Information and Software Technology 196 (2026) 108172

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Runtime verification of cause–effect latency in black-box systems

Yong-Jun Shin

Electronics and Telecommunications Research Institute (ETRI), Republic of Korea

ARTICLE INFO

Keywords:
Runtime verification
Cause–effect latency
End-to-end latency
Cause–effect chain
Black-box systems

ABSTRACT

Context: Safety-critical systems such as autonomous driving rely on cause–effect chains of asynchronous tasks, where end-to-end latency must meet strict timing constraints. While design-time analysis provides guarantees, runtime verification is required to handle runtime uncertainties. However, existing runtime methods often rely on intrusive instrumentation or detailed white-box access, which are rarely available in practice.

Objective: This study aims to enable runtime verification of cause–effect latency in black-box systems, where only task output events (write-events) are observable.

Methods: We formally extend the classic cause–effect latency verification problem to the black-box setting and propose two lightweight algorithms: (1) a job-chain estimation algorithm that safely over-estimates latency based on limited observations, and (2) a statistical verification algorithm that produces verdicts under user-specified confidence levels, avoiding false positives by design.

Results: We theoretically prove the over-estimation error bounds and empirically validate them through controlled experiments. Results show that estimation errors inherent to limited observability diminish when chain utilization is high, the verification algorithm remains trustworthy with extremely low false positives, while exhibiting limited conservative false negatives as a trade-off, and both algorithms incur negligible runtime cost. An industrial case study on an autonomous driving system further confirms the practicality of the proposed approach, successfully verifying 39 chains under black-box constraints with minimal engineering effort.

Conclusion: The findings demonstrate that black-box runtime verification of cause–effect latency is both feasible and effective, providing a lightweight and practical foundation for verifying timing requirements in complex, safety-critical systems.

1. Introduction

Safety-critical systems such as autonomous driving and robotics consist of numerous asynchronous tasks that together accomplish complex missions. Their data dependencies form *cause–effect chains*; for example, sensor data may be processed through a sequence of computations and ultimately drive an actuator or control decision. A key safety requirement is that the end-to-end latency of these chains, or *cause–effect latency*, must remain below a specified threshold. While design-time verification provides guarantees, runtime verification is essential to account for uncertainties such as workload fluctuations and environmental variations.

Analyzing cause–effect latency requires knowledge of three key aspects: (1) the time when a task reads its input, (2) the time when it writes its output from the input, and (3) the data-dependency relations that connect tasks. Capturing this information typically requires thorough instrumentation and white-box access to the system implementation. In practice, however, such access is often not readily available. Black-box modules, vendor components, and distributed execution across network boundaries make full observability impractical. In industrial settings, tasks may be developed by multiple partners, leaving verification teams with only limited visibility into externally observable events (as illustrated in Section 2).

To address this challenge, we propose a *domain-agnostic runtime solution* for estimating and verifying cause–effect latencies in black-box systems. We assume that only task output timestamps (i.e., write-events) are observable at runtime, while other details such as read-events or internal states remain hidden. Our solution consists of two algorithms: (1) a *job-chain estimation algorithm* that reconstructs worst-case job chains from observable events to provide safe over-estimates, and (2) a *statistical verification algorithm* that estimates the $p\%$ upper tolerance limit of latency and issues verdicts against a threshold δ, avoiding false positives under a user-specified confidence level γ while exhibiting limited conservative false negatives as a trade-off.

The contributions of this paper are:

E-mail address: yjshin@etri.re.kr.

https://doi.org/10.1016/j.infsof.2026.108172
Received 10 October 2025; Received in revised form 24 April 2026; Accepted 27 April 2026
Available online 28 April 2026

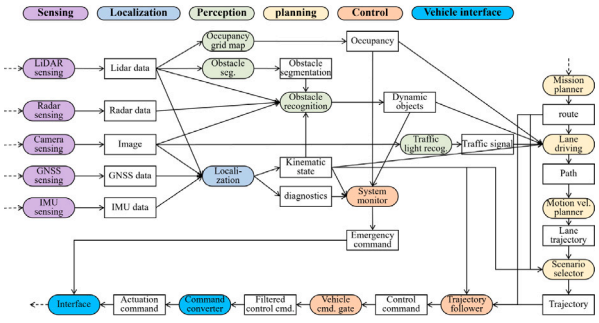


Fig. 1. Cause–effect chains in the autonomous driving system.

1. Formalization of the runtime verification problem for cause–effect latency in black-box systems, extending the classic definition.
2. Proposal of two lightweight algorithms: job-chain estimation and statistical verification.
3. Formal proofs of estimation error bounds, showing safe overestimation of latency.
4. Empirical evaluation of estimation error, verification trustworthiness, and runtime cost.

Finally, we briefly report our industrial experience applying the proposed methods, demonstrating their applicability under real black-box constraints.

The remainder of this paper is organized as follows. Section 2 introduces a motivating example, Section 3 defines the problem, and Section 4 outlines our approach. Sections 5 and 6 detail the estimation and verification algorithms, Section 7 presents the evaluation of the algorithms, and Section 8 discusses an industrial case study. Finally, Section 9 reviews related work and Section 10 concludes.

2. Motivating example

To illustrate the challenges of verifying cause–effect latencies, we present our experience with an autonomous driving system developed with industrial partners. The system integrates asynchronous tasks from diverse sensing modalities (LiDAR, radar, camera, GNSS, IMU), forming a complex network of cause–effect chains from inputs to outputs (Fig. 1).

Our mission was to verify whether the end-to-end latency from input to output satisfied the safety requirement, defined as an upper threshold (e.g., 0.5 s). However, the system operated under strict black-box constraints: individual tasks were developed by different partners, full access to the source code was not available to us, some tasks were executed remotely across network boundaries, and the complexity of the overall integration made it impractical to deploy complete instrumentation for end-to-end measurement. At least the output events of each task (i.e., write-events) were observable, but other events and internal states remained hidden.

This setting reflects a common industrial challenge: verifying latency in black-box systems where direct measurement is infeasible and only partial runtime data are available. Our algorithms address this by enabling safe, lightweight analysis of cause–effect latencies without full internal access.

3. Problem definitions

Motivated by the above, we base our study on the cause–effect chain latency analysis problem. We first review the classic formulation under white-box assumptions, then extend it to the runtime black-box setting where only partial observations are available, requiring new estimation and verification methods.

3.1. Classic problem definition

The cause–effect latency problem is well characterized by [1]; we refer to it here as the *classic problem*, summarizing only the parts relevant to our extension.

Consider a system with a set of tasks $\mathbb{T}$ and a sequence $\tau_1, \dots, \tau_n$ ($n \le |\mathbb{T}|$), where each task depends on the output of its predecessor. Such a sequence defines a *cause–effect chain*:

$$E = (\tau_1 \to \cdots \to \tau_n), \quad \text{where } \tau_i \in \mathbb{T}. \tag{1}$$

The tasks in the cause–effect chain E execute under a given scheduling mechanism of the system. Each execution instance of a task is referred to as a *job*, and the cause–effect chain is instantiated as job chains at runtime. A *job chain* c is defined as:

$$c = (J_1, \dots, J_n), \tag{2}$$

where each J_i is a job of task τ_i.

For each job J, let $re(J)$ and $we(J)$ denote its read-event and write-event times, respectively, where $re(J) \le we(J)$. Since each job J_{i+1} of task τ_{i+1} reads the output produced by job J_i of task τ_i, the following causality condition holds:

$$we(J_i) \le re(J_{i+1}). \tag{3}$$

We measure the cause–effect latency of the chain E using the *Maximum Data Age* (MDA) metric, which captures the worst-case data freshness along the chain. To express the latency in terms of MDA, we extend a job chain with two external time instants z and z', yielding an *immediate backward augmented job chain* [1,2]:

$$\overline{ac} = (z, J_1, \dots, J_n, z'). \tag{4}$$

Here, each job J_i for $i \in \{1, \dots, n-1\}$ is the latest job of task τ_i satisfying the causality condition in Eq (3). The time $z = re(J_1)$ is the read-event time of the first job J_1, and z' is a time pivot such that J_n is the latest job with $we(J_n) \le z'$. To ensure that such augmented job chains exist, we consider only time instants z' after a *warm-up*, which is the period during which the system reaches a steady state and relevant data have fully propagated [1].

The length of the augmented job chain is defined as:

$$l(\overline{ac}) = z' - z. \tag{5}$$

The *cause–effect latency* (i.e., end-to-end latency) of the chain E using the MDA metric is defined as:

$$\mathrm{Lat}(E) = \sup \left\{ l(\overline{ac}_{z'}) \;\middle|\; z' > t_{\mathrm{warmup}} \right\}, \tag{6}$$

i.e., the supremum of the lengths of all immediate backward augmented job chains considered after the end of the warm-up period t_{warmup}.

The latency is verified against a user-specified threshold δ (e.g., timing requirements) as follows:

$$\mathrm{Lat}(E) \le \delta. \tag{7}$$

3.2. Necessity of problem extension

To determine $\mathrm{Lat}(E)$, the classic problem assumes that the true immediate backward augmented job chain $\overline{ac}$ is known. Specifically, the following information is required to capture the true chain $\overline{ac}$ as shown in Fig. 2:

- The precise start (i.e., read-event $re(J)$) and end (i.e., write-event $we(J)$) times of each job.
- The data dependencies between specific inputs and outputs of individual jobs.

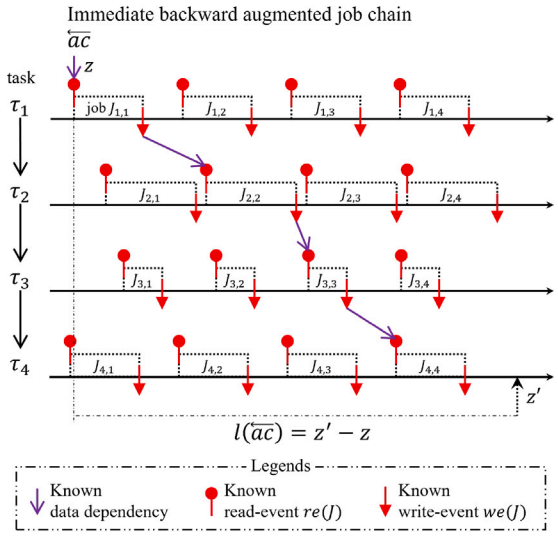


Fig. 2. Information Required for the classic problem definition.

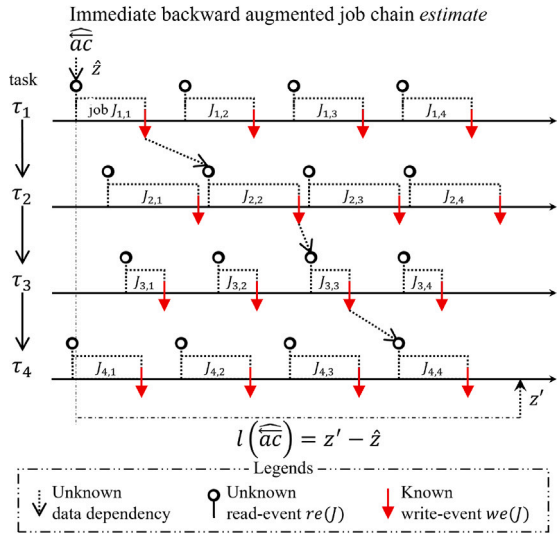


Fig. 3. Observability scope assumed in this paper for the extended problem definition.

However, these requirements are often unrealistic in practice. Black-box or third-party modules run asynchronously, making code instrumentation for data flow or job timing infeasible. In such cases, the true data dependencies and lengths of jobs are not directly measurable and must be inferred through indirect observation or runtime logging. In addition, the execution time of the job is often nondeterministic, so the cause–effect latency $\mathrm{Lat}(E)$ is no longer fixed at runtime. It varies due to uncertainties in operation, such as computation load, network delays, or power constraints. These render static analysis insufficient and necessitate runtime verification for safe estimation.

3.3. Extended problem definition

To address the realistic constraints discussed above, we extend the classic problem.

We first define the estimated job chains of a given cause–effect chain E as:

$$\hat{c} = (\hat{J}_1, \ldots, \hat{J}_n), \tag{8}$$

and the corresponding estimated immediate backward augmented job chain as:

$$\widehat{\overleftarrow{ac}} = (\hat{z}, \hat{J}_1, \ldots, \hat{J}_n, z'). \tag{9}$$

Here, $\hat{c}$ and $\widehat{\overleftarrow{ac}}$ denote the estimations of the actual but unknown job chain c and immediate backward augmented job chain $\overleftarrow{ac}$ of E, respectively, and $\hat{J}_i$ are the constituent jobs of the estimate. The value $\hat{z} = re(\hat{J}_1)$ denotes the read-event time of the first estimated job, and $z' \geq we(\hat{J}_n)$ is the given time pivot, as in the original definition of $\overleftarrow{ac}$. It is important to emphasize that z' is given and not estimated, whereas the other elements of $\widehat{\overleftarrow{ac}}$ are inferred backward from z'.

We also extend the cause–effect latency of E using the MDA metric under the running system state at a specific time point t as $\mathrm{Lat}_t(E)$, and define the function of *estimated cause–effect latency* of E at the time t based on the estimated immediate backward augmented job chains $\widehat{\overleftarrow{ac}}$, as:

$$\widehat{\mathrm{Lat}}_t(E) = \sup \left\{ l(\widehat{\overleftarrow{ac}}_{z'}) \;\middle|\; t \geq z' > t_{\text{warmup}} \right\}. \tag{10}$$

$$\widehat{\mathrm{Lat}}_t(E) \geq \mathrm{Lat}_t(E). \tag{11}$$

The goal is to find an estimate $\widehat{\mathrm{Lat}}_t(E)$ that closely approximates the true latency while safely over-estimating it, minimizing the difference $\widehat{\mathrm{Lat}}_t(E) - \mathrm{Lat}_t(E)$. A verification use case based on this definition is to use the estimated latency to verify whether $\mathrm{Lat}_t(E) \leq \delta$ holds, by checking:

$$\widehat{\mathrm{Lat}}_t(E) \leq \delta. \tag{12}$$

This extended problem facilitates the practical verification of cause–effect latency of running black-box systems. The central challenge lies in deriving a sound estimate of immediate backward augmented job chain $\widehat{\overleftarrow{ac}}$ and $\widehat{\mathrm{Lat}}_t(E)$ of the given E using only the limited information that is readily available at runtime. Accordingly, the system's observable scope and models must be further specified, as it can vary depending on the system under verification.

3.4. System models of this paper for the extended problem

This section defines the system models that we focus in this paper for proposing a practical solution of the extended problem.

Fig. 3 first illustrates the *limited observability scope* of the target system. We assume that the write-events of jobs in the cause–effect chain of interest are only observable at runtime. In contrast to the information required for the classic problem shown in Fig. 2, the true data dependencies and read-events of jobs may remain unknown.

This level of observability is practical in many real-world systems. By monitoring the intra-task communication layer, publication signals of task outputs can still be observed, even when some tasks are implemented as or encapsulate black-box modules. For example, in robotics applications developed on the Robot Operating System (ROS), publication events can be captured via Data Distribution Service (DDS) middleware or an observer node. In contrast, monitoring data dependencies and read-events is harder, especially for black-box ROS nodes without dedicated probes or logging support.

Furthermore, we adopt the semantics of *periodic tasks* and their *implicit communication* [1] to formally model the cause–effect chain of the system under verification. Fig. 4 illustrates these semantics. A periodic task τ_i is defined as $\tau_i = (C_{\tau_i}, T_{\tau_i}, \phi_{\tau_i})$, where C_{τ_i} is the worst-case execution time of a job of the task, T_{τ_i} is the period between consecutive job releases, and ϕ_{τ_i} is the release offset of the first job. That is, the task releases a job every T_{τ_i} time units starting from ϕ_{τ_i}, and each released job requires at most C_{τ_i} units of execution time within its release interval under the given scheduling mechanism (e.g., preemptive or non-preemptive). The Job J's read-event $re(J)$ and write-event $we(J)$ correspond to the beginning and end of its execution, respectively, thereby realizing the implicit communication between

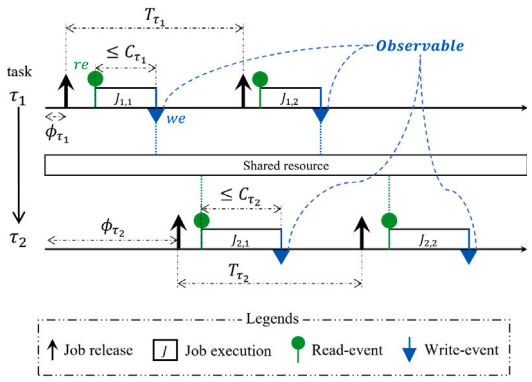


Fig. 4. Task communication semantic assumed in this paper for the extended problem definition.

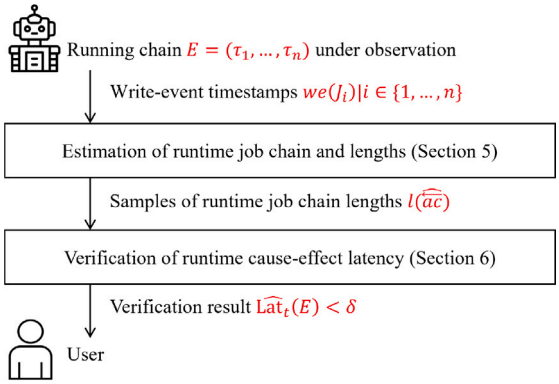


Fig. 5. Overall process for runtime cause–effect latency verification in a black-box system.

dependent tasks. Accordingly, the read-event $re(J_{i,j})$ and write-event $we(J_{i,j})$ of the jth job of task τ_i satisfy the following timing constraint:

$$\phi_{\tau_i} + jT_{\tau_i} \le re(J_{i,j}) \le we(J_{i,j}) \le \phi_{\tau_i} + (j+1)T_{\tau_i}. \tag{13}$$

This model is widely adopted in both formal analysis [3–5] and industrial applications [6–8], for example in autonomous driving pipelines composed of sensing, localization, perception, planning, and control modules. Accordingly, our proposed solution for the extended cause–effect latency verification problem assumes that the cause–effect chain E under analysis follows the above semantic model. It provides the theoretical basis for deriving the latency estimate, and underpins the ground truth used throughout this paper, including its use in the evaluation setup. While the complete timing parameters of the chain are not available at runtime, observable write-events form the basis of the estimation grounded in the model.

This section has defined the extended problem of runtime cause–effect chain latency verification in black-box systems and introduced the system models assumed in our proposed solution. The following sections present the practical solution to this problem.

4. Overall approach

We propose a practical solution for the runtime verification of cause–effect latency formally defined in Section 3. Recall that the problem has two core objectives:

- Estimate cause–effect chains from the limited observability of black-box systems.
- Verify at runtime whether the observed latency satisfies a user-specified threshold.

Our solution addresses the objectives with two core steps, as illustrated in Fig. 5: (1) job-chain estimation, and (2) runtime verification of cause–effect latency. We assume that task output events (write-events) in a chain $E = (\tau_1, \dots, \tau_n)$ are observable at runtime, as described in Section 3.4. From observed timestamps $\{we(J_i) \mid i \in \{1, \dots, n\}\}$, the first step reconstructs worst-case job chains and derives samples of job-chain lengths $l(\widehat{c})$. The second step statistically analyzes these samples to estimate the cause–effect latency and verify whether it satisfies the threshold δ under a user-specified confidence γ. The result is a verdict on whether $\widehat{\text{Lat}}_t(E) < \delta$. Both steps run automatically and asynchronously, enabling continuous verification from the latest observations. The following sections detail these two steps.

5. Estimation of the job chain length

We propose an algorithm that estimates job chains at runtime based on the observed write-events of the given cause–effect chain. The following subsections present the estimation algorithm and provide a formal proof of the over-estimation bound.

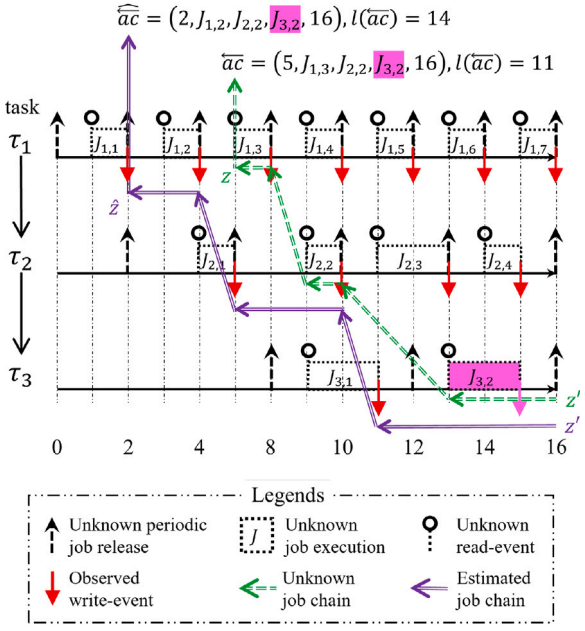


Fig. 6. An example of the job chain estimation.

5.1. Job chain estimation algorithm

Based on the definition of our runtime latency estimate $\widehat{\text{Lat}}_t(E)$ defined in Eq. (10), the first step is to collect instances of $\widehat{\overline{ac}}$ from write-events only. By tracing backward from the given end time pivot z' to its source z, this estimation runs online, ensuring verification reflects the most up-to-date chain length.

Fig. 6 illustrates the concept of our job-chain estimation algorithm. We consider a cause–effect chain $E = (\tau_1 \to \cdots \to \tau_3)$, where the write-events of each task — marked as red downward arrows — are observable at runtime. All other details, such as job releases and read-events, remain unobservable but follow the task semantics defined in Section 3.4.

After a warm-up period, a backward chain $\overline{ac}$ can be defined from the current time. In the example, the current time is $z' = 16$, and the second job of the third task, $J_{3,2}$, is taken as the target job. According to the definition, the corresponding $\overline{ac}$ is $(z = 5, J_{1,3}, J_{2,2}, J_{3,2}, z' = 16)$, shown as the green dotted double-line arrow, with length $l(\overline{ac}) = 11$. Although this $\overline{ac}$ is the actual job chain, it remains unobservable due to limited visibility.

We therefore perform a *safe* over-estimation of the job chain, starting from the current time $z' = 16$. Such over-estimation is safe in the

Algorithm 1: Job chain length estimation

Precon.: Cause–effect chain $E = (\tau_1 \to \cdots \to \tau_n)$
Input : Write-event time-series dataset D
$| D := \{D_{\tau_1}, D_{\tau_2}, \ldots, D_{\tau_n}\}$
$| D_{\tau_i} := [we(J_{i,1}), \ldots, we(J_{i,k_i})]$,
Current time $z' \geq we(J_{n,k_n})$
Output : Estimated job chain length $\widehat{l}$
Init : Empty estimation of job chain $\widehat{\overline{ac}}$
$| \widehat{\overline{ac}} := (\hat{z}, \widehat{J}_1, \ldots, \widehat{J}_n, z')$
$| \widehat{J} := \langle t_{re}, t_{we} \rangle$,
Read-event lower-bound $re(\widehat{J}) := t_{re}$,
Dataset filter $D_\tau^{<x} := \{t \in D_\tau \mid t < x\}$

1 $m \leftarrow |D_{\tau_n}|$
2 $\widehat{J}_n \leftarrow \langle D_{\tau_n}[m-1], D_{\tau_n}[m] \rangle$
3 **foreach** $i \in [n-1, n-2, \ldots, 1]$ **do**
4 $\quad D'_{\tau_i} \leftarrow D_{\tau_i}^{<re(\widehat{J_{i+1}})}$
5 $\quad$ **if** $D'_{\tau_i} = \emptyset$ **then return** none
6 $\quad m \leftarrow |D'_{\tau_i}|$
7 $\quad \widehat{J}_i \leftarrow \langle D'_{\tau_i}[m-1], D'_{\tau_i}[m] \rangle$
8 $\quad$ **if** $i = 1$ **then**
9 $\quad\quad \hat{z} \leftarrow re(\widehat{J}_i)$
10 $\quad$ **end**
11 **end**
12 $\widehat{l} \leftarrow z' - \hat{z}$
13 **return** $\widehat{l}$

context of cause–effect latency verification — i.e., checking whether $\mathrm{Lat}(E) \leq \delta$ — because it avoids false positives. The estimated job chain $\widehat{\overline{ac}}$ is shown as the purple double-line arrow in Fig. 6. The estimation process proceeds as follows:

First, we find the most recent write-event of the target task, $we(J_{3,2}) = 15$, and estimate its read-event $re(J_{3,2})$. Although $re(J_{3,2})$ is unknown, the semantic rule (Eq. (13)) ensures that $we(J_{3,1}) < re(J_{3,2})$. Thus, we safely estimate the lower bound of $re(J_{3,2})$ as $we(J_{3,1}) = 11$.

Second, we estimate $\widehat{J}_2 = J_{2,x}$ based on $we(J_{3,1})$, the safe estimate of $re(J_{3,2})$. By the causality rule (Eq. (3)), we require $we(J_{2,x}) \leq re(J_{3,2})$. Since $we(J_{3,1})$ is a safe lower bound for $re(J_{3,2})$, we select the most recent write-event $we(J_{2,2}) = 10$ satisfying $we(J_{2,2}) < we(J_{3,1})$. Following the same rationale as in the first step, we estimate the lower bound of $re(J_{2,2})$ as $we(J_{2,1}) = 5$.

Third, we estimate $\widehat{J}_1 = J_{1,x}$ using the result of the previous step in the same manner. In this example, we obtain $\widehat{J}_1 = J_{1,2}$ with write-event $we(J_{1,2}) = 4$, and estimate the safe lower bound of its read-event $re(J_{1,2})$ as $we(J_{1,1}) = \hat{z} = 2$. As a result, the estimated length of the job chain is $l(\widehat{\overline{ac}}) = 14$. Although the estimated length $l(\widehat{\overline{ac}})$ does not exactly match the true length $l(\overline{ac})$, it provides a safe over-estimation of the chain length under limited observability.

Algorithm 1 presents our runtime job chain length estimation procedure for a given cause–effect chain. The **Precondition** specifies the analysis target as a cause–effect chain $E = (\tau_1 \to \cdots \to \tau_n)$, where each task τ_i produces an observable sequence of write-events during execution. The **Input** includes a write-event time-series dataset D, where each task τ_i has its own ordered list D_{τ_i} of observed write-event timestamps, indexed from 1 with earlier events appearing first. Each list may have a different number of entries, denoted by k_i. In addition, the current observation time z' is provided, which must be greater than the most recent write-event of the last task $we(J_{n,k_n})$. The **Output** is the estimated chain length $\widehat{l} = l(\widehat{\overline{ac}})$ of the estimated job chain derived from these observations.

The **Init** block prepares the necessary definitions and intermediate structures. The estimated job chain $\widehat{\overline{ac}}$ is initialized as an empty chain containing the estimated start time $\hat{z}$, a sequence of estimated jobs $\widehat{J}_1, \ldots, \widehat{J}_n$, and the given time pivot z'. Each estimated job $\widehat{J}$ is represented as a pair $\langle t_{re}, t_{we} \rangle$, where t_{re} denotes a conservative lower bound on the (unobservable) read-event time, and t_{we} is the exact (observable) write-event time. Specifically, an estimation $\widehat{J} = \langle t_{re}, t_{we} \rangle$ estimates J implying the following:

- The read-event time is expected to satisfy $t_{re} \leq re(J) < t_{we}$.
- The write-event time is expected at $we(J) = t_{we}$.

For $\widehat{J}$, we define its read-event lower-bound estimate as $re(\widehat{J}) := t_{re}$. Note that this notation overloads $re(\cdot)$: while $re(J)$ denotes the read-event of an actual but unknown job J, $re(\widehat{J})$ is used throughout this paper to represent an inferred lower-bound estimate of the read-event for the estimated job $\widehat{J}$. In addition, we define a filter operation $D_\tau^{<x}$ to extract the sublist of write-events from D_τ that occur strictly earlier than a given threshold x—used later to select candidate jobs when propagating backward. The filtered dataset preserves the original ordering, meaning earlier timestamps (i.e., lower indices starting from 1) always appear first.

The process begins by setting m to the size of the dataset D_{τ_n} of the last task τ_n (**Line 1**). It then identifies the most recent job $\widehat{J}_n$ of task τ_n from the dataset D_{τ_n}, constructing its estimate as $\langle D_{\tau_n}[m-1], D_{\tau_n}[m] \rangle$ (**Line 2**). Here, the last observed write-event provides the exact t_{we}, while the preceding event offers a conservative lower bound on the read-event.

Next, the algorithm iteratively estimates upstream jobs $\widehat{J}_i$ for each task τ_i ($i = n-1, \ldots, 1$) (**Lines 3–11**). At each iteration, the dataset D_{τ_i} is filtered to retain only those write-events that occur strictly earlier than the estimated read-event of the downstream job $\widehat{J_{i+1}}$ (**Line 4**). This step assumes no explicit inter-task transmission delay, a common simplification since such delays are typically negligible compared to computation time [9,10], making the resulting over-estimation still valid. If the filtered dataset D'_{τ_i} is empty, meaning that no preceding write-event is available to conservatively bound the read-event of the current job, the algorithm returns `none` (**Line 5**). This indicates that the job-chain length cannot yet be estimated from the available data. This situation typically occurs when the system has not yet reached the warm-up phase required to form valid immediate backward chains, or during the initial phase of runtime monitoring. The estimation then resumes once sufficient write-events have been accumulated to provide an up-to-date job-chain length estimate.

From the filtered dataset, the most recent two events are selected to construct $\widehat{J}_i = \langle D'_{\tau_i}[m-1], D'_{\tau_i}[m] \rangle$ (**Line 7**), ensuring causality is preserved while maintaining a conservative bound. When the first task ($i = 1$) is reached, the algorithm sets $\hat{z}$ to the lower-bound estimate of its read-event (**Line 9**), marking the beginning of the estimated chain.

After completing the loop, the estimated job chain length $\widehat{l}$ is computed as the difference $z' - \hat{z}$ (**Line 12**). This value $\widehat{l}$ provides a safe over-approximation of the runtime job-chain length based solely on observable write-events, ensuring conservative verification under limited observability. Finally, the algorithm returns the estimated length $\widehat{l}$ (**Line 13**).

The algorithm runs continuously, triggered each time a new write-event is added to the dataset D. At each step, it anchors at the most recent sink job at time z' and propagates backward to reconstruct the chain up to its inferred source. Each run yields an estimated length $\widehat{l}$ for the current chain instance, and these values accumulate into a sample set used for statistical verification.

5.2. Proof of the over-estimation error bound

Our algorithm uses only observable write-events to estimate the job chain length. To ensure safety under limited observability, the algorithm is designed to produce an over-estimate of the oracle chain length. This may introduce a discrepancy between the estimated and the oracle chain lengths. However, the resulting over-estimation error

is provably bounded under the semantics in Section 3.4, enabling users to take this bound into account. Below, we present the proof of the over-estimation property and the corresponding error bound.

Consider an actual job chain $\overline{ac} = (z, J_{1,j_1}, \ldots, J_{n,j_n}, z')$ with length $l(\overline{ac}) = z' - z$, and an estimated job chain $\widehat{\overline{ac}} = (\hat{z}, \hat{J}_1, \ldots, \hat{J}_n, z')$ with length $l(\widehat{\overline{ac}}) = z' - \hat{z}$, where the latter corresponds to the algorithm output denoted by $\hat{l}$. The over-estimation error Δ is defined as

$$\Delta = l(\widehat{\overline{ac}}) - l(\overline{ac}). \tag{14}$$

To bound Δ, we derive and compare theoretical ranges for $l(\overline{ac})$ and $l(\widehat{\overline{ac}})$ in terms of the given time pivot z' and the task period T specified by the system semantic in Section 3.4.

Lemma 1 (*Over-Estimation Property*). *For a given pivot time z', let $\overline{ac}$ be the oracle augmented job chain and $\widehat{\overline{ac}}$ the estimated chain returned by the algorithm. Then the estimated chain length is never smaller than the oracle chain length, i.e.,*

$$l(\widehat{\overline{ac}}) \geq l(\overline{ac}). \tag{15}$$

Proof. Let $l(\overline{ac}) = z' - z$ denote the oracle chain length, where z is the true start time of the chain, and let $l(\widehat{\overline{ac}}) = z' - \hat{z}$ be the estimate returned by the algorithm, where $\hat{z} = re(\hat{J}_1)$. Hence, it suffices to show that $\hat{z} \leq z$. We argue this by backward induction along the chain.

Base case. For the last job, the algorithm constructs $\hat{J}_n = \langle t_{re}, t_{we} \rangle$ from the final two write events of τ_n, where $t_{we} = we(J_n)$ and t_{re} is interpreted as a lower bound on the true read time. By the estimation semantics, we have $t_{re} \leq re(J_n)$, implying that the estimated read-event lower bound does not exceed the oracle one.

Inductive step. Assume that for some i, the estimated read-event lower bound satisfies $re(\hat{J}_i) \leq re(J_i)$. When estimating the preceding job $\widehat{J_{i-1}}$, the algorithm uses $re(\hat{J}_i)$ as the pivot and filters the dataset to write events strictly earlier than this pivot. Since the pivot is no later than the oracle read time $re(J_i)$, this filtering cannot introduce later ones. As a result, the selected pair of write events for $\widehat{J_{i-1}}$ can only stay the same or move earlier in time, and the resulting lower bound $re(\widehat{J_{i-1}})$ is no greater than $re(J_{i-1})$.

By induction, this argument propagates to the first job J_1, yielding $re(\hat{J}_1) \leq re(J_1)$, i.e., $\hat{z} \leq z$, and thus

$$l(\widehat{\overline{ac}}) \geq l(\overline{ac}). \quad \square \tag{16}$$

Lemma 2 (*Bound of the Actual Augmented Job Chain Length*). *Let $\overline{ac} = (z, J_1, \ldots, J_n, z')$ be an immediate backward augmented job chain of a cause–effect chain $E = (\tau_1, \ldots, \tau_n)$ under the periodic task semantics defined in Eq. (13). Then the length of the augmented job chain satisfies*

$$l(\overline{ac}) \in [0,\ 2\sum_{i=1}^{n} T_{\tau_i}). \tag{17}$$

Proof. To bound $l(\overline{ac})$, we decompose the chain length at the read-event of each job:

$$\begin{aligned} l(\overline{ac}) &= z' - z \\ &= z' - re(J_{1,j_1}) \\ &= (z' - re(J_{n,j_n})) + \\ &\quad ((re(J_{n,j_n}) - re(J_{n-1,j_{n-1}})) + \cdots + (re(J_{2,j_2}) - re(J_{1,j_1}))) \\ &= \overbrace{(z' - re(J_{n,j_n}))}^{l_n} + \sum_{i=1}^{n-1} \overbrace{(re(J_{i+1,j_{i+1}}) - re(J_{i,j_i}))}^{l_i}. \end{aligned} \tag{18}$$

Here, l_i denotes the latency contribution associated with task τ_i.

Bounding $l_n = (z' - re(J_{n,j_n}))$. Since z' is provided as the time immediately following $we(J_{n,j_n})$, we have

$$re(J_{n,j_n}) \leq\ we(J_{n,j_n}) \leq z' < we(J_{n,j_n+1}). \tag{19}$$

By the periodic-task semantics in Eq. (13), the following bounds hold:

$$\begin{aligned} &we(J_{n,j_n}) \in [\phi_{\tau_n} + j_n T_{\tau_n}, \phi_{\tau_n} + (j_n+1)T_{\tau_n}], \\ &we(J_{n,j_n+1}) \in [\phi_{\tau_n} + (j_n+1) T_{\tau_n}, \phi_{\tau_n} + (j_n+2)T_{\tau_n}], \\ &re(J_{n,j_n}) \in [\phi_{\tau_n} + j_n T_{\tau_n}, \phi_{\tau_n} + (j_n+1)T_{\tau_n}]. \end{aligned} \tag{20}$$

Combining the first two inequalities in Eq. (20) with Eq. (19) yields

$$z' \in [\phi_{\tau_n} + j_n T_{\tau_n}, \phi_{\tau_n} + (j_n+2)T_{\tau_n}). \tag{21}$$

Hence, by combining the third inequality in Eqs. (20) and (21) with Eq. (19), we obtain

$$l_n = (z' - re(J_{n,j_n})) \in [0,\ 2T_{\tau_n}). \tag{22}$$

Bounding $l_i = (re(J_{i+1,j_{i+1}}) - re(J_{i,j_i}))$ for $i \leq n-1$. By the semantics defined in Eqs. (3) and (13), the following temporal relationship holds:

$$re(J_{i,j_i}) \leq we(J_{i,j_i}) \leq re(J_{i+1,j_{i+1}}) < we(J_{i,j_i+1}). \tag{23}$$

From Eq. (13), the events of τ_i satisfy

$$\begin{aligned} &re(J_{i,j_i}) \in [\phi_{\tau_i} + j_i T_{\tau_i}, \phi_{\tau_i} + (j_i+1)T_{\tau_i}], \\ &we(J_{i,j_i}) \in [\phi_{\tau_i} + j_i T_{\tau_i}, \phi_{\tau_i} + (j_i+1)T_{\tau_i}], \\ &we(J_{i,j_i+1}) \in [\phi_{\tau_i} + (j_i+1) T_{\tau_i}, \phi_{\tau_i} + (j_i+2)T_{\tau_i}]. \end{aligned} \tag{24}$$

Combining Eq. (24) with Eq. (23), we obtain

$$re(J_{i+1,j_{i+1}}) \in [\phi_{\tau_i} + j_i T_{\tau_i}, \phi_{\tau_i} + (j_i+2)T_{\tau_i}). \tag{25}$$

Hence, by combining the first inequality in Eqs. (24) and (25) with Eq. (23), we obtain

$$l_i = (re(J_{i+1,j_{i+1}}) - re(J_{i,j_i})) \in [0, 2T_{\tau_i}). \tag{26}$$

Bounding $l(\overline{ac})$. Summing the bounds derived in Eqs. (22) and (26), we obtain

$$l(\overline{ac}) = l_n + \sum_{i=1}^{n-1} l_i \in [0,\ 2\sum_{i=1}^{n} T_{\tau_i}). \quad \square \tag{27}$$

Lemma 3 (*Bound of the Estimated Augmented Job Chain Length*). *Let $\widehat{\overline{ac}} = (\hat{z}, \hat{J}_1, \ldots, \hat{J}_n, z')$ be the estimated immediate backward augmented job chain obtained by the estimation algorithm. Then the length of the estimated augmented job chain satisfies*

$$l(\widehat{\overline{ac}}) \in [0,\ 3\sum_{i=1}^{n} T_{\tau_i}). \tag{28}$$

Proof. To bound $l(\widehat{\overline{ac}})$, we decompose the chain length at the estimated read-event of each job:

$$\begin{aligned} l(\widehat{\overline{ac}}) &= z' - \hat{z} \\ &= z' - re(\hat{J}_1) \\ &= (z' - re(\hat{J}_n)) \\ &\quad + (re(\hat{J}_n) - re(\widehat{J_{n-1}})) + \cdots + (re(\hat{J}_2) - re(\hat{J}_1)) \\ &= \overbrace{(z' - re(\hat{J}_n))}^{\hat{l}_n} + \sum_{i=1}^{n-1} \overbrace{(re(\widehat{J_{i+1}}) - re(\hat{J}_i))}^{\hat{l}_i}. \end{aligned} \tag{29}$$

Bounding $\hat{l}_n = (z' - re(\hat{J}_n))$. By the algorithm,

$$we(J_{n,k_n-1}) \le re(\hat{J}_n) \le we(J_{n,k_n}) \le z' < we(J_{n,k_n+1}). \tag{30}$$

By the periodic task semantics in Eq. (13), we have

$$\begin{aligned} we(J_{n,k_n-1}) &\in [\phi_{\tau_n} + (k_n-1)T_{\tau_n},\ \phi_{\tau_n} + k_nT_{\tau_n}],\\ we(J_{n,k_n}) &\in [\phi_{\tau_n} + k_nT_{\tau_n},\ \phi_{\tau_n} + (k_n+1)T_{\tau_n}],\\ we(J_{n,k_n+1}) &\in [\phi_{\tau_n} + (k_n+1)T_{\tau_n},\ \phi_{\tau_n} + (k_n+2)T_{\tau_n}] \end{aligned} \tag{31}$$

Combining Eq. (31) with Eq. (30), we obtain

$$\begin{aligned} z' &\in [\phi_{\tau_n} + k_nT_{\tau_n},\ \phi_{\tau_n} + (k_n+2)T_{\tau_n}),\\ re(\hat{J}_n) &\in [\phi_{\tau_n} + (k_n-1)T_{\tau_n},\ \phi_{\tau_n} + (k_n+1)T_{\tau_n}]. \end{aligned} \tag{32}$$

Hence, by combining the inequalities in Eq. (32) with Eq. (30)

$$\hat{l}_n = (z' - re(\hat{J}_n)) \in [0,\ 3T_{\tau_n}). \tag{33}$$

Bounding $\hat{l}_i = (re(\widehat{J_{i+1}}) - re(\hat{J}_i))$ *for* $i \le n-1$. The algorithm estimates the read-event of job $\hat{J}_i$ as the second-latest write-event before the estimated read-event of its successor $re(\widehat{J_{i+1}})$ (Lines 4–7). Let the estimated value be $re(\hat{J}_i) = we(J_{i,x})$.

By the semantics defined in Eqs. (3) and (13), we have

$$we(J_{i,x}) \le re(\hat{J}_i) \le we(J_{i,x+1}) \le re(\widehat{J_{i+1}}) < we(J_{i,x+2}). \tag{34}$$

From Eq. (13), the events of τ_i satisfy

$$\begin{aligned} we(J_{i,x}) &\in [\phi_{\tau_i} + xT_{\tau_i},\ \phi_{\tau_i} + (x+1)T_{\tau_i}],\\ we(J_{i,x+1}) &\in [\phi_{\tau_i} + (x+1)T_{\tau_i},\ \phi_{\tau_i} + (x+2)T_{\tau_i}],\\ we(J_{i,x+2}) &\in [\phi_{\tau_i} + (x+2)T_{\tau_i},\ \phi_{\tau_i} + (x+3)T_{\tau_i}] \end{aligned} \tag{35}$$

Combining Eq. (35) with Eq. (34), we obtain

$$\begin{aligned} re(\widehat{J_{i+1}}) &\in [\phi_{\tau_i} + (x+1)T_{\tau_i},\ \phi_{\tau_i} + (x+3)T_{\tau_i}),\\ re(\hat{J}_i) &\in [\phi_{\tau_i} + xT_{\tau_i},\ \phi_{\tau_i} + (x+2)T_{\tau_i}]. \end{aligned} \tag{36}$$

Hence,by combining the inequalities in Eq. (36) with Eq. (34)

$$\hat{l}_i = (re(\widehat{J_{i+1}}) - re(\hat{J}_i)) \in [0,\ 3T_{\tau_i}). \tag{37}$$

Bounding $l(\widehat{\overline{ac}})$. Summing the bounds derived in Eqs. (33) and (37), we obtain

$$l(\widehat{\overline{ac}}) = \hat{l}_n + \sum_{i=1}^{n-1} \hat{l}_i \in [0,\ 3\sum_{i=1}^{n} T_{\tau_i}). \quad \square \tag{38}$$

Theorem 1 (*Bound on the Over-Estimation Error*)**.**
Let $\overline{ac}$ *be the oracle augmented job chain and* $\widehat{\overline{ac}}$ *the estimated chain returned by the algorithm for a given pivot time* z'*. Then the over-estimation error* $\Delta = l(\widehat{\overline{ac}}) - l(\overline{ac})$ *satisfies*

$$0 \le \Delta < 3\sum_{i=1}^{n} T_{\tau_i}. \tag{39}$$

Proof. By Lemmas 2 and 3,

$$l(\overline{ac}) \in \left[0,\ 2\sum_{i=1}^{n} T_{\tau_i}\right) \quad \text{and} \quad l(\widehat{\overline{ac}}) \in \left[0,\ 3\sum_{i=1}^{n} T_{\tau_i}\right). \tag{40}$$

Hence,

$$\Delta = l(\widehat{\overline{ac}}) - l(\overline{ac}) \in \left(-2\sum_{i=1}^{n} T_{\tau_i},\ 3\sum_{i=1}^{n} T_{\tau_i}\right). \tag{41}$$

Moreover, by Lemma 1, $\Delta \ge 0$ holds for any pivot time z'. Therefore,

$$\Delta \in \left[0,\ 3\sum_{i=1}^{n} T_{\tau_i}\right). \quad \square \tag{42}$$

This theorem provides a formal guarantee that the algorithm's over-estimation error is bounded by constants determined by the task periods, under the system models specified in Section 3.4. Consequently, practitioners can pre-compute the worst-case error budget based on available system knowledge or design assumptions; in particular, the maximum over-estimation is $\Delta_{\max} < 3\sum_{i=1}^{n} T_{\tau_i}$. For example, in the extreme case where the oracle chain length $l(\overline{ac})$ approaches its minimum while the estimated chain length $l(\widehat{\overline{ac}})$ approaches its upper bound, the error approaches $3\sum_{i=1}^{n} T_{\tau_i}$. Conversely, if both $l(\overline{ac})$ and $l(\widehat{\overline{ac}})$ approach their upper bounds, the gap approaches $\sum_{i=1}^{n} T_{\tau_i}$, i.e., roughly 50% of the actual chain length in that regime.

While this theoretical bound is conservative, it provides a formal worst-case guarantee on the estimation error based solely on observable write-events in the runtime black-box setting, which we further evaluate empirically in Section 7.

6. Verification of the cause–effect latency

The cause–effect chain latency estimate is defined as $\widehat{\mathrm{Lat}}_t(E) = \sup\{l(\widehat{\overline{ac}})\}$ in Eq. (10), the *least upper bound* of job-chain lengths. Since the exact supremum is intractable without full system knowledge, we instead estimate a *statistical* supremum from samples of job-chain lengths collected online. The verification problem becomes a statistical tolerance test: *Does the upper p% tolerance limit (UTL) of the sample distribution lie below the threshold* δ*, with confidence* γ*?* This enables statistically sound bounds at prescribed coverage and confidence, using only recent samples in a sliding window.

We solve this online with a sequential procedure combining Welford's mean/variance updates [11] and Howe's k-factor tolerance intervals [12]. At each step, a one-sided tolerance bound is computed, declaring `safe` if below δ or `unsafe` otherwise. This ensures rigorous, efficient verification while minimizing sample requirements at runtime.

Algorithm 2 presents our verification procedure based on one-sided tolerance intervals. The **Precondition** is a given cause–effect chain E under verification. The **Input** consists of a sequence of estimated job-chain lengths $L = [\hat{l}_1, \hat{l}_2, \ldots, \hat{l}_k]$ passed from the estimator; the latency threshold δ, specifying the desired upper bound; the coverage proportion $0 < p < 1$ (e.g., 95%), defining which population quantile must lie below δ; the confidence level $0 < \gamma < 1$ (e.g., 99%), guaranteeing statistical reliability; and two runtime bounds: the minimum number of warm-up samples $n_{\min}$ and the optional maximum number of samples $n_{\max}$ used in a single verification run.

The **Output** reports a verification verdict $\psi \in \{\texttt{safe}, \texttt{unsafe}\}$. Here, `safe` (corresponding to a *positive* verification result) indicates that the requirement $\mathrm{Lat}_t(E) \le \delta$ is expected to hold with the prescribed coverage and confidence, where this requirement captures a safety-related constraint on the end-to-end latency, while `unsafe` (corresponding to a *negative* verification result) indicates that the requirement is statistically violated. In addition, the algorithm reports the current estimate of the cause–effect latency, given by the computed upper tolerance limit $\widehat{\mathrm{Lat}}_t(E)$.

For the **interpretation of the verification verdict**, due to the statistical nature of the verification, the reported verdict may be subject to false positives and false negatives. In this context, a *false positive* corresponds to reporting `safe` when the requirement $\mathrm{Lat}_t(E) \le \delta$ is in fact violated, whereas a *false negative* corresponds to reporting `unsafe` despite the requirement being satisfied. These outcomes reflect different aspects of verification trustworthiness, ranging from potential safety risks due to false positives to conservative verification decisions due to false negatives, and are analyzed in detail in Section 7.3.

The **Init** block initializes the online statistics and decision variables. It sets the sample count $n \leftarrow 0$, the running mean $\bar{x} \leftarrow 0$, and the second-moment accumulator $M_2 \leftarrow 0$ for Welford's updates [11]. The verdict is initialized as $\psi \leftarrow$ NaN, and the estimated latency upper and lower limit as $u \leftarrow$ NaN and $l \leftarrow$ NaN.

Algorithm 2: Cause–Effect latency verification

Precon.: Cause–Effect chain E
Input : Samples $L = [\widehat{l}_1, \widehat{l}_2, \ldots, \widehat{l}_k]$,
Upper limit threshold δ,
Content (coverage) $p \in (0, 1)$,
Confidence level $\gamma \in (0, 1)$,
Warm-up steps $n_{\min} \geq 3$,
Optional max steps $n_{\max}$ ($\geq n_{\min}$)
Output : Verdict $\psi \in \{\texttt{safe}, \texttt{unsafe}\}$,
Latency estimate $\widehat{\mathrm{Lat}}_t(E)$
Init : $n \leftarrow 0,\ \bar{x} \leftarrow 0,\ M_2 \leftarrow 0,$
$\psi \leftarrow \texttt{NaN},\ u \leftarrow \texttt{NaN}, l \leftarrow \texttt{NaN}$
Def : `Welford`$(n, \bar{x}, M_2, x)$:
Online mean/variance update [11]
return $n, \bar{x}, M_2$
`HoweK`(p, γ, ν, n):
k-factor for normal tolerance intervals [12]
return k

1 **foreach** $x \in L$ *in reverse order* **do**
2 $n, \bar{x}, M_2 \leftarrow \texttt{Welford}(n, \bar{x}, M_2, x)$
3 **if** $n < n_{\min}$ **then continue**
4 $\nu \leftarrow n - 1$
5 $s \leftarrow \sqrt{M_2/\nu}$
6 $k \leftarrow \texttt{HoweK}(p, \gamma, \nu, n)$
7 $u \leftarrow \bar{x} + ks$
8 $l \leftarrow \bar{x} - ks$
9 **if** $u \leq \delta$ **then**
10 $\psi \leftarrow \texttt{safe}$, **break** // $\widehat{\mathrm{Lat}}_t(E) \leq \delta$
11 **end**
12 **else if** $l > \delta$ **then**
13 $\psi \leftarrow \texttt{unsafe}$, **break** // $\widehat{\mathrm{Lat}}_t(E) \not\leq \delta$
14 **end**
15 **else if** $n = \min(n_{max}, |L|)$ **then**
16 $\psi \leftarrow \texttt{unsafe}$, **break** // $\widehat{\mathrm{Lat}}_t(E) \not\leq \delta$
17 **end**
18 **end**
19 $\widehat{\mathrm{Lat}}_t(E) \leftarrow u$
20 **return** $\psi, \widehat{\mathrm{Lat}}_t(E)$

For the procedure, two statistical tools are employed in the **Def** block. Welford's algorithm [11] enables numerically stable online updates of the mean and variance, supporting sequential latency estimates without storing past data. Howe's k-factor method [12] provides exact constants for one-sided tolerance intervals with unknown variance, enabling rigorous verification under the unknown latency distribution of a black-box system. Together, they form the statistical foundation for the procedure described below.

The procedure begins by iterating over the job-chain length samples starting from the most recent one (**Line 1**). For each new sample, the mean and variance are updated online using Welford's method (**Line 2**). If the number of processed samples has not yet reached the minimum warm-up threshold $n_{\min}$, the algorithm skips decision making and retrieves the next sample (**Line 3**). Once enough samples are available, the procedure computes the sample standard deviation (**Line 5**), retrieves Howe's k-factor for the current sample size and degrees of freedom (**Line 6**), and then constructs the upper and lower tolerance limits (u and l, **Lines 7–8**).

The k-factor serves as the principal mechanism for estimating the upper tolerant limit. Intuitively, k increases when: (i) the required coverage proportion p is higher, pushing the tolerance bound further; (ii) the required confidence level γ is higher, demanding stronger statistical guarantees; or (iii) the available sample size n (and thus the degrees of freedom ν) is smaller, reflecting greater uncertainty from limited data.

Based on these bounds, decisions are made as early as possible using the smallest number of samples, while focusing on the most recent data whenever possible. If the upper bound u lies below the threshold δ, the system is declared `safe` (**Line 10**). If the lower bound l lies above δ, the system is deemed `unsafe`, even if not all samples have been consumed (**Lines 13–16**). If neither condition is met (e.g., when $u > \delta$ but $l \leq \delta$, meaning the upper tolerance bound suggests possible violation while the distribution is not fully above the threshold), the algorithm continues to process additional samples. If the maximum number of samples $n_{\max}$ or the dataset size $|L|$ is reached without a conclusive decision, the system is judged `unsafe` solely based on the final u. At this point, the most recent upper tolerance limit is reported as the runtime estimate of the cause–effect latency $\widehat{\mathrm{Lat}}_t(E)$ (**Line 19**). Finally, the algorithm returns both the verdict and the latency estimate (**Line 20**).

The verification algorithm is executed periodically with new length samples accumulated by the estimator. The verdict ψ is continuously updated to indicate whether the requirement $\mathrm{Lat}_t(E) \leq \delta$ holds, while incurring only the minimal cost needed to guarantee the prescribed coverage and confidence. The feasibility of adopting tolerance-interval-based verification in this context, as well as the empirical cost of our algorithm, is evaluated in Section 7.

7. Evaluation

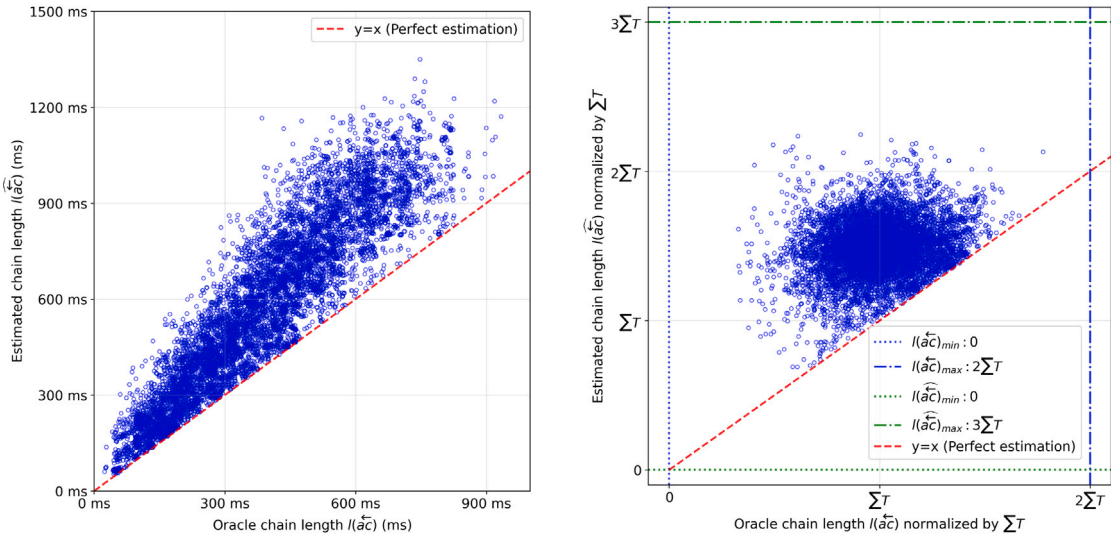
This section evaluates our proposed approach for runtime verification of cause–effect latency in black-box systems by addressing four research questions (RQs):

- **RQ1:** Does the job-chain estimation algorithm satisfy the theoretical over-estimation error bound?
- **RQ2:** How do chain characteristics (e.g., utilization, number of tasks, task periods) affect the estimation error?
- **RQ3:** How trustworthy are the results of our verification algorithm?
- **RQ4:** Are the estimation and verification algorithms lightweight enough for online deployment?

The following subsections address these questions. Section 7.1 validates the theoretical error bound and quantifies empirical over-estimation. Section 7.2 investigates how chain parameters influence estimation accuracy. Section 7.3 evaluates the statistical reliability of verification under varying tolerance (p) and confidence (γ) levels. Finally, Section 7.4 measures runtime overhead, confirming that both algorithms operate efficiently for real-time use.

Although our approach targets black-box systems, the evaluation is conducted through *white-box simulations*, where the ground-truth latencies, grounded in the model described in Section 3.4, are observable but remain unknown to our approach, rather than relying on a purely black-box or confounded evaluation setting. The cause–effect chain parameters are generated synthetically in a domain-agnostic manner to introduce controlled randomness suited to each experiment's objective, without relying on domain-specific knowledge or general task-graph models typically used for scheduling benchmarks [13–15]. Detailed settings are explained in the following subsections. All experiments were performed on a workstation with an Intel Core i9-class CPU and 64 GB DDR5 RAM running Ubuntu 22.04 LTS. The evaluation was purely CPU-based without GPU parallelization, ensuring that the reported runtime costs reflect algorithmic efficiency rather than hardware acceleration. The experimental code used in this evaluation is available in a repository.[1]

[1] https://github.com/yongjunshin/Runtime_Verification_of_Cause_Effect_Latency_in_Black_box_Systems



(a) Oracle vs. estimated job-chain lengths.

(b) Normalized oracle vs. estimated job-chain lengths.

Fig. 7. (RQ1) Comparison of oracle and estimated job-chain lengths. Subfigure (a) shows absolute chain lengths produced by simulation, while subfigure (b) shows lengths normalized by the theoretical chain–length bound of the simulated semantics.

7.1. Validation of over-estimation error bound

In RQ1, we aim to validate the correctness of the theoretical error bound of our estimation algorithm. To this end, mimicking the uncertainty of real cause–effect chains in practice, we simulate random chains and collect their execution traces, which include read- and write-events as well as the oracle job-chain lengths. The oracle lengths serve as ground truth against which our estimation algorithm is evaluated. Using only the write-event traces, our algorithm reconstructs job chains and produces estimated lengths. We then compare these estimates to the oracle values, focusing primarily on confirming that the over-estimation error never exceeds the bound proved in Section 5.2, while also assessing estimation accuracy within our random simulation setting.

For the experiment, each chain E was modeled as an $|E|$-length sequence of tasks $\tau = (C, T, \phi)$ as defined in Section 3.4, without any additional tasks outside the chain causing external interference. Specifically, we considered 180 random profiles with the following parameter ranges: chain length $|E| \in \{2, \dots, 10\}$, task period $T \in [20, 100]$ ms, worst case job utilization ratio $C/T \in [0.1, 0.9]$, and task phase $\phi \in [0, T]$, where all parameters were sampled uniformly from integer values except for ratio C/T, which was sampled from a continuous uniform distribution. Profiled tasks in E were allowed to execute concurrently assuming sufficient processing resources, which abstracts away scheduling interference and narrows the nondeterminism to the dynamic placement of read- and write-events, so that the execution time of a job J corresponds to $we(J) - re(J)$. For each job release, the execution time was first sampled uniformly from integer values in $[1, C]$. Then a feasible pair of integer events $(re(J), we(J))$ satisfying $we(J) - re(J)$ equal to the sampled execution time was uniformly placed within the corresponding period. This results in dynamically varying execution times and consequently variability in the resulting job chains. These settings aim to simulate unknown chains encountered in practice through unbiased, uniformly random sampling, without any adversarial or favorable bias toward our estimation.

For each chain length $|E|$ (9 cases), 20 random profiles were generated. Each profile was simulated 50 times to account for nondeterminism in the job chains. In total, this procedure produced 9000 traces, providing the oracle job-chain lengths as ground truth.

Fig. 7 presents the comparison between oracle and estimated job-chain lengths. Subfigure (a) shows the raw simulation results: the x-axis gives the oracle job-chain length, which varies across random chain profiles depending on the number of tasks, task periods, and utilization. As the number of tasks or task periods increases, the oracle chain length tends to grow. The y-axis shows the corresponding estimates produced by our algorithm. The over-estimation property is clearly evident, as all points lie on or above the diagonal line; in other words, the estimated length is always greater than or equal to the oracle length, providing a conservative guarantee of safety.

To remove the direct influence of random chain profile parameters, Subfigure (b) reconstructs the plot using values normalized by the theoretical bound: the oracle length $l(\overleftarrow{ac}) \in [0, 2\sum T]$ and the estimated length $\widehat{l(\overleftarrow{ac})} \in [0, 3\sum T]$ for each simulation. All points fall strictly within the theoretical error bound proved in Section 5.2, confirming that the over-estimation is correct and conservative. The theoretical bound represents a worst-case over-estimation scenario, which can occur only under unfavorable alignments of job executions—for example, when the oracle chain achieves near-minimal propagation while the backward estimation incurs unfavorable selections at every step. In contrast, our simulation profiles are generated using uniformly randomized parameters to reflect chains that may be encountered without any adversarial intent, making such adversarial settings across all parameters highly unlikely. As a result, the observed estimation errors remain well within the interior of the bound, reflecting typical estimation behavior that users may expect when encountering unknown chains in practice. Nevertheless, the proven bound remains valid as a theoretical guarantee against all possible worst-case conditions.

Fig. 8 shows the histogram of estimation errors across all simulation results, where the error is defined as $\Delta / l(\overleftarrow{ac})$ with $\Delta = \widehat{l(\overleftarrow{ac})} - l(\overleftarrow{ac})$. The average error was 53.99%, the 90th percentile was 95.7%, and the maximum error was about 200%. In concrete terms, if the oracle chain length were 100 ms, then half of the estimates would be below 153 ms, 90% of cases below 195 ms, and in the worst case the estimate could reach nearly 300 ms. These results confirm that our algorithm consistently produces conservative estimates of job-chain length, in line with the theoretical guarantees.

Although the magnitude of error may appear large, it provides practitioners with a clear margin when only write-events are observable in a black-box system. This margin quantifies the conservativeness of verification: the procedure never yields a false positive for $\mathrm{Lat}_t(E) \le \delta$, and the true latency $\mathrm{Lat}_t(E)$ may on average be up to 53% smaller than the reported $\widehat{\mathrm{Lat}}_t(E)$. By acknowledging this conservativeness,

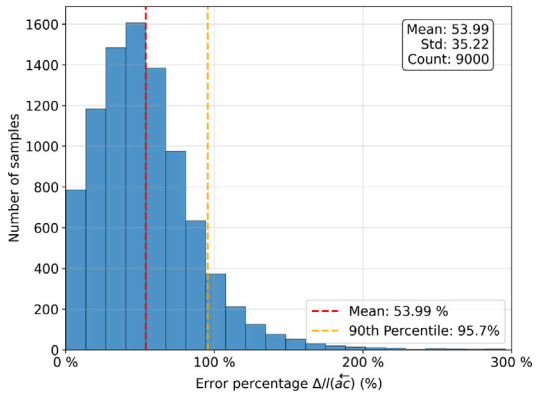


Fig. 8. (RQ1) Histogram of estimation error expressed as a percentage of the oracle job-chain length.

engineers can obtain runtime latency verification results without significant system intrusion and leverage them in subsequent iterative system improvement. Nevertheless, the random-simulation analysis alone is insufficient to predict the over-estimation error Δ more precisely; we investigate this aspect further in RQ2.

> **Answer to RQ1:** Our job-chain length estimation always produces an over-estimate, ensuring safe verification when only write-events are observable in a black-box system. The experiments confirm that the theoretical bound of the over-estimation error proved in Section 5.2 is never violated.

7.2. Impact of chain characteristics on estimation error

In RQ2, we aim to analyze how the degree of over-estimation error varies with the characteristics of a cause–effect chain. To this end, we conducted controlled experiments focusing on three key parameters derived from the semantic definition: the number of tasks $|E|$, the task period T, and the job utilization ratio C/T, while fixing the job execution time to its worst-case value C. Building on the random profile generation used in RQ1, we introduced controlled variations in these parameters to isolate and evaluate their individual impact on the estimation error.

Specifically, we generated random chain profiles while controlling the target parameters and fixing the others: (1) varying $|E| \in \{2, \dots, 10\}$ with four edge-case combinations of $T \in \{20, 100\}$ and $C/T \in \{0.1, 0.9\}$, (2) varying $T \in [20, 100]$ ms with four edge-case combinations of $|E| \in \{2, 10\}$ and $C/T \in \{0.1, 0.9\}$, and (3) varying $C/T \in [0.1, 0.9]$ with four edge-case combinations of $|E| \in \{2, 10\}$ and $T \in \{20, 100\}$.

To isolate the independent variable, all tasks within a chain shared the same profile (i.e., identical T and C/T). For each experiment, we generated 200 uniformly random parameter profiles and repeated 5 simulations for each of the four controlled settings, producing 1000 results for each of the three experiments. From the resulting traces, we obtained the oracle job-chain lengths (ground truth) and the corresponding estimates from our algorithm, and then analyzed the error ratio $\Delta/l(\overline{ac})$ as the dependent variable.

Fig. 9 illustrates how the estimation error varies with different chain characteristics. The three subfigures are ordered by decreasing impact of the independent parameter on the error.

Subfigure (a) shows the relationship between job utilization and error. A strong negative correlation is observed: when utilization is low (e.g., 0.1), the average error exceeds 100%, which corresponds to the tail cases seen in Fig. 8. As utilization increases, the error steadily decreases; above 0.6 utilization the average error drops to around 50% (close to the overall average in Fig. 8), and at 0.9 utilization it falls to as low as about 6%. This behavior arises because our algorithm infers unknown read-events solely from write-events: given two consecutive write-events $we(J_{i,j-1})$ and $we(J_{i,j})$ of task τ_i, the unknown read-event $re(J_{i,j})$ must lie between them; with higher utilization, this forces $re(J_{i,j})$ to be closer to $we(J_{i,j-1})$, reducing uncertainty and thus the estimation error.

Subfigure (b) shows the effect of the number of tasks in the chain. The relationship is relatively weak, but a slight downward trend is visible: as $|E|$ increases, the estimation error is accumulated across more tasks, which reduces the likelihood of extremely high errors. However, the average error does not change significantly with $|E|$, indicating that task count has only a limited effect on overall accuracy.

Subfigure (c) shows the effect of task period. No significant correlation is observed, indicating that the task period alone does not strongly influence estimation error. If the task period is longer, the absolute error has more chance to increase; however, since the error ratio is normalized by the oracle chain length, this dependency does not appear strongly in the relative error.

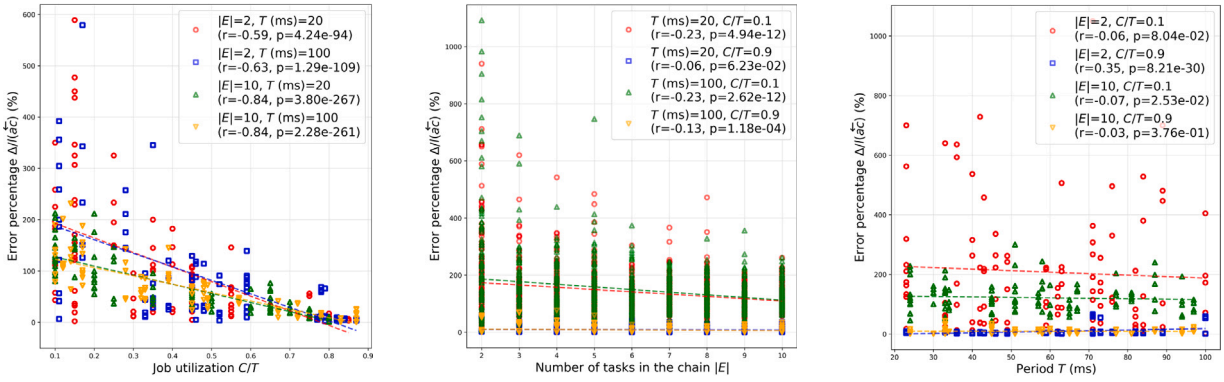
Overall, the results highlight that job utilization is the dominant factor affecting estimation error. Fig. 10 illustrates this with concrete simulation traces under different utilization scenarios. In the heterogeneous case (Sub Fig. 10(a)), the error was about 56%, which is close to the average observed in RQ1. When utilization was extremely low (Sub Fig. 10(b)), the error rose to 121%, demonstrating how idle gaps between write- and read-events can exacerbate over-estimation. Conversely, at high utilization (Sub Fig. 10(c)), the error decreased sharply to only 6%, as dense execution of tasks leaves little uncertainty for the estimator.

For practitioners, this implies that the accuracy of our estimation algorithm improves significantly when the chain operates at high utilization. In other words, if the periodic tasks of a chain leave little idle time in resource usage, the verification results are expected to be highly accurate. Recall that this is because our algorithm infers unknown read-events solely from write-events, and higher utilization places read-events closer to preceding write-events, thereby limiting estimation error. Although job utilization may be unknown in a black-box setting, it is often the case that periodic tasks are designed with well-chosen periods to achieve relatively high utilization, in which case the error of our estimation algorithm will remain limited. Moreover, when some knowledge or assumptions about utilization are available, the resulting estimation error can be better interpreted, making verification outcomes more reliable during system development as well as in-field monitoring.

> **Answer to RQ2:** Estimation error is mainly affected by job utilization of the chain: as utilization increases, the error decreases significantly. At high utilization (e.g., $C/T \approx 0.9$), the error ratio dropped to around 6% on average, enabling highly accurate estimates in realistic, well-utilized chains. Other factors such as task count and period showed only minor influence on the error.

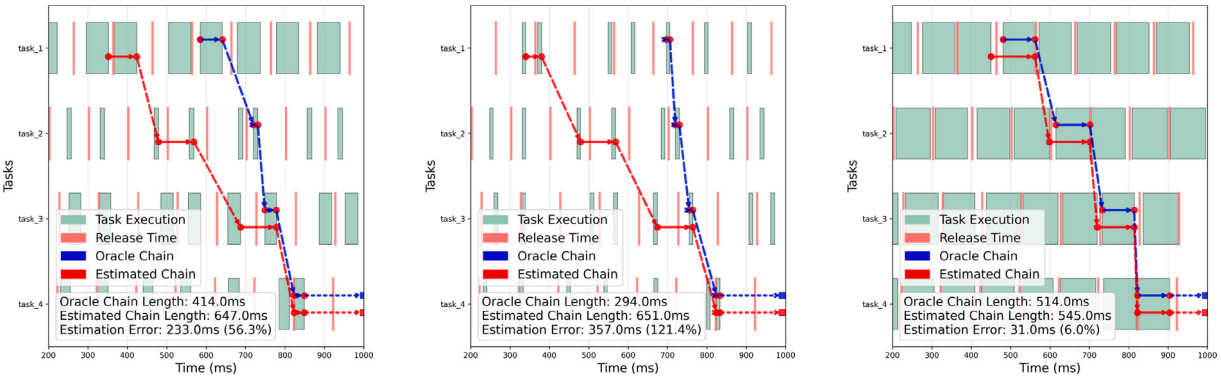
7.3. Trustworthiness of verification results

In RQ3, we aim to evaluate the trustworthiness of the verification algorithm by analyzing the correctness of its two outputs: the verdict ψ and the reported latency estimate $\widehat{\mathrm{Lat}}_t(E)$. First, the verdict $\psi \in \{\texttt{safe}, \texttt{unsafe}\}$ indicates whether the requirement $\mathrm{Lat}_t(E) \le \delta$ holds under the prescribed coverage p and confidence γ. Here, `safe` corresponds to a *positive* verification outcome with respect to the timing constraint, while `unsafe` corresponds to a *negative* outcome. Accordingly, the verdict may be subject to *false positives*, which can lead to safety risks by incorrectly accepting a violated constraint, and *false negatives*, which arise as a conservative trade-off by rejecting a satisfied constraint. In this evaluation, we analyze the rates of false positives and false



(a) Job utilization vs. error. (b) Task count vs. error. (c) Task period vs. error.

Fig. 9. (RQ2) Factors influencing over-estimation error. Scatter plots illustrate how (a) utilization, (b) task count, and (c) task period affect the relative estimation error.



(a) Heterogeneous utilization across tasks. (b) Low utilization across tasks. (c) High utilization across tasks.

Fig. 10. (RQ2) Simulation visualizations of cause–effect chains with all tasks sharing the same release time.

negatives. Second, the reported value $\widehat{\mathrm{Lat}}_t(E)$ represents the p% upper tolerance limit estimated at runtime, which should closely approximate the true p% quantile of the samples. For this output, we analyze the estimation error with respect to the corresponding ground-truth value. Together, these outputs determine whether the algorithm can provide reliable runtime decisions and meaningful latency bounds.

To evaluate this, we designed a controlled experiment to compare verification outcomes against ground truth. We generated 10,000 random scenarios, assuming that the job-chain length samples provided to the statistical verification algorithm follow a normal distribution to model a general form of uncertainty where samples vary around a mean with a fixed variance. The mean was uniformly drawn from $[90, 110]$ and the standard deviation was fixed to $\sigma = 1$. For each scenario, the verification algorithm was tasked with checking whether $\mathrm{Lat}_t(E) \leq 100$ ms. The algorithm was configured with varying coverage p and confidence γ, each chosen from $\{0.90, 0.92, 0.95, 0.97, 0.99\}$.

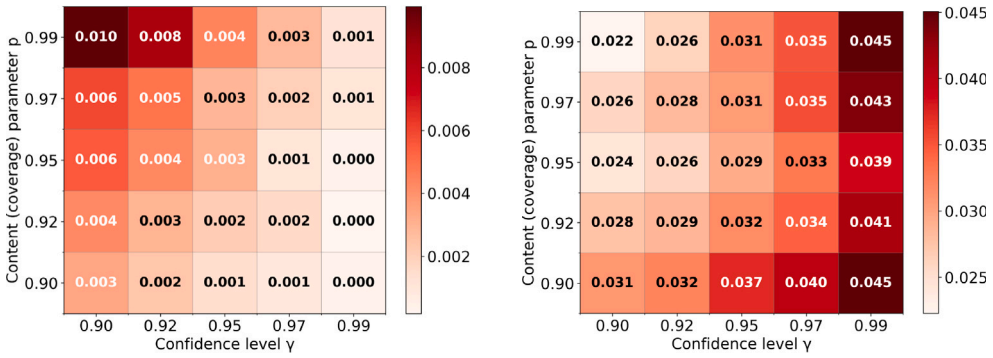
For example, if the sample distribution had mean 98.7 ms and $\sigma = 1$, the 95% quantile would be about 100.3 ms, slightly exceeding the requirement $\delta = 100$ ms. In this case, the correct verdict should be unsafe. However, if the algorithm — configured with $p = 0.95$ and $\gamma = 0.95$ — reported $\widehat{\mathrm{Lat}}_t(E) = 99$ ms, it would incorrectly return safe, constituting a false positive. Conversely, if the true value satisfied the requirement but the algorithm returned unsafe, this would constitute a false negative.

Note that this RQ analyzes only the errors in the verification algorithm's outputs ψ and $\widehat{\mathrm{Lat}}_t(E)$, based on the synthetic job-chain length samples $L = [\hat{l}_1, \hat{l}_2, \ldots, \hat{l}_k]$. The broader estimation error between the true job-chain lengths $l(\overline{ac})$ and their estimates in the black-box setting $l(\widehat{\overline{ac}})$ remains the subject of RQ1 and RQ2.

Fig. 11 presents the error rates of the verification verdict ψ. Subfigure (a) shows the false positive rates, which are very small, mostly below 0.006 and never exceeding 0.010. As expected, higher confidence γ decreases false positives, since stricter verification pushes latency estimates more conservatively. Conversely, decreasing p weakly reduces false positives, as Howe's k-factor makes the p-quantile prediction more conservative when coverage is lower. Most importantly, all false positive rates remain below the prescribed confidence γ, meaning the observed error is always within the level of risk explicitly accepted by the user.

Subfigure (b) shows the false negative rates, which range from about 0.022 to 0.045. In general, false negatives are more frequent than false positives, since the algorithm is deliberately designed to verify conservatively. Oppositely to the false positive rate, the settings that increase false positives also make false negatives more likely, effectively raising the chance of false alarms even when the true latency satisfies the requirement. This reflects the conservative bias of the algorithm, which favors safety by avoiding unsafe acceptance at the cost of occasionally rejecting safe cases.

Overall, these results show that the verification algorithm strongly prioritizes safety by keeping false positives extremely rare, at the expense of slightly higher false negatives. In practice, the confidence level γ serves as a major handle for managing this trade-off: increasing γ reduces the risk of false positives but raises the likelihood of false negatives.



(a) False positive error rates.

(b) False negative error rates.

Fig. 11. (RQ3) False positive and false negative error rates of the verification verdict ψ under varying coverage p and confidence γ.

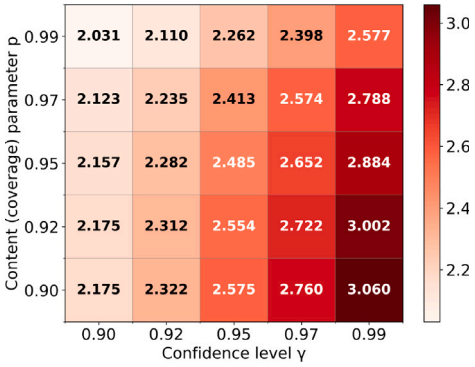


Fig. 12. (RQ3) Heatmap of the error of the reported latency $\widehat{\mathrm{Lat}}_t(E)$ compared to the true upper bound, across different parameters p and γ.

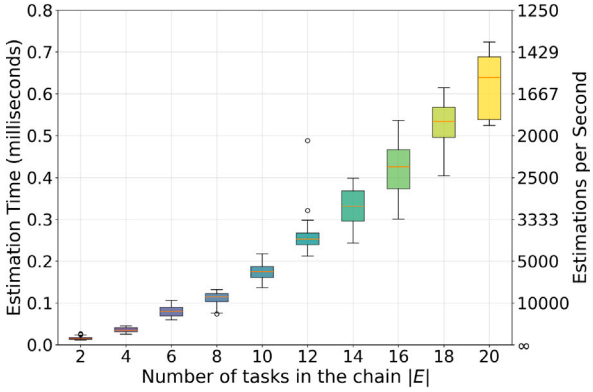


Fig. 13. (RQ4) Execution time of the estimation algorithm across repeated runs. The distribution shows that estimation incurs minimal overhead suitable for online use.

Fig. 12 shows the deviation of the reported cause–effect latency estimate $\widehat{\mathrm{Lat}}_t(E)$ from the $p\%$ upper tolerance limit of the given length samples across different (p, γ) settings. As the parameters become more conservative (higher γ or lower p), the deviation increases. This trend is expected: stricter verification requires a larger safety margin, which manifests as higher reported latency estimates. In our simulation setup, where the sample means were uniformly drawn from $[90, 110]$, the deviation reached at most about 3 ms.

Answer to RQ3: The verification algorithm is trustworthy for safe verification of $\mathrm{Lat}_t(E) \leq \delta$: false positive rates remain extremely low (always below the prescribed confidence level γ), while latency estimates closely track the true $p\%$ upper tolerance limit with only small deviations under conservative settings. As a trade-off, false negatives occur more often than false positives.

7.4. Runtime cost of estimation and verification

In RQ4, we aim to evaluate the runtime cost of our algorithms to determine their suitability for online deployment. To this end, we measured the execution times of both the estimation and verification algorithms. Specifically, we report the execution time as a function of the key parameters that influence runtime until the algorithm produces an output. For the estimation algorithm, this parameter is the number of tasks $|E|$ in the cause–effect chain. For the verification algorithm, the runtime is primarily determined by the number of estimation samples processed n, which is bounded within $n_{\min}$ and $n_{\max}$. These parameters mainly affect how many iterations the algorithm performs, and thus directly impact the total execution time. Each configuration was repeated 30 times to account for variability.

Fig. 13 shows the execution time of the estimation algorithm when analyzing job-chains of varying length from the given write-event logs. The x-axis represents the number of tasks in the chain $|E|$, while the left y-axis shows the measured execution time and the right y-axis converts this to the equivalent number of estimations per second. As expected, the runtime increases linearly with $|E|$, since longer chains require more inference steps. Nevertheless, the algorithm remains lightweight: even for chains of 20 tasks, the average execution time is only about 0.65 ms, corresponding to over 1500 estimations per second. In practice, this performance is sufficient to handle demanding scenarios such as pipelines operating at 100 FPS, where the estimator can reliably update job-chain lengths whenever the end task completes.

Fig. 14 shows the execution time of the verification algorithm as a function of the number of estimation samples n. The x-axis represents n, while the left y-axis shows the measured verification time and the right y-axis indicates the equivalent number of verifications per second. Each measurement corresponds to running the verification algorithm once, producing both the verdict ψ and the reported latency bound $\widehat{\mathrm{Lat}}_t(E)$ from the given sample dataset. The most computationally expensive operation is computing Howe's k-factor; however, since k depends only on (p, γ, n), it can be cached to greatly reduce overhead. Accordingly, the two subfigures compare results without caching (top) and with caching (bottom), showing that caching reduces execution time by nearly an order of magnitude.

Overall, the verification algorithm is lightweight: with $n = 100$ samples, verification takes less than 0.14 ms (about 7000 verifications

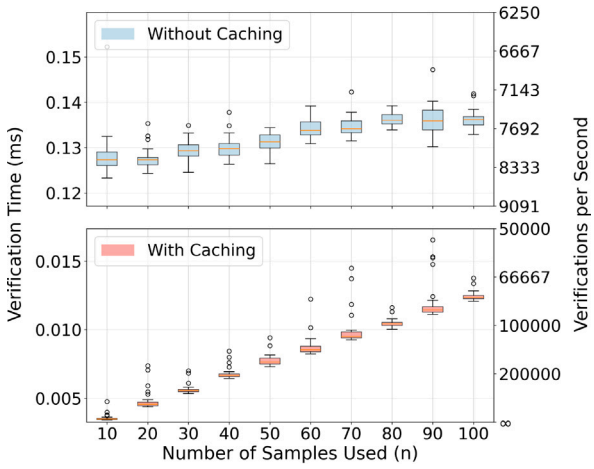


Fig. 14. (RQ4) Execution time of the verification algorithm as a function of the latency threshold δ. Results confirm that verification also imposes negligible runtime cost.

per second), and with caching the cost drops to roughly one-tenth of that. While runtime grows linearly with the given sample length n, reflecting the additional work required for more reliable tolerance-limit estimates, the cost remains low enough for frequent online use. Moreover, when the decision can be made early (e.g., in a clear positive or negative case), the required number of samples may be close to $n_{\min}$, further reducing runtime cost.

Taken together, these results demonstrate that both the estimation and verification algorithms are lightweight, making them well-suited for runtime verification that must be repeated frequently. We emphasize that these measurements capture only the raw execution time of the algorithms themselves. In a practical system, additional costs such as data collection, event parsing, and integration overhead may arise, but the core algorithms introduce negligible computational burden.

Answer to RQ4: Both the estimation and verification algorithms incur minimal runtime cost. The estimation algorithm scales linearly with the number of tasks $|E|$, enabling over 1500 runs per second for $|E| = 20$. The verification algorithm scales with the number of samples n, achieving about 7000 runs per second for $n = 100$, and up to ten times faster with caching.

8. Case study experience and discussion

In this section, we revisit the autonomous driving system introduced in Section 2, applying our estimation and verification algorithms in practice and discussing their applicability and limitations. While the evaluation in Section 7 relied on controllable simulations, this case study was conducted under real black-box conditions, where only write-events were observable and system internals remained opaque. Given these industrial constraints, our analysis is necessarily limited in depth, but we share our practical experience to highlight key insights and outline directions for future application.

As introduced in Section 2 and illustrated in Fig. 1, the software-defined autonomous driving system was developed within a consortium of industrial and research partners, where different companies and institutes implemented individual components of the vehicle's control chain and integrated them into an end-to-end autonomous driving pipeline. Table 1 summarizes the composition of the 39 chains within the system, spanning across multiple sensing modalities and targeting either the system monitor or the command converter. In this setting,

Table 1
(Case study) Summary of the 39 cause–effect chains analyzed in the case study.

Source task	Target task	# chains
Lidar sensing	system monitor	6
	command converter	7
Radar sensing	system monitor	1
	command converter	1
Camera sensing	system monitor	4
	command converter	6
GNSS sensing	system monitor	3
	command converter	4
IMU sensing	system monitor	3
	command converter	4
Total		39

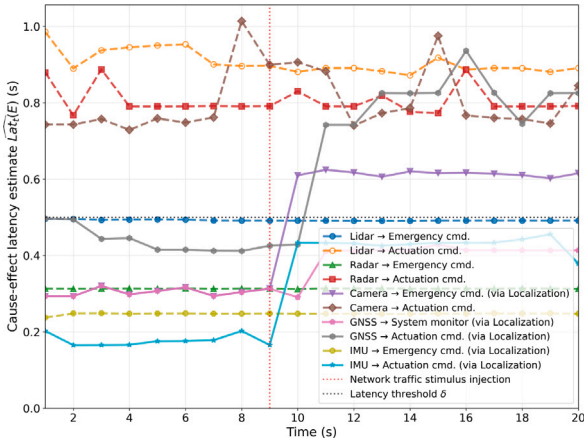


Fig. 15. (Case study) Cause–effect latency estimates of 10 chains when server overload is injected into the target autonomous driving system.

our role was to verify, from a system-wide perspective, whether the resulting cause–effect latency across the integrated chain satisfies safety requirements, and to provide feedback to the participating partners to support subsequent development milestones. Our mission was concretized as verifying, for all 39 chains, whether the cause–effect latency estimate satisfies $\widehat{\mathrm{Lat}}_t(E) \leq 0.5$ s. Based on this mission, we share three key insights from our case study: wide applicability, low runtime overhead, and limitations with opportunities for extension.

Wide applicability of black-box approach with minimal engineering effort. Fig. 15 shows time-series snippets of cause–effect latency estimates for ten arbitrary chains, one from each category in Table 1. At runtime, the estimation and verification algorithms checked whether the latencies satisfied the safety threshold (e.g., 0.5 s) every second. To analyze the robustness of system latency, we injected artificial server overload, as the localization task was known to be sensitive to increased system load. The results revealed a sharp latency increase in chains involving localization, which are highlighted using solid line styles in the figure, while other chains remained unaffected, demonstrating that our black-box approach can capture critical system dynamics with minimal engineering effort.

This experience shows that our method goes beyond verifying latency thresholds, supporting diverse engineering tasks at minimal cost. By observing only output events — without heavy instrumentation — we enabled fast, low-effort latency reporting. Although over-estimated, these reports provided actionable evidence that accelerated team discussions, aided stimulus-injection testing, and hinted at bottlenecks across chains, such as the following examples from our practical experience: First, by rapidly identifying chains likely to violate latency

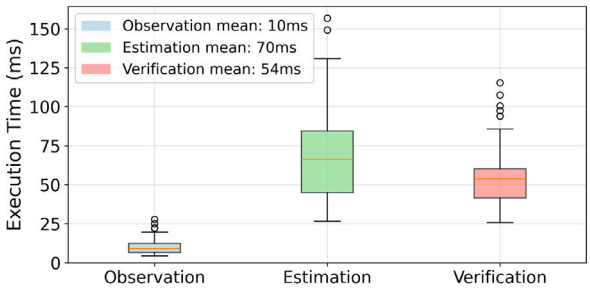


Fig. 16. (Case study) Runtime overhead of task write-event observation, job-chain length estimation, and cause–effect latency verification for 39 chains running in the autonomous driving system.

requirements based on conservative over-estimates, the development team was able to narrow down the subset of partners requiring coordination before conducting costly follow-up tests (e.g., field tests). Second, this enabled the application of high-fidelity, higher-effort runtime latency profiling techniques, such as instrumentation-based or trace-level methods, only to the identified critical modules instead of system-wide application. Third, for partners associated with chains exhibiting significant discrepancies between expectations derived from unit-level latency test results and the observed end-to-end runtime latency, our reports motivated the refinement and tightening of unit-level latency testing criteria in subsequent development iterations. In addition to the above experiences, our black-box approach could also extend to regression testing under updates, workload profiling, or runtime anomaly detection.

Low runtime overhead and suitability for external verification infrastructure. Fig. 16 reports the runtime overhead of write-event observation, job-chain length estimation, and cause–effect latency verification when applied to all 39 chains in the autonomous driving system. Each second of data was accumulated and then processed in a single batch, yielding mean execution times of 10 ms for observation, 70 ms for estimation, and 54 ms for verification. These values are larger than those reported in RQ4, since they reflect execution across all 39 chains simultaneously and include additional system costs such as logging and database access, rather than just the raw algorithm runtime. Nevertheless, the combined overhead remains lightweight, enabling real-time analysis of many chains.

This confirms that our approach is well-suited for runtime use even in complex industrial systems. Moreover, because the algorithms rely solely on externally observable write-events, the verification infrastructure can be deployed as a separate, non-intrusive component [16,17]. This separation minimizes deployment complexity, avoids interference with the system's internals, and allows verification to be integrated flexibly into existing operational environments.

Limitations and domain-specific extensions. Our algorithms are domain-agnostic, targeting situations where only write-events are accessible. This ensures broad applicability across black-box systems, but it also leaves room for further improvement when domain-specific information is available.

On the algorithmic side, extensions could address heterogeneous observability, where some tasks expose full white-box information while others remain black-box. Similarly, integrating partial knowledge such as job utilization profiles, execution times, task priorities, or additional system models could further tighten estimation bounds and improve the runtime verification under partial observability [18,19]. For the verification algorithm, a promising extension is proactive verification, which considers not only past samples but also forecasts future chain lengths using learning techniques [20–22]. This would enable early warning of potential requirement violations by detecting trends before thresholds are breached.

On the domain-specific side, our approach could be further enhanced by leveraging platform-specific support. For example, in robotics platforms using ROS 2, tracing package can capture both read- and write-event timestamps for black-box nodes implemented in C++, whereas Python nodes offer more limited visibility. In such cases, our methods could be combined with richer event logs to reduce estimation error and improve verification accuracy. These extensions highlight that while the core algorithms remain lightweight and general, tailoring them to specific domains can unlock stronger guarantees and broader applicability. Across all such extensions, our proposal provides a fundamental basis with formally proven bounds, ensuring that improvements build on a sound and reliable core.

9. Related work on cause–effect latency analysis

9.1. Formal timing analysis approaches

Substantial research on cause–effect chain latency has focused on formal modeling and worst-case analysis at design time. Many works provide analytical frameworks to compute these latencies under different assumptions and task models. For example, Günzel et al. unify various definitions and concepts of cause–effect latency and provide fundamental properties of latency analysis [1]. They also provide computational proofs of end-to-end latency bounds in chains of periodic tasks [3], while Dürr et al. present a formal analysis of chains of sporadic tasks in distributed systems [2]. Martínez et al. explore trade-offs among three communication models (e.g., implicit communication) for cause–effect chains [4], and Abdullah et al. propose an efficient algorithm for latency computation in systems with non-blocking communication [23]. Beyond task models, Coll-Perales et al. study system-level and network effects through modeling and simulation of end-to-end latencies [24]. In addition, formal verification techniques such as model checking have been applied to compute precise worst-case response times in embedded and distributed systems [5,25].

Overall, these formal approaches provide a strong theoretical foundation for reasoning about end-to-end latencies, and our work similarly builds on a formal system model, aligned with many of these approaches [3–5,25], to support job chain estimation and its theoretical analysis. However, these approaches typically aim to compute precise or worst-case latency bounds by relying on given design-time system specifications, which are often unavailable in practice and thus limit their applicability to black-box systems at runtime. In contrast, our work follows a runtime-based perspective rather than a formal design-time approach, as introduced in the following subsection.

9.2. System monitoring and measurement approaches

Beyond design-time analysis, researchers have developed methods to measure and analyze the end-to-end latency of chains in deployed systems across specific domains.

In the automotive domain, Becker et al. present methods to compute end-to-end delays based on different levels of system information in embedded systems [8]. Houtan et al. extract timing models from TSN-aware distributed vehicle software [26], and Coll-Perales et al. study system- and network-level effects on end-to-end latency in vehicular networks [24]. Schlatow et al. address latency analysis for multi-rate distributed cause–effect chains [27], while Li et al. automate trace point insertion for latency measurement in the Autoware autonomous driving platform [28].

In robotics, where ROS and ROS 2 are dominant middleware frameworks, several runtime analysis techniques have been proposed. Betz et al. measure both end-to-end latency and its constituent components by extracting implicit and explicit data flows directly from source code [29]. Abaza et al. propose a trace-enabled model synthesis framework that automatically reconstructs DAGs of callbacks from middleware and OS traces [6]. Teper et al. compute end-to-end latency for the

ROS 2 native scheduler [30], while Betz et al. optimize callback chains of containerized ROS 2 autonomous driving software [7]. Latency evaluation workflows across the application, middleware (ROS 2, DDS), OS, and hardware layers have also been presented [31].

These runtime methods provide ground-truth latency values and can capture unexpected delays caused by runtime uncertainties. However, they often require substantial system access (e.g., white-box knowledge or probe insertion), which is not always feasible in practice. In contrast, our algorithms are domain-agnostic, relying only on observable write-events; to the best of our knowledge, no prior work has pursued this objective, making our approach broadly applicable while offering complementary lightweight verification across diverse domains under the considered system semantics (e.g., periodic tasks with implicit communication).

10. Conclusion

In this paper, we addressed the problem of verifying end-to-end cause–effect latency in black-box systems, where only task write-events are observable. We formally defined this problem and proposed two lightweight algorithms: a job-chain estimation algorithm that safely over-estimates chain lengths, and a verification algorithm that avoids false positives.

We proved theoretical error bounds for over-estimation under the considered system model and validated them experimentally. Our results showed the estimation error inherent to limited observability, which should be considered but can be significantly reduced when chain utilization is high. We also found that verification remains trustworthy with low false positives under the prescribed confidence level, and that both algorithms incur negligible runtime cost. An industrial application on an autonomous driving system demonstrated the practicality of our approach, successfully verifying 39 chains with minimal engineering effort and low overhead, and confirming its effectiveness under real-world constraints.

Future directions include extending our approach with richer domain-specific black-box observables and applying it to broader industrial cases.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Yong-Jun Shin reports financial support was provided by Institute of Information & Communications Technology Planning & Evaluation. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00406245).

Data availability

Data will be made available on request.

References

[1] M. Günzel, H. Teper, G.v.d. Brüggen, J.-J. Chen, End-to-end latency of cause-effect chains: A tutorial, ACM Trans. Embed. Comput. Syst. 24 (1) (2024) 1–18.

[2] M. Dürr, G.V.D. Brüggen, K.-H. Chen, J.-J. Chen, End-to-end timing analysis of sporadic cause-effect chains in distributed systems, ACM Trans. Embed. Comput. Syst. 18 (5s) (2019).

[3] M. Günzel, K.-H. Chen, N. Ueter, G. von der Brüggen, M. Dürr, J.-J. Chen, Timing analysis of asynchronized distributed cause-effect chains, in: 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium, RTAS, IEEE, 2021, pp. 40–52.

[4] J. Martinez, I. Sañudo, M. Bertogna, End-to-end latency characterization of task communication models for automotive systems, Real-Time Syst. 56 (3) (2020) 315–347.

[5] A. Rajeev, S. Mohalik, M.G. Dixit, D.B. Chokshi, S. Ramesh, Schedulability and end-to-end latency in distributed ecu networks: Formal modeling and precise estimation, in: Proceedings of the Tenth ACM International Conference on Embedded Software, 2010, pp. 129–138.

[6] H. Abaza, D. Roy, S. Fan, S. Saidi, A. Motakis, Trace-enabled timing model synthesis for ROS2-based autonomous applications, in: 2024 Design, Automation & Test in Europe Conference & Exhibition, DATE, IEEE, 2024, pp. 1–6.

[7] T. Betz, H. Teper, D. Ebner, M. Leitenstern, S. Sagmeister, M. Weinmann, J.-J. Chen, M. Lienkamp, End-to-end latency optimization for containerized ROS 2 autonomous driving software, IEEE Access (2025).

[8] M. Becker, D. Dasari, S. Mubeen, M. Behnam, T. Nolte, End-to-end timing analysis of cause-effect chains in automotive embedded systems, J. Syst. Archit. 80 (2017) 104–113.

[9] T. Betz, L. Wen, F. Pan, G. Kaljavesi, A. Zuepke, A. Bastoni, M. Caccamo, A. Knoll, J. Betz, A containerized microservice architecture for a ROS 2 autonomous driving software: An end-to-end latency evaluation, in: 2024 IEEE 30th International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA, IEEE, 2024, pp. 57–66.

[10] M. Günzel, M. Becker, Optimal task phasing for end-to-end latency in harmonic and semi-harmonic automotive systems, in: 2025 IEEE 31st Real-Time and Embedded Technology and Applications Symposium, RTAS, IEEE, 2025, pp. 164–176.

[11] B.P. Welford, Note on a method for calculating corrected sums of squares and products, Technometrics 4 (3) (1962) 419–420.

[12] W.G. Howe, Two-sided tolerance limits for normal populations—Some improvements, J. Amer. Statist. Assoc. 64 (328) (1969) 610–620.

[13] R.P. Dick, D.L. Rhodes, W. Wolf, TGFF: task graphs for free, in: Proceedings of the Sixth International Workshop on Hardware/Software Codesign, CODES/CASHE'98, IEEE, 1998, pp. 97–101.

[14] M. Neukirchner, S. Stein, R. Ernst, Smff: System models for free, WATERS 2011, in: 2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems, vol. 6, 2011.

[15] A. Hamann, D. Dasari, S. Kramer, M. Pressler, F. Wurst, D. Ziegenbein, Waters industrial challenge 2017, in: International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems, WATERS, 2017.

[16] J. Mertz, I. Nunes, Software runtime monitoring with adaptive sampling rate to collect representative samples of execution traces, J. Syst. Softw. 202 (2023) 111708.

[17] L. Bulej, T. Bureš, A. Filandr, P. Hnětynka, I. Hnětynková, J. Pacovskỳ, G. Sandor, I. Gerostathopoulos, Managing latency in edge–cloud environment, J. Syst. Softw. 172 (2021) 110872.

[18] S. Pinisetty, T. Jéron, S. Tripakis, Y. Falcone, H. Marchand, V. Preoteasa, Predictive runtime verification of timed properties, J. Syst. Softw. 132 (2017) 353–365.

[19] A. Cimatti, C. Tian, S. Tonetta, Assumption-based runtime verification with partial observability and resets, in: International Conference on Runtime Verification, Springer, 2019, pp. 165–184.

[20] R. Babaee, A. Gurfinkel, S. Fischmeister, Prevent: a predictive run-time verification framework using statistical learning, in: International Conference on Software Engineering and Formal Methods, Springer, 2018, pp. 205–220.

[21] R. Babaee, A. Gurfinkel, S. Fischmeister, Predictive run-time verification of discrete-time reachability properties in black-box systems using trace-level abstraction and statistical learning, in: International Conference on Runtime Verification, Springer, 2018, pp. 187–204.

[22] Z. Ge, J. Hou, A. Nayak, Forecasting SDN end-to-end latency using graph neural network, in: 2023 International Conference on Information Networking, ICOIN, IEEE, 2023, pp. 293–298.

[23] J. Abdullah, G. Dai, W. Yi, Worst-case cause-effect reaction latency in systems with non-blocking communication, in: 2019 Design, Automation & Test in Europe Conference & Exhibition, DATE, IEEE, 2019, pp. 1625–1630.

[24] B. Coll-Perales, M.C. Lucas-Estañ, T. Shimizu, J. Gozalvez, T. Higuchi, S. Avedisov, O. Altintas, M. Sepulcre, End-to-end V2X latency modeling and analysis in 5G networks, IEEE Trans. Veh. Technol. 72 (4) (2022) 5094–5109.

[25] S. Mohalik, A. Rajeev, M.G. Dixit, S. Ramesh, P.V. Suman, P.K. Pandya, S. Jiang, Model checking based analysis of end-to-end latency in embedded, real-time systems with clock drifts, in: Proceedings of the 45th Annual Design Automation Conference, 2008, pp. 296–299.
[26] B. Houtan, M.O. Aybek, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Mubeen, End-to-end timing model extraction from TSN-aware distributed vehicle software, in: 2022 48th Euromicro Conference on Software Engineering and Advanced Applications, SEAA, IEEE, 2022, pp. 366–369.
[27] J. Schlatow, M. Mostl, S. Tobuschat, T. Ishigooka, R. Ernst, Data-age analysis and optimisation for cause-effect chains in automotive control systems, in: 2018 IEEE 13th International Symposium on Industrial Embedded Systems, SIES, IEEE, 2018, pp. 1–9.
[28] Z. Li, A. Hasegawa, T. Azumi, Autoware_perf: A tracing and performance analysis framework for ROS 2 applications, J. Syst. Archit. 123 (2022) 102341.
[29] T. Betz, M. Schmeller, A. Korb, J. Betz, Latency measurement for autonomous driving software using data flow extraction, in: 2023 IEEE Intelligent Vehicles Symposium, IV, IEEE, 2023, pp. 1–8.
[30] H. Teper, O. Bell, M. Günzel, C. Gill, J.-J. Chen, Reconciling ROS 2 with classical real-time scheduling of periodic tasks, in: 2025 IEEE 31st Real-Time and Embedded Technology and Applications Symposium, RTAS, IEEE, 2025, pp. 177–189.
[31] T. Betz, M. Schmeller, H. Teper, J. Betz, How fast is my software? latency evaluation for a ros 2 autonomous driving software, in: 2023 IEEE Intelligent Vehicles Symposium, IV, IEEE, 2023, pp. 1–6.