From Tracepoints to Timeliness: A Semi-Markov Framework for Predictive Runtime Analysis

Benno Bielmeier
Technical University of Applied Sciences Regensburg
Regensburg, Germany
benno.bielmeier@othr.de

Ralf Ramsauer
Technical University of Applied Sciences Regensburg
Regensburg, Germany
ralf.ramsauer@othr.de

Takahiro Yoshida
Tokyo University of Science
Tokyo, Japan
yoshida@ee.kagu.tus.ac.jp

Wolfgang Mauerer
Technical University of Applied Sciences Regensburg
Siemens AG, Technology
Regensburg/Munich, Germany
wolfgang.mauerer@othr.de

***Abstract*—Detecting and resolving violations of temporal constraints in real-time systems is both, time-consuming and resource-intensive, particularly in complex software environments. Measurement-based approaches are widely used during development, but often are unable to deliver reliable predictions with limited data.**

This paper presents a hybrid method for worst-case execution time estimation, combining lightweight runtime tracing with probabilistic modelling. Timestamped system events are used to construct a semi-Markov chain, where transitions represent empirically observed timing between events. Execution duration is interpreted as time-to-absorption in the semi-Markov chain, enabling worst-case execution time estimation with fewer assumptions and reduced overhead.

Empirical results from real-time Linux systems indicate that the method captures both regular and extreme timing behaviours accurately, even from short observation periods. The model supports holistic, low-intrusion analysis across system layers and remains interpretable and adaptable for practical use.

***Index Terms*—Probabilistic Timing Analysis, Stochastic Modelling, Real-Time Linux**

I. Introduction

Real-time systems with complex software stacks, comprising multiple interdependent layers and components, pose challenges for traditional worst-case execution time (WCET) analysis [1]. Static timing analysis provides sound and conservative upper bounds with formal guarantees, making it indispensable for safety-critical systems. However, it tends to yield overly pessimistic estimates and requires impractical effort, particularly when confronted with rare timing anomalies and context-dependent behaviours [2]–[4].

This challenge is further exacerbated in emerging high-performance and heterogeneous architectures, such as quantum-accelerated systems, where latency predictability and execution-time characterisation are critical but poorly understood at system level [5]. Similarly, modern hardware platforms employing mechanisms such as static hardware partitioning on RISC-V introduce architectural features that complicate timing analysis due to increased concurrency, resource contention, and architectural opacity [6]. The need for modular, transparent, and verifiable development processes, as exemplified in open source engineering contexts, underscores the importance of reproducibility and empirical validation in timing analysis [7].

Probabilistic timing analysis techniques based on measurement data have emerged to address both issues by offering probabilistic estimates instead of deterministic timing guarantees [1], [8]. Due to their inherent uncertainty, these methods are not meant as alternatives for deterministic methods in (safety) critical domains. They rather provide valuable insights when firm guarantees are not (yet) required, for instance in soft real-time applications, or during development phases of a system. They allow for rapid runtime validation, while avoiding extensive overheads known to be associated with more sophisticated exhaustive analyses. However, existing measurement-based probabilistic timing analyse (MBPTA) approaches typically require large independent and identically distributed (IID) sample sets and assume statistical independence, which can be difficult to ensure on complex hardware and leads to high measurement overheads.

We propose the use of semi-Markov chains (SMCs) to stochastically model the probabilistic execution time of real-time tasks based on a limited number of runtime measurements. This enables efficient timing analysis without requiring extensive system instrumentation, detailed manual modelling, or exhaustive trace evaluation. The resulting model captures both the temporal characteristics—via stochastic transition delays—and the logical execution flow—through the structure of states and transitions. This state-based representation facilitates not only intuitive visualisation and interpretation of distinct execution phases, but also supports flexible refinement guided by domain-specific knowledge of the target system. By combining lightweight measurement with probabilistic modelling, the approach yields practical and scalable insights into timing behaviour, suitable even in early development stages or in settings with restricted observability.

Given a real-time task whose timing behaviour is to be analysed, our approach proceeds as outlined in Figure 1. First, the target system is instrumented ① to capture essential timing events using lightweight tracepoints; these can appear, for instance, in task scheduling, during interrupt processing, or in function calls. The placement of tracepoints is guided by system-specific considerations and expert judgement, depending on the concrete system, application, and analysed tasks, as

This work was supported by the European Union (Project Reference 101083427) and the European Funds for Regional Development (EFRE) (Project Reference 20-3092.10-THD-105).

2325-1301/25/$31.00 ©2025 IEEE
DOI 10.1109/RTCSA66114.2025.00021

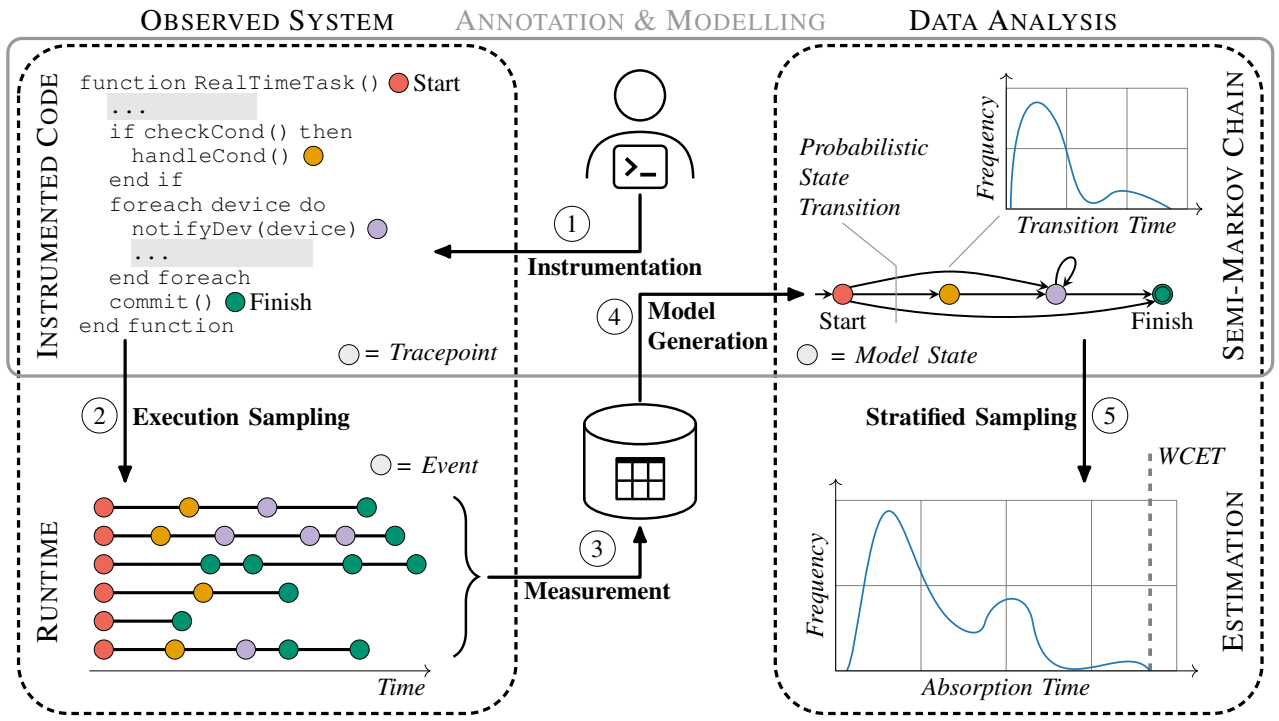

Figure 1. Conceptual overview of our approach. Based on expert knowledge about the architecture of a system under test, our starts with system instrumentation, by enabling existing or inserted tracepoitns ①. We execute the instrumented code repeatedly ② and record the trace data, which we store as tabular data for offline analysis ③. We utilise this data to generate a stochastic model in the form of an semi-Markov chain ④. We analyse the model through sampling by conducting multiple simulation runs ⑤. This analysis yields an estimated worst-case execution time.

detailed in Section IV. Next, the system is executed under realistic conditions ②, and trace events are recorded for subsequent analysis ③. This measurement data then serves as foundation to construct semi-Markov chain models ④. The probability distribution of probabilistic worst-case execution time (pWCET) is derived by simulating numerous state transition sequences ⑤, where each sequence represents the execution of a single task run. The resulting distribution of execution times is then analysed to extract probabilistic temporal properties, such as quantile bounds and estimates of the WCET. These values characterise the task's temporal behaviour under realistic conditions and can inform design decisions, safety arguments, and schedulability assessments in real-time systems.

By constructing SMC models exclusively from trace points in all software layers of the system (*i.e.*, kernel space, user space, and possibly firmware), our framework is holistically applicable to the whole software stack. It can seamlessly integrate data from established tracing mechanisms or custom implementations, underscoring its practical flexibility. Rather than isolated or individual reasoning on single observations, our method probabilistically combines effects of scheduling decisions, hardware influence, and other systemic effects, and thereby simplifies abstract analysis of complex real-world systems.

Moreover, this hybrid approach—positioned between rigorous formal verification and purely empirical analysis—not only mitigates the inherent pessimism of traditional deterministic WCET methods, but also reduces the complexity and rigidity typically associated with such methods [8], [9]. An important practical benefit is to assist developers, particularly as the framework readily provides rapid feedback on the effect of code changes to real-time behaviour, similar to the goal of DevOps efforts. This allows for early identification of performance regressions or improvements, and thereby accelerated development cycles and facilitates targeted optimisations. The low overhead of our measurement technique further supports integration into continuous integration pipelines to provide early feedback to developers.

Our main contributions are

- a hybrid analysis method that integrates measurement-based observations with probabilistic semi-Markov chain modelling for real-time system analysis;
- a demonstration of real-world applicability and the effectiveness of runtime modelling with an evaluation of worst-case latency prediction, measured by cyclictest, a standard Linux RT test suite, on a RISC-V target;
- an assessment of the model's quality with respect to the amount of measurement data to build expressive models.

This paper is structured as follows. Section II provides

background information on timing analysis of real-time system and semi-Markov chain. After the presentation of related work in Section III, we detail our method in Section IV. In Section V we present the implementation components of the approach. Section VI evaluates the approach for worst-case latency estimation of cyclictest on a real-time Linux system. We discuss our results in Section VII and conclude, together with suggestions for future research, in Section VIII.

II. Background

A. Timing Analysis in Real-Time Systems

In real-time computing systems, correctness depends on both, correct functional, and precise temporal behaviour. Payloads (*e.g.*, *real-time* tasks) must meet strict timing constraints. Missed deadlines may lead to severe consequences. Such tasks often exhibit event-driven behaviour, that is, they are initiated or triggered by specific event occurrence or changes in system state. Determining a task's WCET—the maximum execution time a task can exhibit under any feasible scenario—is central to ensure these constraints [10], [11].

Traditionally, WCET analysis relies on either *static* or *measurement-based* approaches [10]. Static approaches that employ modelling and formal verification provide safe bounds, but are often overly conservative owing to the complexity of accurately modelling advanced hardware features such as caches, pipelines, and multicore interaction bounds [10]. They also do not apply well to complex real-world systems like Linux, for reasons ranging from state explosion to practically unmanageable computational requirements [12], [13].

Measurement-based approaches estimate WCET from empirical data obtained through execution profiling. They offer more realistic but non-exhaustive estimates that risk underestimating worst-case conditions. *Hybrid* methods attempt to balance these limitations by combining elements of static and empirical analyses [14], [15].

The mentioned challenges with excessive pessimism or insufficient coverage motivated the transition towards probabilistic timing analysis [1], [16]. Probabilistic methods, notably static probabilistic timing analysis (SPTA) and MBPTA, characterise execution times as probability distributions, and thus explicitly model uncertainty [11]. While SPTA analytically computes execution-time probabilities from program and hardware models, MBPTA statistically derives execution-time distributions from measurements, and extrapolates probabilistic WCET. Although probabilistic techniques significantly reduce pessimism, ensuring validity requires careful assumptions about hardware and execution conditions.

B. Observation of Real-Time Systems

Reliable observation of real-time system behaviour requires accurate and minimally intrusive event capture. While hardware-assisted tracing offers precise timestamps with negligible perturbation, it depends on specialised equipment, increases cost, and often lacks visibility into higher software layers. Software-based tracing has become widespread due to its flexibility across system layers, lack of hardware requirements, and support for fine-grained, customisable instrumentation. However, it can introduce runtime overhead and perturb timing behaviour, potentially distorting observed phenomena. Tracing methods thus span software, hardware-assisted, and hybrid approaches [17]–[20], each balancing accuracy, intrusiveness, and implementation effort [21], [22].

An essential requirement for tracing real-time systems is to maintain minimal overhead, as any perturbation degrades analysis quality. Lightweight instrumentation is therefore critical to preserve runtime characteristics. Efficient data collection methods, such as memory-resident ring buffers or selective logging, help ensure low overhead and practical applicability in real-time contexts.

Trace data may be processed offline or online. Offline analysis collects events for later evaluation, offering flexibility and low hardware demands, albeit with delayed insights. Online analysis, in contrast, processes events during execution for immediate feedback and low memory usage, but is more complex and may require significant computational resources [23].

Linux-based systems benefit from a mature ecosystem of tracing tools. Prominent frameworks such as `ftrace`, `eBPF`, and `LTTng` support efficient in-memory recording, and both offline and online analysis modes. However, their general-purpose design and associated instrumentation overhead can interfere with the timing behaviour of real-time tasks, particularly when analysing latencies in the sub-microsecond range. This limits their applicability in strict real-time scenarios and often necessitates the development of specialised, low-intrusion tracing solutions tailored to the specific characteristics and constraints of the target system.

C. Semi-Markov Chain

A *semi-Markov chain* (SMC) is a stochastic process $Z(t)_{t\in\mathcal{T}}$ defined over a finite state space $\mathcal{Q}$ and an index set $\mathcal{T}$, representing time (discrete or continuous). It models the system's temporal evolution as a trajectory of state transitions. It models the system's temporal evolution as a trajectory of state transitions, as visualised in Figure 2, which shows the system's state at each time. In contrast to standard Markov chains, SMCs permit arbitrary sojourn time distributions between transitions, thus enabling the modelling of non-Markovian, memory-dependent behaviour [24], [25]. In particular, the time spent in a state need not follow the exponential distribution. In particular, the time spent in a state need not follow the exponential distribution, but may also depend on the elapsed time since the last transition.

Formally, an SMC is defined as a sequence of state-time pairs $(J_n, T_n)_{n\in\mathbb{N}}$, where $J_n \in \mathcal{Q}$ denotes the state entered at time $T_n \in \mathcal{T}$, and the sequence (T_n) is strictly increasing. The time spent in state J_n before transitioning is the sojourn time $S_n = T_{n+1} - T_n$, characterised by the cumulative distribution function $F_i(d) = \Pr(S_n \leq d \mid J_n = q_i)$ where $q_i \in \mathcal{Q}$ and $d \in \mathcal{T}$. State transitions are governed by the probability matrix P with entries $P_{ij} = \Pr(J_{n+1} = q_j \mid J_n = q_i)$ where $q_i, q_j \in \mathcal{Q}$ subject to $\sum_j P_{ij} = 1$ for all q_i. We consider a transition $q_i \to q_j$ absent if $P_{ij} = 0$.

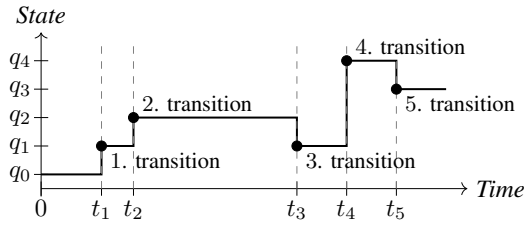

Figure 2. Trajectory of a semi-Markov chain, visualising one realisation of its state evolution over time. The horizontal axis represents continuous time $\mathcal{T}$, while the vertical axis corresponds to the discrete set of states $\mathcal{Q}$. The trajectory starts with initial state $J_0 = q_0$ and transitions at time t_1 to state q_1. Assuming no state change happens after t_5, q_3 is considered an absorbing state.

The joint distribution of transitioning from q_i to q_j within a duration $d \in \mathcal{T}$ is given by the conditional sojourn time distribution $F_{ij}(d) = \Pr(S_n \leq d, J_{n+1} = q_j \mid J_n = q_i)$ which forms the hold time kernel Q.

The initial distribution π assigns probabilities to initial states in which the SMC starts at $t = 0$, satisfying $\pi_i = \Pr(J_0 = q_i)$ with $\sum_{i=1} \pi_i = 1$. We denote the set of states with non-zero initial probability as $\mathcal{S} = \{q_i \in \mathcal{Q} \mid \pi_i > 0\}$.

We assume time-homogeneity, meaning that the transition probabilities P_{ij} and sojourn time distributions F_{ij} are independent of both absolute time and the transition index n.

A state q_i is considered *absorbing* if $P_{ij} = 0$ for all $j \neq i$, equivalently $P_{ii} = 1$. The set of absorbing states is defined as $\mathcal{A} = \{q_i \in \mathcal{Q} \mid P_{ii} = 1\}$. The process is *absorbed* once it enters any $q_a \in \mathcal{A}$. Let $N = \min\{n \geq 0 \mid J_n \in \mathcal{A}\}$ be the index of the absorbing transition, which is also the length of the process' state sequence. The corresponding *time to absorption* is then $T_{\text{abs}} = T_N = \sum_{n=0}^{N-1} S_n$.

III. Related Work

Semi-Markov models (SMMs) are widely used to capture systems with memory-dependent temporal behaviour. Applications span reliability analysis [26], predictive maintenance [27], biological sequence classification [25], [28], and pattern recognition such as speech and handwriting [29]. These models allow arbitrary sojourn time distributions and are often extended with hidden states to represent latent dynamics.

In real-time systems, Bozhko et al. [30] introduce the first axiomatic characterisation of probabilistic worst-case execution time (pWCET). Their Coq-verified framework establishes a mathematically sound upper bound on the execution time distribution under IID assumptions. However, the model is abstract, defers empirical validation, and presumes statistical independence once pWCETs are derived. Our approach contrasts with this by constructing empirical semi-Markov models from lightweight tracepoints, capturing system-wide timing behaviour without relying on the IID assumption. As detailed on in Section IV, execution time is modelled as time-to-absorption, using Gaussian mixture models (GMMs) and stratified sampling to estimate pWCET. Thus, while Bozhko et al. provide formal soundness, our method offers a scalable, tool-supported alternative suited to data-constrained scenarios.

Several state-based MBPTA approaches share conceptual ground with our method. In particular, *Basic Block Measurements*, *Bayesian Networks*, and *Timed Automata* have been used as probabilistic and state-based methods for runtime analysis.

Basic block measurement is a hybrid approach used in program profiling and performance analysis [4], [31]. It identifies execution hotspots, optimisation opportunities, or security vulnerabilities at the CPU level [32], [33]. A program is decomposed into *basic blocks*, i.e., straight-line code sequences with a single entry and exit point, for which metrics such as execution count, time, or resource usage are determined [31], [34]. However, basic block–based techniques generally scale poorly to large code bases comprising millions of blocks, and they are ill-suited for analysing parallel, multithreaded, or highly dynamic programs. They rely on a fixed control-flow structure, rendering them vulnerable in the context of dynamic code generation, self-modifying behaviour, or even common compiler optimisations such as inlining or loop unrolling. To address some of these limitations, the WE-HML approach [35] combines static control-flow analysis with machine learning–based WCET estimation at the binary level. It mitigates the impact of compiler rearrangements by working on compiled code and models non-deterministic timing effects caused by caches via learned pollution-sensitive timing models. By training on a large set of auto-generated code fragments, WE-HML improves coverage and generalisation while avoiding per-application measurements.

Bayesian networks are state-based probabilistic models in which every node represents a random variable and the edges denote conditional dependencies between them. Their strength lies in modelling uncertainty, interference capability, and probabilistic reasoning, enabling the estimation of confidence intervals for WCET [36]. Applied in the field of runtime analysis, they focus on modelling complex probabilistic dependencies among a set of variables using directed acyclic graphs [37], [38]. While SMCs model temporal behaviour explicitly through sojourn time distributions, Bayesian Networks model the transition times implicit through dependencies among variables [39]. Bayesian Networks are well suited when dealing with complex (runtime) dependencies and requiring probabilistic inference to handle uncertainties in system behaviour. However, their reliance on static conditional probabilities may limit the accurate capture of dynamic temporal behaviours, particularly in systems where timing plays a critical role. Moreover, the computational complexity of probabilistic inference in large-scale networks remains a challenge.

State-based models are foundational not only for timing analysis but also for formal verification of functional and temporal properties. Finite-state machines, such as deterministic finite automata (DFAs), specify valid event sequences, while timed automata extend them with real-valued clocks to capture real-time constraints [40]. These models support runtime verification via model checking and online monitoring. Oliveira et al. [41]–[43], for example, employ automata-based monitors

to verify that events occur only in valid temporal and causal contexts. Though these approaches focus on correctness rather than timing estimation, they demonstrate the expressiveness of state-based models. Our semi-Markov framework complements them by adding stochastic timing semantics, enabling probabilistic analysis within a unified, state-based abstraction.

Lesange et al. [44] proposed a framework for measurement-based timing analyses, using an abstract model of synthetic tasks, derived via dynamic instrumentation using Valgrind on FFmpeg. Execution times are aggregated via discrete convolution over syntax-tree-like structures with typed nodes (*e.g.*, loop, conditional), resembling basic block methods. Although this fine-grained abstraction captures specific execution characteristics, it does not constitute dynamic behaviour, flexibility, and the option to extrapolate unobserved behaviours (on possible unseen paths) like our method allows utilizing a semi-Markov framework with a stochastic process.

Friebe et al. [45] model execution times using Markov chains with Gaussian emissions to bound deadline-miss probabilities in reservation servers. To avoid exponential sojourn times, they introduce hidden states. However, their method assumes regularity and is less effective in capturing non-deterministic phenomena such as interrupts. Our semi-Markov approach directly models such variability with mathematically grounded support for complex timing behaviour and minimal measurement intrusion.

IV. Method

Our approach employs SMCs to capture the probabilistic runtime behaviour of tasks, explicitly accounting for both intrinsic task execution and non-deterministic interference from concurrent system activities. We construct an abstract probabilistic model based on runtime measurements. It aims to characterise the distribution of latency values, with particular emphasis on sparsely populated regions where extrapolation is challenging, such as the tail of the latency distribution.

This framework reflects the idea that the execution of a real-time task can be conceptualised as a finite sequence of distinct segments separated by certain events or conditions, for example: task initiation, execution, preemption, resumption, and termination. The duration of each segment is influenced by various factors, such as input data characteristics, other system activities, and microarchitectural state of the hardware. An SMC naturally accommodates this scenario and represents each segment as a state, with the hold-time distribution estimated from the timestamps of observed events.

Unlike methods that assume independent and identically distributed (IID) execution times, our SMC approach retains context by conditioning the timing of each transition on the current state. Moreover, SMCs capture correlations between events; for example, an interrupt that is frequently followed by a cache refill penalty will manifest as a state transition characterised by a heavier-tailed duration distribution. This explicit, state-dependent modelling fills gaps in existing probabilistic timing analyses, offering a more interpretable framework. Consequently, developers can readily identify which phases—such as *waiting for I/O* or *running without preemption*—contribute most to variability and worst-case latency, rather than relying solely on abstract statistical parameters.

A critical challenge in our approach entails balancing overfitting and overestimating worst-case extremes, employing overly conservative distributions. An overfitted model represents generalisation of the system and yields inaccurate predictions, especially in data-sparse regions like the tail regions. Our approach addresses this trade-off. It integrates empirical measurements with a flexible, state-based stochastic framework. This framework robustly extrapolates tail latencies.

In the following, we elaborate on the process of measurement, the incorporation of SMCs, and the construction of the stochastic model based on the recored data.

A. Measurement Process

During the measurement process, trace data is generated, which is later supplied to the generation process of the SMC models. Lightweight instrumentation techniques capture precise timestamps and contextual metadata from relevant system events, ensuring close alignment between observations and the actual system state.

The workflow begins with the setup of instrumentation mechanisms, including kernel tracepoints, user-space logging, and manual instrumentation. Domain experts select events to be traced based on their understanding of system internals, the structure and implementation of real-time tasks, and potential sources of interference, such as task switches, interrupt handling, or function entry and exit. Where tracing spans domain boundaries (*e.g.*, kernel and user space), time consistency is achieved by using a shared monotonic clock.

During execution of a defined scenario—often repeated under varying side conditions—the system records events in chronological order. Each event is annotated with a high-resolution timestamp, a unique identifier, and optional context (*e.g.*, CPU or process ID) used to differentiate concurrent execution contexts.

Following data collection, the dataset is examined for anomalies, including duplicate timestamps, missing entries, and domain desynchronisation. Filtering and consistency checks are then applied to ensure the integrity and fidelity of the captured temporal behaviour.

B. Stochastic Model

We model the runtime behaviour of a task with an SMC. In our formulation, the task's execution time (from initiation to completion) equals the SMC's time-to-absorption. We represent the segmented execution structure by a stochastic sequence of states, where hold-times correspond to transition durations. System events (*e.g.*, interrupts, signals, exceptions, tracepoints, and function entry/exit points) link the observed system to the model by serving as the nodes of the semi-Markov chain. Each realisation of the stochastic process represents one task execution instance, and its time-to-absorption directly reflects its latency. The estimated WCET $\hat{T}_{\max} = \max\{T_{\text{abs}}\}$ is

Table I
MAPPING BETWEEN MODEL COMPONENTS AND MEASUREMENT DATA

MODEL	MEASUREMENT
Transition-Level Mapping	
$\mathcal{Q}$	Event Set
$\mathcal{S} \subset \mathcal{Q}$	Start Event(s)
$\mathcal{A} \subset \mathcal{Q}$	Termination Event(s)
P	Event Sequence
Q	Elapsed Time Between Events
π	Frequency of Start Event(s)
Derived Properties	
T_{abs}	Task Duration
$T_{\max}$	WCET

therefore given by the maximal time-to-absorption T_{abs}. We assume that all executions reach an absorbing state in finite time, excluding non-terminating paths. While loops may occur, the probability of infinite recurrence is assumed to be zero.

Determining $\hat{T}_{\max}$ analytically proves challenging for complex models. We instead simulate the process by sampling iteratively, following the probability for next-step transitions $(J_n)_{n\in\mathbb{N}}$ and the sojourn times $(S_n)_{n\in\mathbb{N}}$ that match the hold-times. For a dataset of n simulation runs, we define $\hat{T}_{\max} = \max\{\sum_{i=0}^{n} T_i\}$.

Although the events' timestamps are inherently discrete—due to finite resolution of hardware timers—we adopt a continuous index set $\mathcal{T} \subseteq \mathbb{R}_+$. This choice simplifies the analysis over large discrete domains and preserves the operational abstraction of a continuous time.

We model hold times between states as univariate random variables and represent their distributions using GMMs. This approach balances flexibility and tractability, allowing the approximation of complex, potentially multimodal distributions while maintaining a manageable number of parameters.

Alternative parametric distributions—such as uniform, normal, or gamma—can also be employed; however, they impose structural assumptions on the shape of the data and are often insufficient to capture irregular or heavy-tailed behaviours. In contrast, GMMs offer a data-driven, non-restrictive alternative that supports modelling across a wide range of empirical scenarios. Our emphasis is therefore on maximising expressive power rather than enforcing specific distributional forms.

C. Model-Construction

We derive a semi-Markov chain model from the runtime traces comprising timestamps, event identifiers, and optional context. Table I summarises the semantic correspondence between measurement data and the model components, distinguishing transition-level elements from derived properties.

The recorded events determine the state space $\mathcal{Q}$. Based on our instrumentation semantics, we manually specify the set of initial states $\mathcal{S}$ and absorbing states $\mathcal{A}$, thereby delineate the system's execution paths present in the dataset.

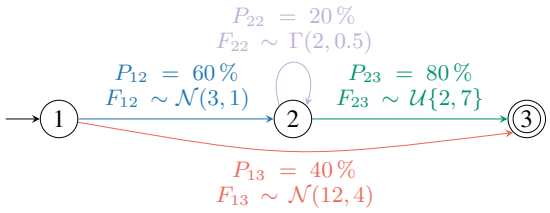

Figure 3. Generator model for synthetic dataset. Starting from the initial state q_1 the model transitions to q_2 in 60 % and directly to the absorbing state q_3 in 40 %. Both transition durations are normally distributed $\mathcal{N}(\mu, \sigma^2)$. The model contains a loop at q_2 which is taken in 20 % with the latency distribution following a gamma distribution $\Gamma(\alpha, \lambda)$. The transition from q_2 to q_3 is uniformly distributed $\mathcal{U}\{a, b\}$.

The empirical initial distribution $\hat{\pi}$ is estimated as the relative frequency of initial states across N observed executions $\hat{\pi}_i = \#_{runs}(J_0 = q_i)/N$, where $\#_{runs}(\cdots)$ denotes the number of runs satisfying the given condition.

Transition probabilities are inferred from observed transitions. The empirical probability of transitioning from q_i to q_j is

$$\hat{P}_{ij} = \frac{\#_{trans}(q_i \to q_j)}{\#_{trans}(q_i \to *)},$$

with $\#_{trans}(q_i \to q_j)$ counting transitions from q_i to q_j, and $\#_{trans}(q_i \to *)$ all transitions originating in q_i. This formulation ensures the matrix $\hat{P}$ is row-stochastic.

Hold times are modelled using GMMs, fitted via expectation-maximisation to match the timestamp differences between source and target events of each transition. For a transition from q_i to q_j, the hold time distribution is expressed as $F_{ij}(t) = \sum_{k=1}^{K} \omega_k \mathcal{N}(t \mid \mu_k, \sigma_k^2)$, where ω_k, μ_k, and σ_k^2 denote the weight, mean, and variance of the k-th component, respectively, and K is the number of mixture components. To ensure plausibility, the distribution is truncated at $t = 0$ and normalised, such that negative durations are excluded and $F_{ij}(t)$ is defined only for $t \geq 0$.

While standard model selection techniques—such as the Akaike information criterion (AIC) or Bayesian information criterion (BIC)—may be employed to determine a suitable value of K, in our setting the number of components is typically chosen by the developer or domain expert based on the observed complexity of the empirical distribution.

D. Conceptual Demonstration of Application

To illustrate our method, we apply it to a synthetic example resembling typical real-time task behaviour, where latency spans from a trigger event to a termination condition. We generate a dataset of approximately 2 700 event-timestamp pairs from 1 000 independent executions of the generator model shown in Figure 3, using predefined (assumed base truth) transition probabilities and sampling transition durations from normal, gamma, or uniform distributions with a random noise value added to emulate realistic measurement conditions and to capture varying characteristics of different execution segments. In the modelling phase, all hold-times—regardless of their

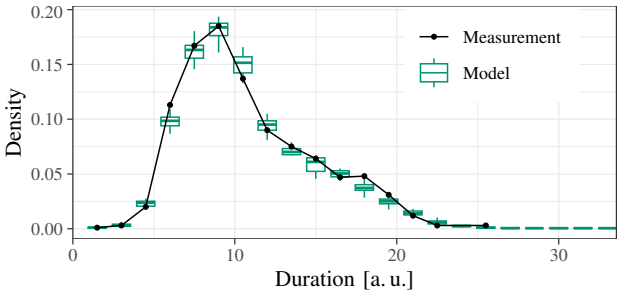

Figure 4. Comparison of latency distributions for the synthetic dataset for empirical measurements and model-based analysis. The black frequency polygon represents the latency distribution (from initiation to termination) derived from the synthetic dataset, while the green box plots show the aggregated density of latencies for each duration bin, as modeled by 24 independent semi-Markov chains.

origin—are re-approximated using GMMs. Distributions with support on negative time ($\Pr(t < 0) > 0$) are truncated at $t = \epsilon > 0$ during sampling to enforce non-negativity.

We partition the generated event log into individual runs using q_1 as the initial and q_3 as the absorbing state with $\pi_1^{(0)} = 1$. From this, 24 independent SMC models over $\mathcal{Q} = \{q_1, q_2, q_3\}$ are reconstructed. Transition probabilities are inferred from sequences in the data, and holding times modeled via GMMs. To estimate the overall latency distribution, each model is simulated 2 000 times. The number of SMC models and the number of runs reflects a trade-off between computational cost and statistical robustness, capturing variability due to stochastic fitting convergence and enabling parallelised evaluation.

Figure 4 demonstrates the close match between empirical and simulated latency distributions, confirming that our method reliably reconstructs timing behaviour under noise. We revisit this example in Sections VI and VII to analyse tail quantiles and maximum values for pWCET estimation.

V. Implementation Components

Our method requires three basic components: (A) tracepoints as data sources of events captured during runtime, (B) a tracing tools to collect and store events, and (C) a data analysis framework that processes the recorded data, generates an SMC model and evaluate its predictions.

The following describes the design and implementation of our custom instrumentation and event tracing mechanism, followed by a brief description of our analysis framework.

A. Timed Tracepoints

To collect timing information with minimal runtime interference, we introduce a custom lightweight tracing mechanism termed *Timed Tracepoints* (TTPs). Conventional tracing frameworks such as `ftrace` or `eBPF`, although expressive and widely supported, may introduce non-negligible overheads that compromise the precision of temporal measurements. To overcome these limitations, timed tracepoints (TTPs) are designed to record only essential data: a unique event identifier, a high-resolution timestamp, and minimal contextual information (*e.g.*, CPU or process ID). This reduction in scope ensures significantly lower overhead while retaining sufficient detail for probabilistic timing analysis.

Each TTP consists of a static probe, that can be inserted across the kernel, and a global handler, statically compiled to avoid dynamic instrumentation costs. Trace events are logged into a statically allocated memory buffer using a global interface function (`ttp_emit`), thereby eliminating the need for dynamic memory allocation and preventing timing perturbations during logging. The interface exposes only a minimal control surface—namely enable, disable, and reset operations—to further reduce runtime impact. After system execution, the contents of the buffer are retrieved from user space and processed in an offline analysis step.

B. Data Analysis Framework

The model construction and statistical evaluation is implemented in R [46], and leverages its ecosystem of parallelised data manipulation, statistical analysis, and visualisation libraries [47]–[49]. The core workflow integrates data preprocessing, transformation, and statistical modelling to construct the SMC model and analyse system runtime behaviour. The complete codebase is published as open-source software.[1]

We generate the semi-Markov chain model and its estimation by processing measurement data in a simple CSV format. First, we transform raw event data into a time-ordered log of transitions that correspond to individual task runs. We scale timestamps to a consistent unit for accuracy and compatibility. We then apply filters to discard incomplete runs and exclude irrelevant events occurring outside the span of the designated start and termination events. Additional checks verify assumptions about the data, such as unique timestamps within each execution run.

Next, we construct the SMC from the refined transition data. We compute transition probabilities and derive transition duration distributions modelled with GMMs, which we fit using the expectation-maximisation algorithm.

Finally, we perform simulation runs to sample the distribution of execution time from the constructed SMC. We generate random samples of the SMC's trajectories to estimate runtime characteristics of the system, including the worst-case execution time. These simulations account for variability in both transition probabilities and sojourn times, yielding probabilistic bounds for runtime analysis.

VI. Evaluation

In the following, we demonstrate our method with cyclictest[2], a widely used benchmarking utility for measuring scheduling latencies in real-time Linux systems. It reports latency as the deviation between a timer's programmed wake-up time and the actual time at which the corresponding task is resumed. While this notion of latency differs from classical execution time—typically defined as the duration from task release to

[1]We published our code and data at https://github.com/lfd/smc-ta.

[2]Cyclictest is part of `rt-tests`, a suite of tools for evaluating real-time behaviour in Linux: https://git.kernel.org/pub/scm/utils/rt-tests/rt-tests.git.

completion—the distinction is immaterial for our approach, which focuses on the temporal distance between a defined start and termination event, irrespective of semantic interpretation.

We adopt cyclictest for three reasons: (i) it exercises the full stack from kernel interrupt handling to user-space scheduling; (ii) it is well understood by kernel developers, including the meaning and structure of its reported jitter; and (iii) it allows us to quantify the instrumentation overhead by comparing latency distributions of annotated and unannotated systems.

In our model, the expected wake-up time is treated as the start event and the actual wake-up time as the absorbing state. The following sections describe the experimental setup, measurement procedure, derived results, and an analysis of robustness under partial data conditions.

A. Experimental Setup

We conduct measurements on a *StarFive VisionFive 3*, a single-board computer featuring a 64 bit RISC-V processor. It runs a real-time enabled Linux kernel (`6.11.0-rc5`), configured with the `PREEMPT_RT` patch stack. Any unused devices and functionalities are disabled. To further RT-tune the system, system noise is reduced by the isolation of CPUs via boot parameter to pin real-time tasks to specific cores.

To minimise latency perturbations while preserving high precision, the kernel was patched with our lightweight TTPs.[3] Tracepoints are inserted at critical locations targeting the key sources of latency in our evaluation: the `riscv_timer_interrupt` event handler (capturing entry ② and exit ③) and within the scheduler to record context switches ① (`TRACE_SCHED_SWITCH`) as well as task wake-ups ④ ⑤ (`TRACE_SCHED_WAKEUP` and `TRACE_SCHED_WAKING`). To capture expected and actual wake-up times, two additional TTPs are integrated into the cyclictest application using the same time base as the ones in the kernel. Data from both the kernel and the application were recorded independently and later merged for offline analysis on a secondary machine.

Cyclictest is executed on the real-time Linux system with a real-time priority of 90 and FIFO scheduling to ensure deterministic behaviour. It was configured to capture 1 000 measurements per second over a period of 5 min yielding a dataset of 600k individual runs. The worst-case latency observed during our measurement is 51.58 µs.

B. Results

To capture the variability inherent in both system execution and probabilistic modelling, we generate 24 independent SMC instances from the recorded event data. This ensemble-based approach ensures that model construction is robust to stochastic effects in both data and parameter estimation. Each SMC encodes transition durations using a GMM with four components. This number was empirically chosen as a trade-off between model flexibility and overfitting, providing sufficient expressiveness to approximate multi-modal timing behaviours without introducing excessive estimation variance or computational cost. Figure 5 illustrates a simplified representation of the resulting state structure and transitions, reflecting the temporal ordering of key events, including both kernel-level TTPs and cyclictest measurements.

[3]Our TTP implementation and instrumentation of the Linux kernel is available at https://github.com/lfd/linux/tree/smc-ta-ttp.

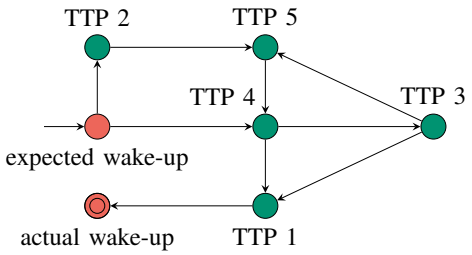

Figure 5. State model of a semi-Markov chain for cyclictest. The model comprises states for each tracing events. Two states (red) represent the expected (initial state) actual wake-up (absorbing state) within the cyclictest application. The remaining states (green) correspond to the manually inserted timed tracepoint in the kernel's interrupts handler and scheduler.

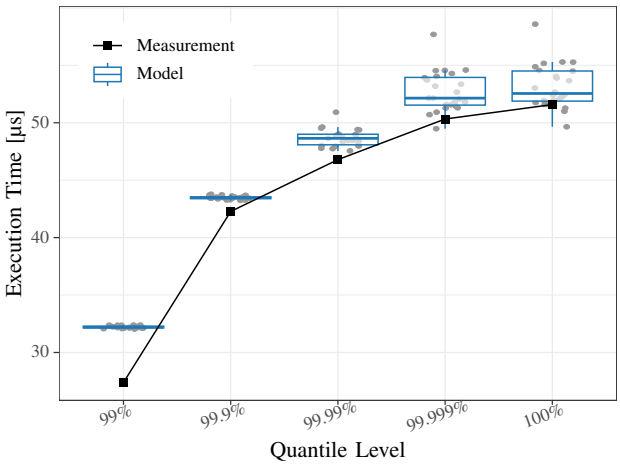

Figure 6. Comparison of quantiles of execution times from the measurement (black squares) and model sampling simulation (blue), aggregated over 24 independently built semi-Markov chains. The connecting line is included only to guide the eye and does not represent interpolation or progression.

For each SMC, we conduct 10 independent Monte Carlo simulations to propagate timing uncertainty and sample the system's latency distribution. Each simulation dataset comprises 10 000 absorption runs—complete state sequences from initial to terminal states—capturing the variability in both path selection and transition timing. This sample size aims for statistical stability in tail estimation while remaining computationally tractable. We analyse each resulting distribution regarding various quantiles and average its maximal value over the 10 simulation runs.

Figure 6 compares the quantiles obtained from empirical measurements (red) with those derived from the model simulation data (blue). The mean of the models' WCET estimation is 53.14 µs, which overestimates the empiric WCET by about 3 %. The quantile values averaged over the 24 generated SMCs

exhibit overestimation of $4.7\,\%$ for the $99.999\,\%$ quantile, $4.0\,\%$ for the $99.99\,\%$, and $2.9\,\%$ for $99.9\,\%$ quantile.

The results indicate a strong correspondence between the model-derived quantiles and the values observed in the original dataset, particularly up to the $99.9\,\%$ quantile. Although the variability increases for the $99.99\,\%$ quantile and the estimated worst-case latency—as reflected by the box plots—the mean and median values across simulations remain very close to those measured. These findings suggest that our approach reliably captures the latency distribution over a wide range of quantiles, while the increased uncertainty in the tail reflects the inherent variability in rare, extreme events.

C. Model Robustness with Partial Datasets

To assess the impact of limited measurement data on model accuracy, we generated partial datasets by selecting only the first x seconds (*i.e.*, $x \times 1000$ runs) from the full dataset. For each subset, we follow the same analysis procedure: independently build 24 SMCs, generate 10 sampling datasets with $10\,000$ simulation runs each.

Figure 7 compares the model predictions (black box plots) with the corresponding measured quantile values from the partial datasets (golden dots) and with the quantiles computed from the full dataset (blue line). The figure clearly shows that, even with a substantially reduced dataset, the SMC-based predictions converge toward the reference values from the full dataset. Notably, even when we reduce the measurement duration to only a few seconds, the model's predictions for extreme quantiles (*e.g.*, the $99.99\,\%$ quantile) and the WCET remain stable. For instance, when we use a 2 second subset—representing only $0.3\,\%$ of the full $5\,\mathrm{min}$ dataset—the empirical WCET in this subset is $45.56\,\mu\mathrm{s}$, which is lower than the $51.16\,\mu\mathrm{s}$ present in the complete dataset. Despite the absence of the latter in the subset, the averaged prediction from the models approximates the full-dataset WCET, yielding $51.16\,\mu\mathrm{s}$.

The impact of data reduction is more apparent at lower quantiles. Convergence of the $95\,\%$ quantile emerges only after approximately 6 seconds of measurement. In contrast, the predictions for the $99.99\,\%$ quantile and the WCET stabilise after 2 seconds and undergo only minor variance with longer measurement durations. On average, the models slightly overestimate the quantile values relative to both the full dataset and the subsets, except for the $99\,\%$ quantile. We observe a sudden increase in the $99.99\,\%$ quantile between 7 and 8 seconds, which likely reflects a higher incidence of elevated latency values in the smaller datasets. As more data becomes available, the quantile estimate gradually converges towards the value derived from the full dataset.

VII. Discussion

The evaluation demonstrates that our method effectively models the timing behaviour of cyclictest latency via SMCs. The event-centric approach integrates user space and kernel level data, ensuring that both frequent and rare extreme behaviours receive accurate representation. The complexity of SMC models scales with the number of distinct events and observed transitions rather than with the number of execution paths. This scaling avoids the exponential complexity that plagues many static analysis techniques [1]. The method provides significant advantage for complex, real-world systems, as it requires only a log of events with precise timestamps.

The robustness analysis suggests that our approach effectively generalises from limited data. Even when we use a few seconds of measurement rather than minutes, the SMC models extrapolate and predict extreme execution times despite the absence of those extremes in the measurement window. This reduction in required measurement time and resources proves especially beneficial when such resources are constrained. Rapid feedback on system performance accelerates the development cycle. Furthermore, the minor conservative bias—where the model tends to overestimate lower quantiles slightly—can serves as an advantage and ensures that predicted worst-case scenarios remain on the safe side.

We observe a rise in the $99.99\,\%$ quantile between 7 and 8 seconds, which indicates sensitivity in the estimation of extreme events. This observation highlights an area for further refinement. Future work may enhance the model's robustness by incorporating adaptive weighting mechanisms or context-sensitive constraints to more accurately capture such transitions. Overall, these findings reinforce the value of our hybrid method, which provides dependable runtime predictions and offers runtime insights for further system optimisation.

We applied our method to an alternative test operating system using the same approach. We instrumented the kernel and implemented an application analogous to `cyclictest`. Results agree with those obtained for the original system.

Our framework provides an intuitive and interpretable representation of system latency by modelling execution time as time-to-absorption of the SMC. It aids developers to identify key characteristics—such as waiting for I/O or interrupt handling—that contribute most significantly to performance variability.

The reliance on GMMs to model transition durations risks overfitting, particularly in scenarios with low variability. We also encountered logically invalid state sequences in our experiments. Incorporating context-sensitive constraints could prevent infeasible state transitions and enhance model validity. These issues are not unique to our approach, however; other measurement-based timing analysis methods also suffer from overfitting and incomplete representation of relevant factors.

The effectiveness of our method depends on the quality and granularity of both the user-provided model (through instrumentation) and the measurement data. The model's predictive accuracy relies on matching its complexity with that of the underlying system. Overly simplistic models may omit critical execution paths, while overly complex models risk overfitting the available data. A limitation to our current implementation is the assumption of time-homogeneity. We also simplify by assuming that transition durations depend solely on the source and target states, thereby neglecting possible long-term dependencies and context-sensitive effects, such as whether preemption is disabled.

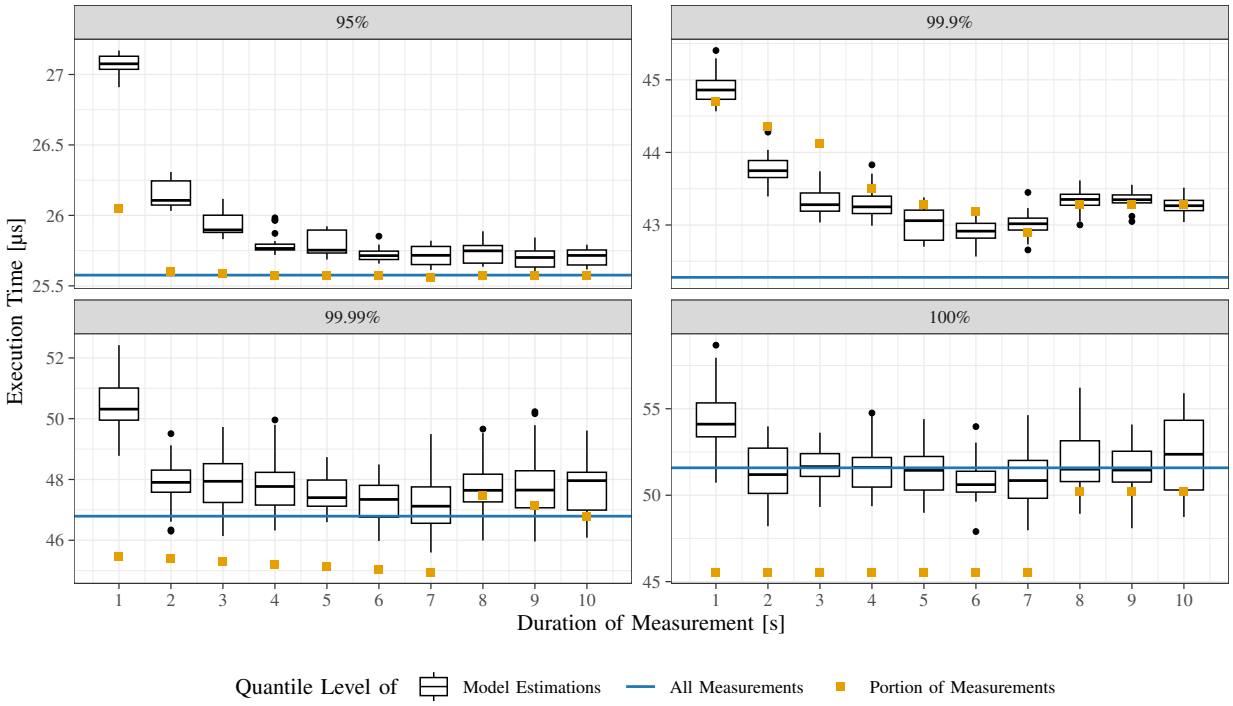

Figure 7. Convergence of quantile estimates with reduced measurement duration. Quantile estimates are derived from simulation data generated using subsets of the full dataset. For each measurement duration (first x seconds, *i.e.*, $x \times 10000$ runs), 24 SMC models are generated. Each of the models is sampled ten times with 10 000 runs. The models' prediction is the mean over quantiles calculated for each of the 10 sampling runs. Black box plots show the quantile predictions from SMC models, golden squares represent measured quantiles, and the blue line represents the quantile computed from the full dataset.

Although our demonstration confirms the model's generalisation capability, challenges remain. Missing transitions yield an incomplete state model that may reduce accuracy [44]. The model's simplicity, however, permits mitigation. We can address missing transitions and adjust assumptions regarding the distribution of transition durations between events manually. In such cases, we readily adapt the model by modifying its probabilistic parameters, such as the transition probability matrix P or the holding-time kernel Q.

VIII. Conclusion

We presented a semiformal hybrid method for runtime analysis that integrates measurement-based observations with model-based techniques. We employ semi-Markov chains to model execution times (latency between critical system events). Our approach balances realism and statistical precision, while preserving simplicity. It operates at a high level of abstraction, and primarily requires a choice of statistical distribution for transition durations. Our method requires only minor modifications to the target system, such as enabling and configuring lightweight tracepoints. As demonstrated in our evaluation, the approach integrates runtime interference within and across different domains, while imposing minimal overhead without compromising accuracy.

Our SMC models capture the structural characteristics of task execution through inter-event latencies. The experimental evaluation with cyclictest shows that the approach overestimates the empirical WCET by only about 3 % on average. This level of precision persists even when the measurement duration is reduced from multiple minutes to only a few seconds. Although longer measurements are typically needed to capture worst-case events directly, our framework predicts WCET from a limited dataset. This offers developers rapid insights into system timing behaviour, enabling quick feedback on the impact of code modifications and accelerating development.

In summary, our approach bridges rigorous modelling and pragmatic measurement-driven validation. It combines empirical data with stochastic models, and is applicable to rapid prototyping and in-depth performance evaluation.

We plan to extend the framework to support online analysis and integrate it with established real-time monitoring tools. This will enable real-time verification, adaptive instrumentation, and broader applicability in dynamic and industrial settings.

On the modelling side, we aim to incorporate context-sensitive factors (*e.g.*, temperature, voltage, preemption state) to improve predictive accuracy and responsiveness to system conditions. We will also explore richer sojourn time distributions to better capture tail behaviour and rare-event dynamics.

Finally, we plan to apply the method to larger and more complex systems, including multicore and distributed real-time platforms, to evaluate scalability and demonstrate its utility in realistic deployment scenarios.

References

[1] R. I. Davis and L. Cucu-Grosjean, "A survey of probabilistic timing analysis techniques for real-time systems," *Leibniz Transactions on Embedded Systems*, 2019. DOI: 10.4230/LITES-V006-I001-A003.

[2] M. Becker, R. Metta, R. Venkatesh, and S. Chakraborty, "Scalable and precise estimation and debugging of the worst-case execution time for analysis-friendly processors: A comeback of model checking," *International Journal on Software Tools for Technology Transfer*, 2019. DOI: 10.1007/s10009-018-0497-2.

[3] F. Meng and X. Su, "Reducing WCET overestimations by correcting errors in loop bound constraints," *Energies*, 2017. DOI: 10.3390/en10122113.

[4] P. Graydon and I. Bate, "Realistic safety cases for the timing of systems," *The Computer Journal*, 2014. DOI: 10.1093/comjnl/bxt027.

[5] K. Wintersperger, H. Safi, and W. Mauerer, "Qpu-system co-design for quantum hpc accelerators," in *Architecture of Computing Systems*. Springer International Publishing, 2022. DOI: 10.1007/978-3-031-21867-5_7.

[6] R. Ramsauer, S. Huber, K. Schwarz, J. Kiszka, and W. Mauerer, "Static hardware partitioning on risc-v: Shortcomings, limitations, and prospects," in *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*, 2022. DOI: 10.1109/WF-IoT54382.2022.10152063.

[7] W. Mauerer and M. C. Jaeger, "Open source engineering processes / open source-entwicklungsprozesse," *itit*, 2013. DOI: 10.1515/itit.2013.1008.

[8] S. Jiménez Gil, I. Bate, G. Lima, L. Santinelli, A. Gogonel, and L. Cucu-Grosjean, "Open challenges for probabilistic measurement-based worst-case execution time," *IEEE Embedded Systems Letters*, 2017. DOI: 10.1109/LES.2017.2712858.

[9] L. Santinelli, F. Guet, and J. Morio, "Revising measurement-based probabilistic timing analysis," in *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Pittsburg, PA, USA, 2017. DOI: 10.1109/RTAS.2017.16.

[10] J. Abella, C. Hernandez, E. Quiñones, *et al.*, "WCET analysis methods: Pitfalls and challenges on their trustworthiness," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2015. DOI: 10.1109/SIES.2015.7185039.

[11] G. Bernat, A. Colin, and S. Petters, "WCET analysis of probabilistic hard real-time systems," in *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, 2002. DOI: 10.1109/REAL.2002.1181582.

[12] C. Daws and S. Tripakis, "Model checking of real-time reachability properties using abstractions," in *Tools and Algorithms for the Construction and Analysis of Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. DOI: 10.1007/bfb0054180.

[13] S. Jeon, K. Cho, C. G. Kang, J. Lee, H. Oh, and J. Kang, "Quantum probabilistic model checking for time-bounded properties," *Artifact for "Quantum Probabilistic Model Checking for Time-Bounded Properties"*, OOPSLA2 2024. DOI: 10.1145/3689731.

[14] A. Betts, N. Merriam, and G. Bernat, "Hybrid measurement-based WCET analysis at the source level using object-level traces," in *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. DOI: 10.4230/OASIcs.WCET.2010.54.

[15] S. Edgar and A. Burns, "Statistical analysis of WCET for scheduling," in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, 2001. DOI: 10.1109/REAL.2001.990614.

[16] C. Maiza, H. Rihani, J. Rivas, J. Goossens, S. Altmeyer, and R. I. Davis, "A survey of timing verification techniques for multi-core real-time systems," *ACM Computing Surveys (CSUR)*, 2019. DOI: 10.1145/3323212.

[17] F. Brandenburg, "A comparison of scheduling latency in linux, preempt rt, and litmusrt," *OSPERT 2013*, 2013.

[18] S. D. Sharma and M. Dagenais, "Hardware-assisted instruction profiling and latency detection," *The Journal of Engineering*, 2016. DOI: 10.1049/joe.2016.0127.

[19] Y. Bao, J. Zhang, Y. Zhu, *et al.*, *HMTT: A hybrid hardware/software tracing system for bridging memory trace's semantic gap*, 2011. DOI: 10.48550/arXiv.1106.2568.

[20] D. B. De Oliveira and R. S. De Oliveira, "Timing analysis of the PREEMPT RT linux kernel," *Software: Practice and Experience*, 2016. DOI: 10.1002/spe.2333.

[21] R. Hofmann, R. Klar, B. Mohr, A. Quick, and M. Siegle, "Distributed performance monitoring: Methods, tools, and applications," *IEEE Transactions on Parallel and Distributed Systems*, 1994. DOI: 10.1109/71.285605.

[22] A. Vergé, N. Ezzati-Jivan, and M. R. Dagenais, "Hardware-assisted software event tracing," *Concurrency and Computation: Practice and Experience*, 2017. DOI: 10.1002/cpe.4069.

[23] A. M. K. Cheng, *Real-time systems: scheduling, analysis, and verification*. Hoboken, NJ: Wiley-Interscience, 2002.

[24] P.-C. G. Vassiliou and A. C. Georgiou, "Markov and semi-markov chains, processes, systems, and emerging related fields," *Mathematics*, 2021. DOI: 10.3390/math9192490.

[25] V. S. Barbu, *Semi-Markov Chains and Hidden Semi-Markov Models Toward Applications: Their Use in Reliability and DNA Analysis* (Lecture Notes in Statistics). New York, NY: Springer New York, 2008. DOI: 10.1007/978-0-387-73173-5.

[26] G. D'Amico, F. Petroni, and F. Prattico, "Reliability measures of second-order semi-markov chain applied to wind energy production," *Journal of Renewable Energy*, 2013. DOI: 10.1155/2013/368940.

[27] F. Cartella, J. Lemeire, L. Dimiccoli, and H. Sahli, "Hidden semi-markov models for predictive maintenance,"

Mathematical Problems in Engineering, 2015. DOI: 10.1155/2015/278120.

[28] S. Ruiz-Suarez, V. Leos-Barajas, and J. M. Morales, "Hidden markov and semi-markov models when and why are these models useful for classifying states in time series data?" *Journal of Agricultural, Biological and Environmental Statistics*, 2022. DOI: 10.1007/s13253-021-00483-x.

[29] S.-Z. Yu, "Hidden semi-markov models," *Artificial Intelligence*, 2010. DOI: 10.1016/j.artint.2009.11.011.

[30] S. Bozhko, F. Marković, G. von der Brüggen, and B. B. Brandenburg, "What really is pWCET? a rigorous axiomatic proposal," in *2023 IEEE Real-Time Systems Symposium (RTSS)*, 2023. DOI: 10.1109/RTSS59052.2023.00012.

[31] G. Chennupati, N. Santhi, P. Romero, and S. Eidenbenz, "Machine learning–enabled scalable performance prediction of scientific codes," *ACM Transactions on Modeling and Computer Simulation*, 2021. DOI: 10.1145/3450264.

[32] J. P. Thoma, J. Feldtkeller, M. Krausz, T. Güneysu, and D. J. Bernstein, "BasicBlocker: ISA redesign to make spectre-immune CPUs faster," *24th International Symposium on Research in Attacks, Intrusions and Defenses*, 2021. DOI: 10.1145/3471621.3471857.

[33] R. Ragel, J. A. Ambrose, and S. Parameswaran, "SecureD: A secure dual core embedded processor," 2015.

[34] H.-Y. Lin and R. Tsay, "A precise program phase identification method based on frequency domain analysis," 2021.

[35] A. N. Amalou, I. Puaut, and G. Muller, "WE-HML: Hybrid WCET estimation using machine learning for architectures with caches," in *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2021. DOI: 10.1109/RTCSA52859.2021.00011.

[36] B. Cai, X. Kong, Y. Liu, *et al.*, "Application of bayesian networks in reliability evaluation," *IEEE Transactions on Industrial Informatics*, 2019. DOI: 10.1109/TII.2018.2858281.

[37] Z. Zhang, "Bayesian networks with examples in RBayesian networks with examples in r," *Journal of Quality Technology*, 2023. DOI: 10.1080/00224065.2023.2171320.

[38] B. G. Marcot and T. D. Penman, "Advances in bayesian network modelling: Integration of modelling technologies," *Environmental Modelling & Software*, 2019. DOI: 10.1016/j.envsoft.2018.09.016.

[39] J. L. Puga, M. Krzywinski, and N. Altman, "Bayesian networks," *Nature Methods*, 2015. DOI: 10.1038/nmeth.3550.

[40] R. Alur, "Timed automata," in *Computer Aided Verification*, N. Halbwachs and D. Peled, Eds., vol. 1633, Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 8–22. DOI: 10.1007/3-540-48683-6_3.

[41] D. B. de Oliveira, R. S. d. Oliveira, and T. Cucinotta, "A thread synchronization model for the PREEMPT_rt linux kernel," *Journal of Systems Architecture*, 2020. DOI: 10.1016/j.sysarc.2020.101729.

[42] D. B. de Oliveira, T. Cucinotta, and R. S. de Oliveira, "Efficient formal verification for the linux kernel," in *Software Engineering and Formal Methods*, P. C. Ölveczky and G. Salaün, Eds., Springer International Publishing, 2019.

[43] D. B. de Oliveira, "Automata-based formal analysis and verification of the real-time linux kernel," Ph.D. dissertation, Universidade Federal de Santa Catarina and Scuola Superiore Sant'Anna, 2020.

[44] B. Lesage, D. Griffin, F. Soboczenski, I. Bate, and R. I. Davis, "A framework for the evaluation of measurement-based timing analyses," in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, Lille France: ACM, 2015. DOI: 10.1145/2834848.2834858.

[45] A. Friebe, F. Marković, A. V. Papadopoulos, and T. Nolte, "Continuous-emission markov models for real-time applications: Bounding deadline miss probabilities," in *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, San Antonio, TX, USA: IEEE, 2023. DOI: 10.1109/RTAS58335.2023.00009.

[46] R Core Team, *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing, 2024.

[47] H. Wickham, M. Averick, J. Bryan, *et al.*, "Welcome to the tidyverse," *Journal of Open Source Software*, 2019. DOI: 10.21105/joss.01686.

[48] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016.

[49] T. Barrett, M. Dowle, A. Srinivasan, *et al.*, *data.table: Extension of 'data.frame'*. 2024.