



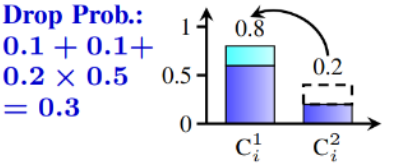
(a) Original $\mathcal{P}_{C_i}$.



(b) Active-dropping control: $\mathrm{DP}_i(\mathrm{C}_i^2) = 1$ and $\mathrm{DP}_i(\mathrm{C}_i^1) = 0.5$.



(c) $\mathcal{P}_{C_i}$ after drop control at $\mathrm{TH}_i^1 = \mathrm{C}_i^2$.



(d) $\mathcal{P}_{C_i}$ after drop control at $\mathrm{TH}_i^1 = \mathrm{C}_i^2$ and $\mathrm{TH}_i^2 = \mathrm{C}_i^1$.

Fig. 2: Changes in the pWCET distribution $\mathcal{P}_{C_i}$ of task τ_i when $\mathrm{DP} = 0.3$ and $\mathrm{C}_i^3 + \mathrm{C}_i^4 \leq \mathrm{DP} < \mathrm{C}_i^2 + \mathrm{C}_i^3 + \mathrm{C}_i^4$.

The figure illustrates:

How Do We Select the Drop Probability?

Let the drop probability be DP.

Then the total WCDFP is decomposed into two parts:

1. probability that a job is dropped:
$$DP$$
2. probability that a job is *not* dropped but still misses its deadline:
$$(1 - DP)\hat{p}$$

Therefore, the total failure probability is:
$$DP + (1 - DP)\hat{p}$$

Theorem 3. *Under the active-dropping rules, given a drop probability DP and the corresponding analytical deadline failure probability $\widehat{P}$, the actual WCDFP of the system is $DP + (1 - DP)\widehat{P}$.*

Interpretation of the Objective

The objective is to choose DP so that

$$DP + (1 - DP)\hat{p}$$

is minimized.

Trade-off:

- larger DP reduces overload and may reduce deadline misses
- but it also directly increases the number of intentionally dropped jobs

Speeding Up Active Dropping

The paper also proposes a faster variant:

- use the **Chernoff bound**
- instead of exact convolution

Advantage:

- much faster computation

Limitation:

- it cannot identify which execution-time patterns have already caused overload
- so it cannot exploit the full pattern-filtering benefit of the new convolution-based analysis
- this improves speed, but sacrifices some tightness

Part X

EXPERIMENTS

Experimental Setting

The paper compares three methods:

Base

- Aggregates overload probabilities over all possible deadline-d busy intervals

New

- Identifies execution patterns that guarantee meeting the deadline
- Reuses computation incrementally from shorter intervals to longer intervals

Active-Drop

- Proactively drops jobs using probability DP
- Uses two thresholds (TH_1, TH_2) to reduce effective system utilization

The methods are evaluated using **synthetically generated task sets**.

Experimental Results

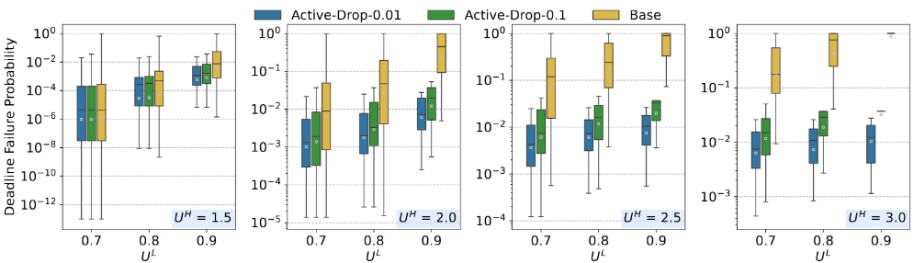


Fig. 4: WCDFP calculated using Chernoff bound for $U^L \in [0.7, 0.9]$ and $U^H \in [1.5, 3.0]$, with $\xi = 5$ and $|\mathcal{P}_{C_i}| = 2$.

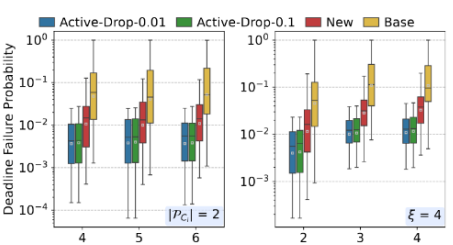


Fig. 5: WCDFP calculated through convolution for $\xi \in [4, 5, 6]$ and $|\mathcal{P}_{C_i}| \in [2, 3, 4]$, with $U^L = 0.9$ and $U^H = 2.5$.

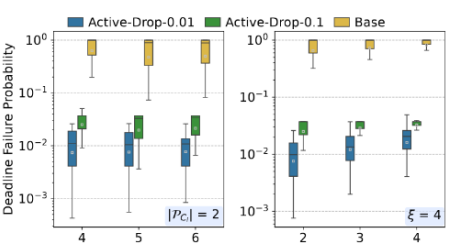


Fig. 6: WCDFP calculated using Chernoff bound for $\xi \in [4, 5, 6]$ and $|\mathcal{P}_{C_i}| \in [2, 3, 4]$, with $U^L = 0.9$ and $U^H = 2.5$.