Approximate Distributed Monitoring Under Partial Synchrony: Balancing Speed & Accuracy

Borzoo Bonakdarpour[1], Anik Momtaz[1], Dejan Ničković[2], and N. Ege Saraç[3](✉)

[1] Michigan State University, East Lansing, USA
{borzoo,momtazan}@msu.edu
[2] AIT Austrian Institute of Technology, Wien, Austria
dejan.nickovic@ait.ac.at
[3] Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria
esarac@ista.ac.at

Abstract. In distributed systems with processes that do not share a global clock, *partial synchrony* is achieved by clock synchronization that guarantees bounded clock skew among all applications. Existing solutions for distributed runtime verification under partial synchrony against temporal logic specifications are exact but suffer from significant computational overhead. In this paper, we propose an *approximate* distributed monitoring algorithm for Signal Temporal Logic (STL) that mitigates this issue by abstracting away potential interleaving behaviors. This conservative abstraction enables a significant speedup of the distributed monitors, albeit with a tradeoff in accuracy. We address this tradeoff with a methodology that combines our approximate monitor with its exact counterpart, resulting in enhanced efficiency without sacrificing precision. We evaluate our approach with multiple experiments, showcasing its efficacy in both real-world applications and synthetic examples.

Keywords: distributed systems · approximate monitoring · partial synchrony

1 Introduction

Distributed systems are networks of independent agents that work together to achieve a common objective. They come in many different forms. For example, cloud computing uses distribution of resources and services over the internet to offer to their users a scalable infrastructure with transparent on-demand access to computing power and storage. Swarms of drones is another family of distributed systems where individual drones collaborate to accomplish tasks like search and rescue or package delivery. While each drone operates independently, it also communicates and coordinates with others to successfully achieve their

This work was supported in part by the ERC-2020-AdG 101020093. This work is ponsored in part by the United States NSF CCF-2118356 award. This research was partially funded by A-IQ Ready (Chips JU, grant agreement No. 101096658).

E. Ábrahám and H. Abbas (Eds.): RV 2024, LNCS 15191, pp. 282–301, 2025.
https://doi.org/10.1007/978-3-031-74234-7_18

common objectives. The individual agents in a distributed system typically do not share a global clock. To coordinate actions across multiple agents, clock synchronization is often needed. While perfect clock synchronization is impractical due to network latency and node failures, algorithms such as the Network Time Protocol (NTP) allow agents to maintain a *bounded skew* between the synchronized clocks. We then say that a distributed system has *partial synchrony*.

Formal verification of distributed system is a notoriously hard problem, due to the combinatorial explosion of all possible interleavings in the behaviors collected from individual agents. *Runtime verification (RV)* provides a more pragmatic approach in which a behavior of a distributed system is observed and its correctness is checked against a formal specification. We consider the *distributed RV* setting where this task is performed by a single central monitor observing the independent agents (as opposed to *decentralized RV* where the monitoring task itself is distributed). Remotely related to the problem of distributed RV under partial synchrony are distributed RV in the fully *synchronous* [5,8,9] and *asynchronous* [6,7,12,17,19,21] settings as well as benchmarking tools [2] for assessing monitoring overhead. The problem of distributed RV under partial synchrony assumption has been studied for Linear Temporal Logic (LTL) [11] and Signal Temporal Logic (STL) [18] specification languages. The proposed solutions use Satisfiability-Modulo-Theory (SMT) solving to provide sound and complete distributed monitoring procedures. Although distributed RV monitors consume only a single distributed behavior at a time, this behavior can have an excessive number of possible interleavings. Put another way, although RV deals only with the verification of a single execution at run time, it is still prone to evaluating an explosion of combinations. Hence, the exact distributed monitors from the literature can still suffer from significant computational overhead. This phenomenon has been observed even under partial synchrony [10,11], and becomes problematic even for offline monitoring of a large set of log files.

To mitigate this issue, we propose a new approach for *approximate* RV of STL under partial synchrony. In essence, we conservatively abstract away potential interleavings in distributed behaviors, resulting in their overapproximation. This abstraction simplifies the representation of distributed behaviors into a set of Boolean expressions, taking into account regions of uncertainty created by clock skews. We define monitoring operations that evaluate temporal specifications over such expressions, which result in monitoring verdicts on overapproximated behaviors. This approximate solution yields an inevitable tradeoff between *accuracy* and *speedup*. For applications where reduced accuracy is not acceptable, we devise a methodology that combines approximate and exact monitors, with the aim to benefit from the enhanced efficiency without sacrificing precision. Approximate monitoring is also valuable in the sequential setting, with applications including monitoring with state estimation [4,23], quantitative monitoring and its resource-precision tradeoffs [13–15], and various other uses [1,3].

We implemented our approach in a prototype tool and performed thorough evaluations on both synthetic and real-world case studies (mutual separation in swarm of drones and a water distribution system). We first demonstrated that in many experiments, our approximate monitors achieve speedups of up to 5 orders of magnitude compared to the exact SMT-based solution. We empirically characterized the classes of specifications and behaviors for which our approximate monitors achieve good precision. We finally showed that combining exact and approximate distributed RV yields significant efficiency gains on average without sacrificing precision, even with low-accuracy approximate monitors.

2 Preliminaries

We denote by $\mathbb{B} = \{\top, \bot\}$ the set of Booleans, $\mathbb{R}$ the set of reals, $\mathbb{R}_{\geq 0}$ the set of nonnegative reals, and $\mathbb{R}_{>0}$ the set of positive reals. An interval $I \subseteq \mathbb{R}$ of reals with the end points $a < b$ has length $|b - a|$.

Let Σ be a finite *alphabet*. We denote by Σ^* the set of finite words over Σ and by ϵ the empty word. For $u \in \Sigma^*$, we respectively write $\mathsf{prefix}(u)$ and $\mathsf{suffix}(u)$ for the sets of prefixes and suffixes of u. We also let $\mathsf{infix}(u) = \{v \in \Sigma^* \mid \exists x, y \in \Sigma^* : u = xvy\}$. For a nonempty word $u \in \Sigma^*$ and $1 \leq i \leq |u|$, we denote by $u[i]$ the ith letter of u. Given $u \in \Sigma^*$ and $\ell \geq 1$, we denote by u^ℓ the word obtained by concatenating u by itself $\ell - 1$ times. Moreover, given $L \subseteq \Sigma^*$, we define $\mathsf{first}(L) = \{u[0] \mid u \in L\}$. For sets $L_1, L_2 \subseteq \Sigma^*$ of words, we let $L_1 \cdot L_2 = \{u \cdot v \mid u \in L_1, v \in L_2\}$. For tuples $(u_1, \ldots, u_m)$ and $(v_1, \ldots, v_m)$ of words, we let $(u_1, \ldots, u_m) \cdot (v_1, \ldots, v_m) = (u_1 v_1, \ldots, u_m v_m)$.

We define the function $\mathsf{destutter} : \Sigma^* \to \Sigma^*$ inductively. For all $\sigma \in \Sigma \cup \{\epsilon\}$, let $\mathsf{destutter}(\sigma) = \sigma$. For all $u \in \Sigma^*$ such that $u = \sigma_1 \sigma_2 v$ for some $\sigma_1, \sigma_2 \in \Sigma$ and $v \in \Sigma^*$, we define it as follows:

$$\mathsf{destutter}(u) = \begin{cases} \mathsf{destutter}(\sigma_2 v) & \text{if } \sigma_1 = \sigma_2 \\ \sigma_1 \cdot \mathsf{destutter}(\sigma_2 v) & \text{otherwise} \end{cases}$$

For a set $L \subseteq \Sigma^*$ of finite words, we define $\mathsf{destutter}(L) = \{\mathsf{destutter}(u) \mid u \in L\}$. We extend $\mathsf{destutter}$ to tuples of words in a synchronized manner: for all $\sigma \in \Sigma \cup \{\epsilon\}$ let $\mathsf{destutter}(\sigma, \ldots, \sigma) = (\sigma, \ldots, \sigma)$. Given a tuple $(u_1, \ldots, u_m) = (\sigma_{1,1}\sigma_{1,2}v_1, \ldots, \sigma_{m,1}\sigma_{m,2}v_m)$ of words of the same length, $\mathsf{destutter}(u_1, \ldots, u_m)$ is defined as expected:

$$\mathsf{destutter}(u_1, \ldots, u_m) = \begin{cases} \mathsf{destutter}(\sigma_{1,2}v_1, \ldots, \sigma_{m,2}v_m) \text{ if } \sigma_{i,1} = \sigma_{i,2} \text{ for all } 1 \leq i \leq m \\ (\sigma_{1,1}, \ldots, \sigma_{m,1}) \cdot \mathsf{destutter}(\sigma_{1,2}v_1, \ldots, \sigma_{m,2}v_m) \text{ otherwise} \end{cases}$$

Moreover, given an integer $k \geq 0$, we define $\mathsf{stutter}_k : \Sigma^* \to \Sigma^*$ such that $\mathsf{stutter}_k(u) = \{v \in \Sigma^* \mid |v| = k \wedge \mathsf{destutter}(v) = \mathsf{destutter}(u)\}$ if $k \geq |\mathsf{destutter}(u)|$, and $\mathsf{stutter}_k(u) = \emptyset$ otherwise.

Signal Temporal Logic (STL) [16]**.** Let $A, B \subset \mathbb{R}$. A function $f : A \to B$ is *right-continuous* iff $\lim_{a \to c^+} f(a) = f(c)$ for all $c \in A$, and *non-Zeno* iff for every bounded interval $I \subseteq A$ there are finitely many $a \in I$ such that f is not continuous at a. A *signal* is a right-continuous, non-Zeno, piecewise-constant function $x : [0, d) \to \mathbb{R}$ where $d \in \mathbb{R}_{>0}$ is the duration of x and $[0, d)$ is its temporal domain. Let $x : [0, d) \to \mathbb{R}$ be a signal. An *event* of x is a pair $(t, x(t))$

where $t \in [0, d)$. An *edge* of x is an event $(t, x(t))$ such that $\lim_{s \to t^-} x(s) \neq \lim_{s \to t^+} x(s)$. In particular, an edge is *rising* if $\lim_{s \to t^-} x(s) < \lim_{s \to t^+} x(s)$, and it is *falling* otherwise. A signal $x : [0, d) \to \mathbb{R}$ can be represented finitely by its initial value and edges: if x has m edges, then $x = (t_0, v_0)(t_1, v_1) \dots (t_m, v_m)$ such that $t_0 = 0$, $t_{i-1} < t_i$, and (t_i, v_i) is an edge of x for all $1 \leq i \leq m$.

Let AP be a set of *atomic propositions*. The syntax of STL is given by the grammar $\varphi := p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \mathcal{U}_I \varphi$ where $p \in \mathsf{AP}$ and $I \subseteq \mathbb{R}_{\geq 0}$ is an interval.

A *trace* $w = (x_1, \dots, x_n)$ is a finite vector of signals. We express atomic propositions as functions of trace values at a time point t, i.e., a proposition $p \in \mathsf{AP}$ over a trace $w = (x_1, \dots, x_n)$ is defined as $f_p(x_1(t), \dots, x_n(t)) > 0$ where $f_p : \mathbb{R}^n \to \mathbb{R}$ is a function. Given intervals $I, J \subseteq \mathbb{R}_{\geq 0}$, we define $I \oplus J = \{i + j \mid i \in I, j \in J\}$, and we simply write t for the singleton set $\{t\}$.

We recall the finite-trace qualitative semantics of STL defined over $\mathbb{B}$. Let $d \in \mathbb{R}_{>0}$ and $w = (x_1, \dots, x_n)$ with $x_i : [0, d) \to \mathbb{R}$ for all $1 \leq i \leq n$. Let φ_1, φ_2 be STL formulas and let $t \in [0, d)$.

$$\begin{aligned}
(w, t) \models p &\iff f_p(x_1(t), \dots, x_n(t)) > 0 \\
(w, t) \models \neg\varphi_1 &\iff \overline{(w, t) \models \varphi_1} \\
(w, t) \models \varphi_1 \wedge \varphi_2 &\iff (w, t) \models \varphi_1 \wedge (w, t) \models \varphi_2 \\
(w, t) \models \varphi_1 \mathcal{U}_I \varphi_2 &\iff \exists t' \in (t \oplus I) \cap [0, d) : \\
&\qquad (w, t') \models \varphi_2 \wedge \forall t'' \in (t, t') : (w, t'') \models \varphi_1
\end{aligned}$$

We simply write $w \models \varphi$ for $(w, 0) \models \varphi$. We additionally use the following standard abbreviations: $\texttt{false} = p \wedge \neg p$, $\texttt{true} = \neg\texttt{false}$, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\Diamond_I \varphi = \texttt{true}\mathcal{U}_I \varphi$, and $\Box_I \varphi = \neg \Diamond_I \neg\varphi$. Moreover, the untimed temporal operators are defined through their timed counterparts on the interval $[0, \infty)$.

Distributed Semantics of STL [18]**.** We consider an asynchronous and loosely-coupled message-passing system of $n \geq 2$ reliable agents producing a set of signals $x_1, \dots, x_n$, where for some $d \in \mathbb{R}_{>0}$ we have $x_i : [0, d) \to \mathbb{R}$ for all $1 \leq i \leq n$. The agents do not share memory or a global clock. Only to formalize statements, we speak of a *hypothetical* global clock and denote its value by T. For local time values, we use the lowercase letters t and s. For a signal x_i, we denote by V_i the set of its events, and by E_i the set of its edges. We represent the local clock of the ith agent as an increasing and divergent function $c_i : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ that maps a global time T to a local time $c_i(T)$.

We assume that the system is *partially synchronous*: the agents use a clock synchronization algorithm that guarantees a bounded clock skew with respect to the global clock, i.e., $|c_i(T) - c_j(T)| < \varepsilon$ for all $1 \leq i, j \leq N$ and $T \in \mathbb{R}_{\geq 0}$, where $\varepsilon \in \mathbb{R}_{>0}$ is the maximum clock skew.

Definition 1. *A* distributed signal *is a pair* $(S, \rightsquigarrow)$*, where* $S = (x_1, \dots, x_n)$ *is a vector of signals and* $\rightsquigarrow$ *is the happened-before relation between events defined as follows: (1) For every agent, the events of its signals are totally ordered, i.e., for all* $1 \leq i \leq n$ *and all* $(t, x_i(t)), (t', x_i(t')) \in V_i$*, if* $t < t'$ *then* $(t, x_i(t)) \rightsquigarrow$

$(t', x_i(t'))$. (2) Every pair of events whose timestamps are at least ε apart is totally ordered, i.e., for all $1 \leq i, j \leq n$ and all $(t, x_i(t)) \in V_i$ and $(t', x_j(t')) \in V_j$, if $t + \varepsilon \leq t'$ then $(t, x_i(t)) \rightsquigarrow (t', x_j(t'))$.

The notion of *consistent cut* captures possible global states.

Definition 2. *Let $(S, \rightsquigarrow)$ be a distributed signal of n signals, and $V = \bigcup_{i=1}^{n} V_i$ be the set of its events. A set $C \subseteq V$ is a* consistent cut *iff for every event in C, all events that happened before it also belong to C, i.e., for all $e, e' \in V$, if $e \in C$ and $e' \rightsquigarrow e$, then $e' \in C$.*

We denote by $\mathbb{C}(T)$ the set of consistent cuts at global time T. Given a consistent cut C, its *frontier* $\mathsf{fr}(C) \subseteq C$ is the set consisting of the last events in C of each signal, i.e., $\mathsf{fr}(C) = \bigcup_{i=1}^{n}\{(t, x_i(t)) \in V_i \cap C \mid \forall t' > t : (t', x_i(t')) \notin V_i \cap C\}$.

Definition 3. *A* consistent cut flow *is a function $\mathsf{ccf} : \mathbb{R}_{\geq 0} \to 2^V$ that maps a global clock value T to the frontier of a consistent cut at time T, i.e., $\mathsf{ccf}(T) \in \{\mathsf{fr}(C) \mid C \in \mathbb{C}(T)\}$.*

For all $T, T' \in \mathbb{R}_{\geq 0}$ and $1 \leq i \leq n$, if $T < T'$, then for every pair of events $(c_i(T), x_i(c_i(T))) \in \mathsf{ccf}(T)$ and $(c_i(T'), x_i(c_i(T'))) \in \mathsf{ccf}(T')$ we have $(c_i(T), x_i(c_i(T))) \rightsquigarrow (c_i(T'), x_i(c_i(T')))$. We denote by $\mathsf{CCF}(S, \rightsquigarrow)$ the set of all consistent cut flows of the distributed signal $(S, \rightsquigarrow)$.

Observe that a consistent cut flow of a distributed signal induces a vector of synchronous signals which can be evaluated using the standard STL semantics described above. Let $(S, \rightsquigarrow)$ be a distributed signal of n signals $x_1, \ldots, x_n$. A consistent cut flow $\mathsf{ccf} \in \mathsf{CCF}(S, \rightsquigarrow)$ yields a trace $w_{\mathsf{ccf}} = (x'_1, \ldots x'_n)$ on the temporal domain $[0, d)$ such that $(c_i(T), x_i(c_i(T))) \in \mathsf{ccf}(T)$ implies $x'_i(T) = x_i(c_i(T))$ for all $1 \leq i \leq n$ and $T \in [0, d)$. The set of traces of $(S, \rightsquigarrow)$ is given by $\mathsf{Tr}(S, \rightsquigarrow) = \{w_{\mathsf{ccf}} \mid \mathsf{ccf} \in \mathsf{CCF}(S, \rightsquigarrow)\}$ (Fig. 1).

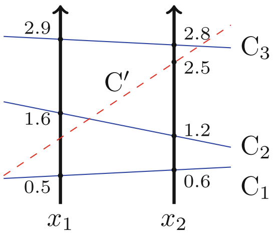

Fig. 1. A distributed signal in with consistent cuts C_1, C_2, C_3 constituting a consistent cut flow. Note that C' is a non-example since $(2.5, x_2(2.5)) \in \mathsf{fr}(C')$ and $(1.6, x_1(1.6)) \notin \mathsf{fr}(C')$, but $(1.6, x_1(1.6))$ happened before $(2.5, x_2(2.5))$.

We define the satisfaction of an STL formula φ by a distributed signal $(S, \rightsquigarrow)$ over a three-valued domain $\{\top, \bot, ?\}$ Notice that we quantify universally over traces for both satisfaction and violation.

$$[(S, \rightsquigarrow) \models \varphi] = \begin{cases} \top & \text{if } \forall w \in \mathsf{Tr}(S, \rightsquigarrow) : w \models \varphi \\ \bot & \text{if } \forall w \in \mathsf{Tr}(S, \rightsquigarrow) : w \models \neg\varphi \\ ? & \text{otherwise} \end{cases}$$

3 Overapproximation of the STL Distributed Semantics

To address the computational overhead in exact distributed monitoring, we define STL$^+$, a variant of STL whose syntax is the same as STL but semantics provide a sound approximation of the STL distributed semantics. In particular, given a distributed signal $(S, \rightsquigarrow)$, STL$^+$ considers an approximation $\mathsf{Tr}^+(S, \rightsquigarrow)$ of the set $\mathsf{Tr}(S, \rightsquigarrow)$ of synchronous traces where $\mathsf{Tr}(S, \rightsquigarrow) \subseteq \mathsf{Tr}^+(S, \rightsquigarrow)$. A signal $(S, \rightsquigarrow)$ satisfies (resp. violates) an STL$^+$ formula φ iff all the traces in $\mathsf{Tr}^+(S, \rightsquigarrow)$ belong to the language of φ (resp. $\neg\varphi$).

$$[(S, \rightsquigarrow) \models \varphi]_+ = \begin{cases} \top & \text{if } \forall w \in \mathsf{Tr}^+(S, \rightsquigarrow) : w \models \varphi \\ \bot & \text{if } \forall w \in \mathsf{Tr}^+(S, \rightsquigarrow) : w \models \neg\varphi \\ ? & \text{otherwise} \end{cases}$$

Throughout the paper, we assume φ is *copyless*, i.e., each signal $x \in S$ occurs in φ at most once. Moreover, the signals are Boolean, non-Zeno, piecewise-constant, and have no edge at time 0. We assume Boolean signals only for convenience; all the concepts and results generalize to non-Boolean signals because finite-length piecewise-constant signals use only a finite number of values. We note that our approach is a sound overapproximation also for non-copyless formulas, although potentially less precise. In Sects. 4 and 5, we respectively define Tr^+ and present an algorithm to compute the semantics of STL$^+$.

Theorem 1. *For every STL formula φ and every distributed signal $(S, \rightsquigarrow)$, if $[(S, \rightsquigarrow) \models \varphi]_+ = \top$ (resp. $\bot$) then $[(S, \rightsquigarrow) \models \varphi] = \top$ (resp. $\bot$).*

Notice that both the distributed semantics of STL and the semantics of STL$^+$ quantify universally over the set of traces for the verdicts $\top$ and $\bot$. Therefore, Theorem 1 holds for the verdicts $\top$ and $\bot$, but not for ?.

4 Overapproximation of Synchronous Traces

In this section, given a distributed signal $(S, \rightsquigarrow)$, we describe an overapproximation $\mathsf{Tr}^+(S, \rightsquigarrow)$ of its set $\mathsf{Tr}(S, \rightsquigarrow)$ of synchronous traces. First, we present the notion of *canonical segmentation*, a systematic way of partitioning the temporal domain of a distributed signal to track partial synchrony. Second, we introduce *value expressions*, sets of finite words representing signal behavior in a time interval. Finally, we define Tr^+ and show that it soundly approximates Tr.

Canonical Segmentation. Consider a Boolean signal x with a rising edge at time $t > \varepsilon$. Due to clock skew, this edge occurs in the range $(t - \varepsilon, t + \varepsilon)$ from the monitor's perspective. This range is an *uncertainty region* because within it, the monitor can only tell that x changes from 0 to 1. Formally, given an edge $(t, x(t))$, we define $\theta_{\text{lo}}(x, t) = \max(0, t - \varepsilon)$ and $\theta_{\text{hi}}(x, t) = \min(d, t + \varepsilon)$ as the endpoints of the edge's uncertainty region.

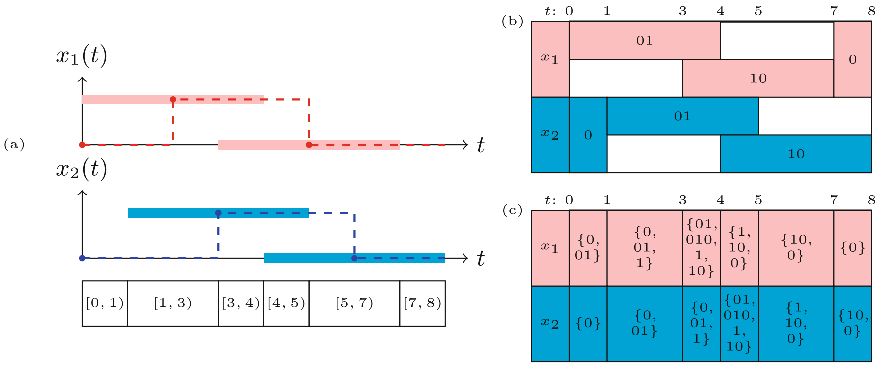

Fig. 2. (a) A distributed signal $(S, \rightsquigarrow)$ with x_1 (top, red) and x_2 (bottom, blue) whose edges are marked with solid balls and their uncertainty regions are given as semi-transparent boxes around the edges. The resulting canonical segmentation G_S is shown below the graphical representation of the signals. (b) The uncertainty regions of $(S, \rightsquigarrow)$ and the corresponding value expressions. (c) The tabular representation of the function γ for $(S, \rightsquigarrow)$, e.g., $\gamma(x_1, [3, 4)) = (\mathsf{suffix}(01) \cdot \mathsf{prefix}(10)) \setminus \{\epsilon\} = \{01, 010, 1, 10\}$. (Color figure online)

Given a temporal domain $I = [0, d) \subset \mathbb{R}_{\geq 0}$, a *segmentation* of I is a partition of I into finitely many intervals $I_1, \ldots, I_k$, called *segments*, of the form $I_j = [t_j, t_{j+1})$ such that $t_j < t_{j+1}$ for all $1 \leq j \leq k$. By extension, a segmentation of a collection of signals with the same temporal domain I is a segmentation of I.

Let $(S, \rightsquigarrow)$ be a distributed signal of n signals. The *canonical segmentation* G_S of $(S, \rightsquigarrow)$ the segmentation of S where the segment endpoints match the temporal domain and uncertainty region endpoints. Formally, we define G_S as follows. For each signal x_i, where $1 \leq i \leq n$, let F_i be the set of uncertainty region endpoints. Let $F = \{0, d\} \cup \bigcup_{i=1}^{n} F_i$ and let $(s_j)_{1 \leq j \leq |F|}$ be a nondecreasing sequence of clock values corresponding to the elements of F. Then, the canonical segmentation of $(S, \rightsquigarrow)$ is $G_S = \{I_1, \ldots, I_{|F|-1}\}$ where $I_j = [s_j, s_{j+1})$ for all $1 \leq j < |F|$. We show an example in Fig. 2a.

Value Expressions. Consider a Boolean signal x with a rising edge within an uncertainty region of (t_1, t_2). As mentioned, the monitor only knows that x changes from 0 to 1 in this interval. This knowledge is represented as a finite word $v = 01$ over the alphabet $\Sigma = \{0, 1\}$. This representation, called a *value expression*, encodes the uncertain behavior of an observed signal relative to the monitor. Formally, a value expression is an element of Σ^*, where Σ is the finite alphabet of signal values. Given a signal x and an edge $(t, x(t))$, the value expression corresponding to the uncertainty region $(\theta_{\text{lo}}(x, t), \theta_{\text{hi}}(x, t))$ is $v_{x,t} = v_- \cdot v_+$, where $v_- = \lim_{s \to t^-} x(s)$ and $v_+ = \lim_{s \to t^+} x(s)$. We omit the concatenation symbol $\cdot$ when the letters are clear from context. This definition is general because finite-length piecewise-constant real-valued signals will only have a finite number of values, making Σ finite.

Notice that (i) uncertainty regions may overlap, and (ii) the canonical segmentation may split an uncertainty region into multiple segments. Consider a signal x with a rising edge in $(1, 5)$ and a falling edge in $(4, 8)$. The corresponding value expressions are respectively $v_1 = 01$ and $v_2 = 10$. Notice that the behavior of x in the interval $[1, 4)$ can be expressed as $\mathsf{prefix}(v_1)$, encoding whether the rising edge has happened yet. Similarly, the behavior in $[4, 5)$ is given by $\mathsf{suffix}(v_1) \cdot \mathsf{prefix}(v_2)$, which captures whether the edges occur in this interval (thanks to prefixing and suffixing) and the fact that the rising edge happens before the falling edge (thanks to concatenation).

Formally, given a distributed signal $(S, \rightsquigarrow)$, we define a function $\gamma : S \times G_S \to 2^{\Sigma^*}$ that maps each signal and segment of the canonical segmentation to a set of value expressions, capturing the signal's potential behaviors in the given segment. Let x be a signal in S, and let $R_1, \ldots, R_m$ be its uncertainty regions where $R_i = (t_i, t'_i)$ and the corresponding value expression is v_i for all $1 \leq i \leq m$. Now, let $I \in G_S$ be a segment with $I = [s, s')$ and for each $1 \leq i \leq m$ define the set V_i of value expressions capturing how I relates with R_i in Eq. (1).

$$V_i = \begin{cases} \{v_i\} & \text{if } t_i = s \wedge s' = t'_i \\ \mathsf{prefix}(v_i) & \text{if } t_i = s \wedge s' < t'_i \\ \mathsf{suffix}(v_i) & \text{if } t_i > s \wedge s' = t'_i \\ \mathsf{infix}(v_i) & \text{if } t_i > s \wedge s' < t'_i \\ \{\epsilon\} & \text{otherwise} \end{cases} \tag{1}$$

The last case happens only when $I \cap R_i$ is empty. We define γ as follows:

$$\gamma(x, I) = \mathsf{destutter}(V_1 \cdot V_2 \cdot \ldots \cdot V_m) \setminus \{\epsilon\}$$

Observe that $\gamma(x, I)$ contains all the potential behaviors of x in segment I by construction. However, it is potentially overapproximate because the sets $V_1, \ldots, V_m$ contain redundancy by definition, and the concatenation does not ensure that an edge is considered exactly once – see Fig. 2b and Fig. 2c.

Overapproximation of Tr. Consider a distributed signal $(S, \rightsquigarrow)$ of n signals, and let G_S be its canonical segmentation. We describe how the function γ defines a set $\mathsf{Tr}^+(S, \rightsquigarrow)$ of synchronous traces that overapproximates the set $\mathsf{Tr}(S, \rightsquigarrow)$. Consider $x \in S$, and let x' be a signal with the same temporal domain, and let $I = [s, s')$ be a segment in G_S. Let $(t_1, x'(t_1)), \ldots, (t_\ell, x'(t_\ell))$ be the edges of x' in segment I with $t_i < t_{i+1}$ for all $1 \leq i < \ell$. The signal x' is *I-consistent with* x iff the value expression $x'(s) \cdot x'(t_1) \cdot \ldots \cdot x'(t_\ell)$ belongs to $\gamma(x, I)$. Moreover, x' is *consistent with* x iff it is I-consistent with x for all $I \in G_S$. Now, let $S = (x_1, \ldots, x_n)$ and define $\mathsf{Tr}^+(S, \rightsquigarrow)$ as follows:

$$\mathsf{Tr}^+(S, \rightsquigarrow) = \{(x'_1, \ldots, x'_n) \mid x'_i \text{ is consistent with } x_i \text{ for all } 1 \leq i \leq n\}$$

Example 1. Recall $(S, \rightsquigarrow)$ and its γ function from Fig. 2. Consider the synchronous trace $w \in \mathsf{Tr}(S, \rightsquigarrow)$ where the rising edges of both signals occur at time 3 and the falling edges at time 5. Such a signal w would be included in $\mathsf{Tr}^+(S, \rightsquigarrow)$ since for each $i \in \{1, 2\}$, the value expression 1 is contained in $\gamma(x_i, [3, 4))$ and $\gamma(x_i, [4, 5))$, while 0 is contained in the remaining sets γ maps x_i to. Now, consider a synchronous trace (x'_1, x'_2) where both signals are initially 0, have rising edges at time 2 and 3.5, and falling edges at time 3 and 5. This trace does not belong to $\mathsf{Tr}(S, \rightsquigarrow)$ since x_1' and x_2' have more edges than x_1 and x_2. However, it belongs to $\mathsf{Tr}^+(S, \rightsquigarrow)$ since x_1' and x_2' are consistent with x_1 and x_2. Specifically, for each $i \in \{1, 2\}$, the value expression 01 is contained in $\gamma(x_i, [1, 3))$ and $\gamma(x_i, [3, 4))$, the expression 1 is contained in $\gamma(x_i, [4, 5))$, and 0 is contained in the remaining sets γ maps x_i to.

Finally, we prove that Tr^+ overapproximates Tr.

Lemma 1. *For every distributed signal* $(S, \rightsquigarrow)$*, we have* $\mathsf{Tr}(S, \rightsquigarrow) \subseteq \mathsf{Tr}^+(S, \rightsquigarrow)$.

5 Monitoring Algorithm

In this section, for a distributed signal $(S, \rightsquigarrow)$, we describe an algorithm to compute $[(S, \rightsquigarrow) \models \varphi]_+$ using the function γ from Sect. 4 without explicitly computing $\mathsf{Tr}^+(S, \rightsquigarrow)$. We introduce the *asynchronous product* of value expressions to capture interleavings within segments, then evaluate *untimed* and *timed operators*. Finally, we combine these steps to compute the *semantics* of STL$^+$. We also discuss an efficient implementation of the monitoring algorithm using *bit vectors*, heuristics to enhance *generalization* for real-valued signals, and a method to *combine* our approach with exact monitoring.

Asynchronous Products. Consider the value expressions $u_1 = 0 \cdot 1$ and $u_2 = 1 \cdot 0$ encoding the behaviors of two signals within a segment. Since behaviors within a segment are asynchronous, to capture their potential interleavings, we consider how the values in u_1 and u_2 can align. In particular, there are three potential alignments: (i) the rising edge of u_1 happens before the falling edge of u_2, (ii) the falling edge of u_2 happens before the rising edge of u_1, and (iii) they happen simultaneously. We respectively represent these with the tuples $(011, 110)$, $(001, 100)$, and $(01, 10)$ where the first component encodes u_1 and the second u_2. Formally, given two value expressions u_1 and u_2 (resp. sets L_1 and L_2 of value expressions), we define their *asynchronous product* as follows:

$$\begin{aligned} u_1 \otimes u_2 &= \left\{\mathsf{destutter}(v_1, v_2) \mid v_i \in \mathsf{stutter}_k(u_i), k = |u_1| + |u_2| - 1, i \in \{1, 2\}\right\} \\ L_1 \otimes L_2 &= \{u_1 \otimes u_2 \mid u_1 \in L_1, u_2 \in L_2\} \end{aligned}$$

Asynchronous products of value expressions allow us to lift value expressions to satisfaction signals of formulas.

Example 2. Recall $(S, \rightsquigarrow)$ and its γ function given in Fig. 2. To compute the value expressions encoding the satisfaction of $x_1 \wedge x_2$ in the segment $[1, 3)$, we first compute the asynchronous product $\gamma(x_1, [3, 4)) \otimes \gamma(x_2, [3, 4))$, and then the bitwise conjunction of each pair in the set. For example, taking the expression 010 for x_1 and 01 for x_2, the product contains the pair $(010, 011)$. Its bitwise conjunction is 010, encoding a potential behavior for the satisfaction of $x_1 \wedge x_2$.

Untimed Operations. As hinted in Example 2, to compute the semantics, we apply bitwise operations on value expressions and their asynchronous products to transform them into encodings of satisfaction signals of formulas. For example, to determine $[(S, \rightsquigarrow) \models \Diamond(x_1 \wedge x_2)]_+$, we first compute for each segment in G_S the set of value expressions for the satisfaction of $x_1 \wedge x_2$, and then from these compute those of $\Diamond(x_1 \wedge x_2)$. This compositional approach allows us to evaluate arbitrary STL$^+$ formulas.

First, we define bitwise operations on Boolean value expressions encoding atomic propositions. Then, we use these to evaluate untimed STL formulas over sets of value expressions. Let u and v be Boolean value expressions of length ℓ. We denote by $u \,\&\, v$ the bitwise-and operation, by $u \mid v$ the bitwise-or, and by $\sim\!u$ the bitwise-negation. We also define the *bitwise strong until* operator:

$$u\mathsf{U}^0 v = \left(\max_{i \leq j \leq \ell}\left(\min\left(v[j], \min_{i \leq k \leq j} u[k]\right)\right)\right)_{1 \leq i \leq \ell}$$

As usual, we derive *bitwise eventually* as $\mathsf{E}u = 1^\ell \mathsf{U}^0 u$, *bitwise always* as $\mathsf{A}u = \sim\!(\mathsf{E}\!\sim\!u)$, and *bitwise weak until* as $u\mathsf{U}^1 v = (u\mathsf{U}^0\ \mathrm{v})|(\mathsf{A}u)$. The distinction between U^0 and U^1 will be useful when we incrementally evaluate a formula. Finally, note that the output of these operations is a value expression of length ℓ. For example, if $u = 010$, we have $\mathsf{E}u = 110$ and $\mathsf{A}u = 000$.

Let $(S, \rightsquigarrow)$ be a distributed signal. Consider an atomic proposition $p \in \mathsf{AP}$ encoded as $x_p \in S$ and let φ_1, φ_2 be two STL formulas. We define the evaluation of untimed formulas with respect to $(S, \rightsquigarrow)$ and a segment $I \in G_S$ inductively:

$$\begin{aligned}
[\![(S, \rightsquigarrow), I \models p]\!] &= \gamma(x_p, I) \\
[\![(S, \rightsquigarrow), I \models \neg\varphi_1]\!] &= \{\sim\!u \mid u \in [\![(S, \rightsquigarrow), I \models \varphi_1]\!]\} \\
[\![(S, \rightsquigarrow), I \models \varphi_1 \wedge \varphi_2]\!] &= \mathsf{destutter}(\{u_1 \,\&\, u_2 \mid (u_1, u_2) \in [\![(S, \rightsquigarrow), I \models \varphi_1]\!] \otimes [\![(S, \rightsquigarrow), I \models \varphi_2]\!]\}) \\
[\![(S, \rightsquigarrow), I \models \varphi_1 \mathcal{U} \varphi_2]\!] &= \mathsf{destutter}(\{u_1 \mathsf{U}^a u_2 \mid (u_1, u_2) \in [\![(S, \rightsquigarrow), I \models \varphi_1]\!] \otimes [\![(S, \rightsquigarrow), I \models \varphi_2]\!], \\
&\qquad\qquad a \in \mathsf{first}([\![(S, \rightsquigarrow), I' \models \varphi_1 \mathcal{U} \varphi_2]\!])\})
\end{aligned}$$

where I' is the segment that follows I in G_S, if it exists. For completeness, for every formula φ we define $[\![(S, \rightsquigarrow), I' \models \varphi]\!] = \{0\}$ when $I' \notin G_S$. When I is the first segment in G_S, we simply write $[\![(S, \rightsquigarrow) \models \varphi]\!]$. Similarly as above, we can use the standard derived operators to compute the corresponding sets of value expressions. For a given formula and a segment, the evaluation above produces a set of value expressions encoding the formula's satisfaction within the segment.

Example 3. Recall $(S, \rightsquigarrow)$ and γ from Fig. 2. To compute $[\![(S, \rightsquigarrow), [5, 7) \models \Diamond(x_1 \wedge x_2)]\!]$, we first compute $[\![(S, \rightsquigarrow), [5, 7) \models x_1 \wedge x_2]\!]$ by taking the bitwise conjunction over the asynchronous product $\gamma(x_1, [5, 7)) \otimes \gamma(x_2, [5, 7))$ and destuttering. For example, since $010 \in \gamma(x_1, [5, 7))$ and $01 \in \gamma(x_2, [5, 7))$, the pair $(0010, 0111)$ is in the product, whose conjunction gives us 010 after destuttering. Repeating this for the rest, we obtain $[\![(S, \rightsquigarrow), [5, 7) \models x_1 \wedge x_2]\!] = \{0, 01, 010, 1, 10\}$. Finally, we compute $[\![(S, \rightsquigarrow), [5, 7) \models \Diamond(x_1 \wedge x_2)]\!]$ by applying each expression in $[\![(S, \rightsquigarrow), [5, 7) \models x_1 \wedge x_2]\!]$ the bitwise eventually operator and destuttering. The resulting set $\{0, 1, 10\}$ encodes the satisfaction signal of $\Diamond(x_1 \wedge x_2)$ in $[5, 7)$. Note that we do not need to consider the evaluation of the next segment for the eventually operator since $[\![(S, \rightsquigarrow), [7, 8) \models x_1 \wedge x_2]\!] = \{0\}$.

Timed Operations. Handling timed operations requires a closer inspection as value expressions are untimed by definition. We address this issue by considering how a given evaluation interval relates with a given segmentation. For example, take a segmentation $G_S = \{[0, 4), [4, 6), [6, 10)\}$ and an evaluation interval $J = [0, 5)$. Suppose we are interested in how a signal $x \in S$ behaves with respect to J over the first segment $I = [0, 4)$. First, to see how J relates with G_S with respect to $I = [0, 4)$, we "slide" the interval J over $I \oplus J = [0, 9)$ and consider the different ways it intersects the segments in G_S. Initially, J covers the entire segment $[0, 4)$ and the beginning of $[4, 6)$, for which the potential behaviors of x are captured by the set $\gamma(x, [0, 4)) \cdot \mathsf{prefix}(\gamma(x, [4, 6)))$. Now, if we slide the window and take $J' = [3, 7)$, the window covers the ending of $[0, 4)$, the entire $[4, 6)$, and the beginning of $[6, 10)$, for which the potential behaviors are captured by the set $\mathsf{suffix}(\gamma(x, [0, 4))) \cdot \gamma(x, [4, 6)) \cdot \mathsf{prefix}(\gamma(x, [6, 9))$. We call these sets the *profiles* of J and J' with respect to $(S, \rightsquigarrow)$, x, and I.

We first present the definitions, and then demonstrate them in Examples 4 and 5 and Fig. 3. Let $(S, \rightsquigarrow)$ be a distributed signal, $I \in G_S$ be a segment, and φ be an STL formula. Let us introduce some notation. First, we abbreviate the set $[\![(S, \rightsquigarrow), I \models \varphi]\!]$ of value expressions as $\tau_{\varphi,I}$. Second, given an interval K, we respectively denote by l_K and r_K its left and right end points. Third, recall that we denote by F the set of end points of G_S (see Sect. 4). Given an interval J, we define the *profile* of J with respect to $(S, \rightsquigarrow)$, φ, and I as follows:

$$\mathsf{profile}((S, \rightsquigarrow), \varphi, I, J) = \begin{cases} \mathsf{prefix}(\tau_{\varphi,I}) & \text{if } l_I = l_J \wedge r_I > r_J \\ \mathsf{infix}(\tau_{\varphi,I}) & \text{if } l_I < l_J \wedge r_I > r_J \\ \tau_{\varphi,I} \cdot \kappa(\varphi, I, J) & \text{if } l_I = l_J \wedge r_I \leq r_J \wedge r_J \in F \setminus J \\ \tau_{\varphi,I} \cdot \kappa(\varphi, I, J) \cdot \mathsf{first}(\tau_{\varphi,I'}) & \text{if } l_I = l_J \wedge r_I \leq r_J \wedge r_J \in F \cap J \\ \tau_{\varphi,I} \cdot \kappa(\varphi, I, J) \cdot \mathsf{prefix}(\tau_{\varphi,I'}) & \text{if } l_I = l_J \wedge r_I \leq r_J \wedge r_J \notin F \\ \mathsf{suffix}(\tau_{\varphi,I}) \cdot \kappa(\varphi, I, J) & \text{if } l_I < l_J < r_I \leq r_J \wedge r_J \in F \setminus J \\ \mathsf{suffix}(\tau_{\varphi,I}) \cdot \kappa(\varphi, I, J) \cdot \mathsf{first}(\tau_{\varphi,I'}) & \text{if } l_I < l_J < r_I \leq r_J \wedge r_J \in F \cap J \\ \mathsf{suffix}(\tau_{\varphi,I}) \cdot \kappa(\varphi, I, J) \cdot \mathsf{prefix}(\tau_{\varphi,I'}) & \text{if } l_I < l_J < r_I \leq r_J \wedge r_J \notin F \\ \{\epsilon\} & \text{otherwise} \end{cases}$$

where we assume J is trimmed to fit the temporal domain of S and $I' \in G_S$ is such that $r_J \in I'$. Moreover, $\kappa(\varphi, I, J)$ is the concatenation $\tau_{\varphi,I_1} \cdot \ldots \cdot \tau_{\varphi,I_m}$ such that $I, I_1, \ldots, I_m, I'$ are consecutive segments in G_S. If $I_1, \ldots, I_m$ do not exist, we let $\kappa(\varphi, I, J) = \{\epsilon\}$. Note that the last case happens when $I \cap J$ is empty. We now formalize the intuitive approach of "sliding" J over the segmentation to obtain the various profiles it produces as follows:

$$\mathsf{pfs}((S, \rightsquigarrow), \varphi, I, J) = \{\mathsf{destutter}(\mathsf{profile}((S, \rightsquigarrow), \varphi, I, J')) \mid J' \subseteq I \oplus J, J' \sim J\}$$

where $J' \sim J$ holds when $|J'| = |J|$ and J' contains an end point (left or right) iff J does so. Note that although infinitely many intervals J' satisfy the conditions given above (due to denseness of time), the set defined by pfs is finite. We demonstrate this and the computation of pfs in Example 4 and Fig. 3.

Example 4. Recall $(S, \rightsquigarrow)$ and γ from Fig. 2. We describe the computation of $\mathsf{pfs}((S, \rightsquigarrow), x_1, [1, 3), [0, 1))$. Sliding the interval $[0, 1)$ over the window $[1, 3) \oplus [0, 1)$ (see Fig. 3) gives us the following sets: $P_1 = \mathsf{destutter}(\mathsf{prefix}(\gamma(x_1, [1, 3))))$, $P_2 = \mathsf{destutter}(\mathsf{infix}(\gamma(x_1, [1, 3))))$, and $P_3 = \mathsf{destutter}(\mathsf{suffix}(\gamma(x_1, [1, 3))))$ where all equal to $\{0, 01, 1\}$. Moreover, we have $P_4 = \mathsf{destutter}(\mathsf{suffix}(\gamma(x_1, [1, 3))) \cdot \mathsf{prefix}(\gamma(x_1, [3, 4)))) = \{0, 01, 010, 0101, 01010, 1, 10, 101, 1010\}$. We obtain that $\mathsf{pfs}((S, \rightsquigarrow), x_1, [1, 3), [0, 1)) = \{P_1, P_2, P_3, P_4\}$. This set overapproximates the potential behaviors of x_1, for all $t \in [1, 3)$, in the interval $t \oplus [0, 1)$.

Let φ_1 and φ_2 be two STL formulas. Intuitively, once we have the profiles of a given interval J with respect to φ_1 and φ_2, we can evaluate the corresponding untimed formulas on the product of these profiles and concatenate them. Formally, we handle the evaluation of timed formulas as follows:

$$\begin{aligned}\llbracket (S, \rightsquigarrow), I \models \varphi_1 \mathcal{U}_J \varphi_2 \rrbracket = \mathsf{destutter}(\{u_1 \mathsf{U}^0 u_2 \mid (u_1, u_2) \in P_1 \otimes Q_1\} \cdot \ldots \\ \ldots \cdot \{u_1 \mathsf{U}^0 u_2 \mid (u_1, u_2) \in P_k \otimes Q_k\})\end{aligned}$$

where $\mathsf{pfs}((S, \rightsquigarrow), \varphi_1, I, J) = \{P_1, \ldots, P_k\}$ and $\mathsf{pfs}((S, \rightsquigarrow), \varphi_2, I, J) = \{Q_1, \ldots, Q_k\}$ such that the intervals producing P_i and Q_i respectively start before those producing P_{i+1} and Q_{i+1} for all $1 \leq i < k$.

Example 5. Let $(S, \rightsquigarrow)$ and γ be as in Fig. 2. We demonstrate the evaluation of the timed formula $\Diamond_{[0,1)}\, x_1$ over the segment $[1, 3)$. Recall from Example 4 the set $\mathsf{pfs}((S, \rightsquigarrow), x_1, [1, 3), [0, 1)) = \{P_1, P_2, P_3, P_4\}$ of profiles. First, we apply the bitwise eventually operator to each value expression in each of these profiles separately: $\{\mathsf{E}u \mid u \in P_1\} = \{\mathsf{E}u \mid u \in P_2\} = \{\mathsf{E}u \mid u \in P_3\} = \{0, 1\}$, and $\{\mathsf{E}u \mid u \in P_4\} = \{0, 10, 1\}$. We then concatenate these and destutter to obtain $\llbracket (S, \rightsquigarrow), [1, 3) \models \Diamond_{[0,1)}\, x_1 \rrbracket = \{0, 01, 010, 0101, 01010, 1, 10, 101, 1010\}$.

Computing the Semantics of STL$^+$. Putting it all together, given a distributed signal $(S, \rightsquigarrow)$ and an STL$^+$ formula φ, we can compute $[(S, \rightsquigarrow) \models \varphi]_+$ thanks to the following theorem.

Theorem 2. *For every distributed signal $(S, \rightsquigarrow)$ and STL formula φ we have $[(S, \rightsquigarrow) \models \varphi]_+ = \top$ (resp. $\bot$, $?$) iff $\mathsf{first}(\llbracket (S, \rightsquigarrow) \models \varphi \rrbracket) = \{1\}$ (resp. $\{0\}$, $\{0, 1\}$).*

Sets of Boolean Value Expressions as Bit Vectors. Asynchronous products are expensive to compute. Our implementation relies on the observation that sets of boolean value expressions and their operations can be efficiently implemented through bit vectors. Intuitively, to represent such a set, we encode each element using its first bit and its length since value expressions are boolean and always destuttered. Moreover, to evaluate untimed operations on such sets, we only need to know the maximal lengths of the four possible types of expressions ($0\dots0$, $0\dots1$, $1\dots0$, and $1\dots1$) and whether the set contains 0 or 1 (to handle some edge cases). This is because value expressions within the same segments are completely asynchronous and the possible interleavings obtained from shorter expressions can be also obtained from longer ones.

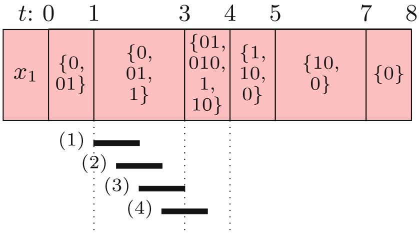

Fig. 3. The profiles of $J = [0, 1)$ with respect to $x_1 \in S$ of Fig. 2. A representative interval for each profile is shown with solid black lines below the table.

Generalization to Real-Valued Signals. Our approximate distributed monitoring method, denoted Adm, can be extended to real-valued signals and numerical predicates. The key is that finite-length piecewise-constant signals take finitely many values. By defining Σ as a finite alphabet of these values, we can compute atomic propositions as above. For example, if the asynchronous product of two signals x_1 and x_2 yields $(2 \cdot 2 \cdot 3, 1 \cdot 0 \cdot 1)$, adding these letter-by-letter results in $3 \cdot 2 \cdot 4$, and comparing with > 2 gives 101.

We can avoid explicit computation of asynchronous products for some formulas and numerical predicates. Since signals are asynchronous within segments, we can compute potential value sets instead of sequences. This approach is called *Fine*, denoted by Adm-F. Assuming $x_1 + x_2$ is constant within this segment, we can avoid explicit interleaving computations. Note that Adm-F overapproximates traces when order matters. The approach *Coarse*, denoted Adm-C, abstracts *Fine* by only considering extreme values, which is useful for monotonic operations where the extreme values of outputs derive from inputs.

We assumed so far that the central monitor runs on a process independent of the observed agents. Lastly, we also consider a setting where the monitor runs on one of the observed agents. This approach reduces asynchrony by using the agent's local clock as a reference point for the monitor. We call this *Relative*, denoted Adm-Fr or Adm-Cr depending on the approach it is paired with. We evaluate these in Sect. 6.

Combining Exact and Approximate Monitoring. We propose a method that combines approximate distributed monitors (Adm) with their exact counterparts (Edm) with the aim to achieve better computational performance while remaining precise. The approach works as follows: Given a distributed signal $(S, \rightsquigarrow)$ and a formula φ, compute the approximate verdict $v \leftarrow [(S, \rightsquigarrow) \models \varphi]_+$.

If the verdict is inconclusive, i.e., $v = ?$, then compute and return the exact verdict $[(S, \rightsquigarrow) \models \varphi]$, else return v. We evaluate this approach in Sect. 6.

6 Experimental Evaluation

6.1 Research Questions

We seek answers to the following research questions (RQs):

RQ1. *What is the tradeoff between the efficiency and the accuracy of approximate distributed monitors?* The approximate distributed monitoring comes with a price in terms of the loss of accuracy. We want to understand the tradeoff between the potential speedups that an approximate distributed monitor can achieve when compared to its exact counterpart and the consequent loss in accuracy due to the approximations. We would also like to identify the classes of signals and properties for which this tradeoff is effective.
RQ2. *Can the combination of approximate and exact distributed monitors increase efficiency while preserving accuracy?* We are interested in evaluating whether a smart, combined use of approximate and exact distributed monitors can still bring improvements in monitoring efficiency while guaranteeing the accuracy of the monitoring verdicts.

6.2 Experimental Setup

Distributed Monitors. In our study, we compare our approximate distributed monitoring (Adm) approach and its variants to an exact distributed monitoring approach (Edm).[1] For Edm, we take a variant of the distributed monitoring procedure from [18] that allows to evaluate STL specifications over distributed traces using SMT-solving. Originally, that procedure assumes that input signals are polynomial continuous functions. We adapt the SMT-based approach to consider input signals as piecewise-constant signals to make a consistent comparison with Adm. We note that the passage from the polynomial continuous to piecewise-constant input signals reduces the efficiency of the SMT-based monitors. We also observe that the SMT-based monitors from [18] can split the input trace into multiple segments and evaluate the specification incrementally, segment-by-segment, allowing early termination of the monitor in some cases. Since the focus of this paper is purely on the offline monitoring, we also use the exact monitors without their incremental mode.

Experimental Subjects. To answer our research questions, we use (1) a *random generator (RG)* of distributed traces, (2) a *water tank (WT)* case study, and (3) a *swarm of drones (SD)* case study. *The random generator (RG)* uses uniform distribution to generate distributed traces, in which the user can control the duration d of the trace, as well as the ε bound on the uncertainty at which the events happen. *Water tank (WT)* model is a SimuLink model of a hybrid

[1] The code is available at https://github.com/egesarac/ApxDistMon.

Table 1. STL specifications used in the experiments.

Subject	STL formula(s)
RG	$\varphi_1 = \Box(p \wedge q) \qquad \varphi_2 = \Box(p \Rightarrow \Diamond q) \qquad \varphi_3 = \Box(p \Rightarrow \Diamond_{[0,1)}\, q)$
WT	$\varphi_{\mathrm{WT}} = \Box \left(\sum_{i=1}^{n} x_i > c\right)$
SD	$\varphi_{\mathrm{SD}} = \bigwedge_{1 \leq i \neq j \leq n} \Box \left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} > c\right)$

high pressure water distribution system consisting of two water tanks. Inlet pipes connect each water tank to an external source, and outlet pipes distribute high pressure water that is regulated by valves. Each valve is operated by a controller that samples the outflow pressure at 20 Hz using its local clock. Our model is a simplified emulation of the Refueling Water Storage Tanks (RWST) module of an Emergency Core Cooling System (ECCS) of a Pressurized Water Reactor Plant [24]. *Swarm of drones (SD)* model is generated using a path planning software, Fly-by-Logic [22]. Here, a swarm of drones perform various reach-avoid missions, while securing objectives such as reaching a goal within a deadline, avoiding obstacles and collisions. The path planner finds the most robust trajectory using a temporal logic robustness optimizer. These trajectories are sampled at 20 Hz. Note that the actual values of clock skew are less important than the fact that when clock skew exceeds the sampling interval, we encounter the problem of uncertainty.

Specifications. Table 1 shows the STL specifications that we use to evaluate our experimental subjects. Specifications φ_1, φ_2 and φ_3 are monitored against the distributed traces created by the random generator and represent different classes and fragments of Boolean-valued temporal formulas. The first specification φ_1 is an LTL formula in which both the outer temporal operator ($\Box$) and the inner Boolean operator ($\wedge$) are conjunctive. The second formula φ_2 is the common LTL response formula which combines conjunctive ($\Box$) and disjunctive ($\Diamond, \Rightarrow$) operators. Finally, φ_3 adds a bounded real-time response requirement to the previous specification. The specification φ_{WT} associated to the water tank case study is an STL formula in which a sum of signals originating from different agents is compared to a constant. Finally, the specification φ_{SD} defines a mutual separation property over a swarm of drones, requiring more sophisticated arithmetic operations on signals originating from different agents.

Computing Platform. We used a laptop with Ubuntu 24.04, an AMD Ryzen 7 4800HS CPU at 2.90 GHz clock rate, and 16GB of RAM. Adm is implemented in C++ and compiled using `g++` version 13.2.0 with the optimization flag `-O3` enabled, and Edm invokes the SMT-solver Z3 [20] and is based on [18].

6.3 Discussion

Random Generator. Figure 4 summarizes the results of evaluating specifications φ_1 to φ_3 against distributed traces from RG. The first column in the figure

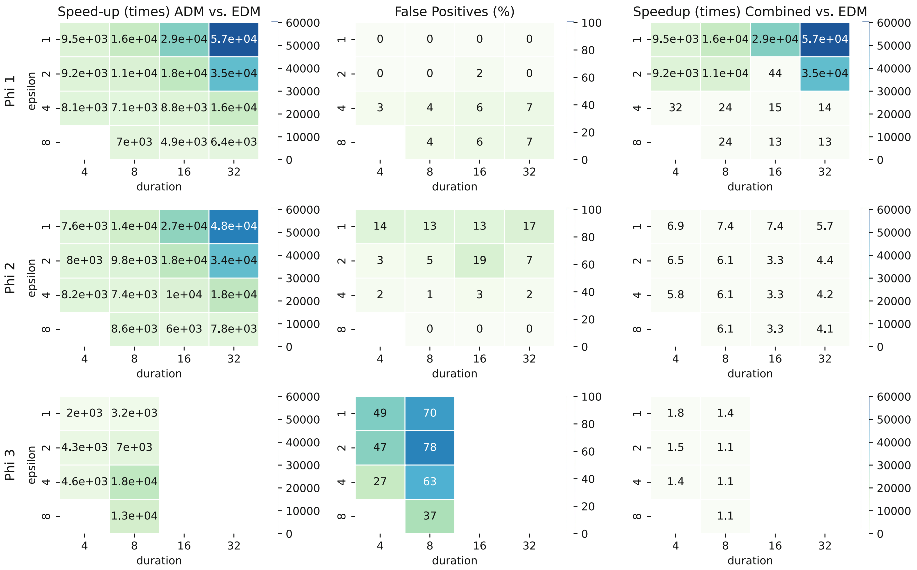

Fig. 4. Results on monitoring φ_1 to φ_3 on distributed traces created by the RG.

depicts a heatmap where cells show the speedup of Adm compared to Edm when evaluating the formula on the given distributed trace with duration d and uncertainty bound ε. The second column shows a heatmap where every cell shows the percentage of *false positives* (FP) introduced by Adm, where Adm evaluates to inconclusive when the Edm (real) verdict is true or false. Finally, the third column depicts a heatmap, where each cell estimates the achieved speedup when combining Adm with Edm, compared to using only Edm.

We see that Adm consistently achieves speedups of *several orders of magnitude* compared to the Edm approach. The speedups range from several thousands to almost 60 thousand times and are the highest for long signals with low uncertainty bounds. The price paid in terms of accuracy highly depends on the type of specification and the uncertainty bounds. For example, Adm is very accurate when monitoring the property φ_1 in which both the temporal and the combinatorial operators are conjunctive. On the other hand, having a combination of conjunctive and disjunctive operations (as in φ_2 and φ_3) increases the number of FPs. Surprisingly, we see that in these cases the introduction of FPs is higher for lower values of ε. This is because even Edm gives many inconclusive verdicts for higher values of ε. We see that adding real-time modalities to the temporal operators increases FPs. Finally, we can see (Fig. 4 right column) that by combining Edm and Adm, we consistently get better performance than by using Edm only, even in cases where Adm introduces a high percentage of FPs.

Water Tank. Speedups increase with the number of signals n and decrease with ε. The Adm-C method shows significant improvements over Edm, with

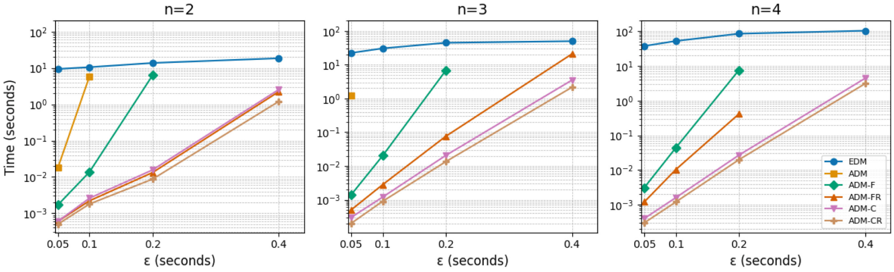

Fig. 5. Running times for monitoring φ_{WT} in log scale. Time limit is 120 s, and timed-out instances are not shown.

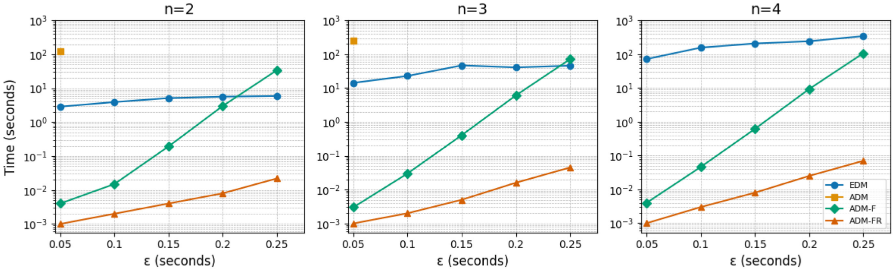

Fig. 6. Running times for monitoring φ_{SD} in log scale. Time limit is 360 s, and timed-out instances are not shown.

up to a 104000× speedup in the best-case (when $n = 4$ and $\varepsilon = 0.05$) and an 8× speedup in the worst-case (when $n = 2$ and $\varepsilon = 0.4$). Note that $\varepsilon = 0.4$ is near the realistic upper limit [18], indicating no scalability issues. The Adm-Cr method adds up to a 1.63× speedup over Adm-C. The Adm-Fr approach significantly improves Adm-F, bringing it below the time-out limit with up to a 476× speedup in non-time-out instances. As expected, Adm does not perform well. All methods produce the same verdict for the considered traces (Fig. 5).

Swarm of Drones. Similar to the previous case scenario, speedups in the mutual separation case increase with n and decrease with ε. The Adm-Fr method achieves about a 78000× speedup in the best-case scenario (when $n = 4$ and $\varepsilon = 0.05$) and a 23× speedup in the worst-case (when $n = 2$ and $\varepsilon = 0.25$). The Adm-F method performs slower than SMT in two cases where n is small and ε is large. As in the previous case, Adm does not perform well. Additionally, Adm-C and Adm-Cr are not applicable here because the arithmetic operations are not monotonic. Again, all methods yield the same verdicts (Fig. 6).

Summary. To answer RQ1, we find that ADM achieves a speedup of three to five orders of magnitude over EDM. However, the efficiency-accuracy tradeoff depends on the type of specifications, input signal duration, and maximal clock skew. Arithmetic and timed operators are particularly affected by ADM's overapproximations, reducing accuracy. Untimed temporal properties, especially those without mixed conjunctive and disjunctive operations, maintain high accuracy and offer an excellent tradeoff. Despite lower accuracy in some cases, combining ADM and EDM still results in significant gains, positively answering RQ2.

7 Conclusion

We presented an approximate, modular procedure for distributed STL monitoring that significantly improves efficiency over exact SMT-based methods. In this paper, the focus was on the offline evaluation of distributed traces. We plan to extend our monitoring approach to the online setting. We will also exploit the modular nature of our monitors to have a better control over their accuracy. More specifically, for every operator, we can either generate the exact or the approximate evaluation algorithm.

References

1. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Lehtinen, K.: The best a monitor can do. In: Baier, C., Goubault-Larrecq, J. (eds.) 29th EACSL Annual Conference on Computer Science Logic, CSL 2021, 25–28 January 2021, Ljubljana, Slovenia (Virtual Conference). LIPIcs, vol. 183, pp. 7:1–7:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPICS.CSL.2021.7
2. Aceto, L., Attard, D.P., Francalanza, A., Ingólfsdóttir, A.: On benchmarking for concurrent runtime verification. In: FASE 2021. LNCS, vol. 12649, pp. 3–23. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-71500-7_1
3. Alechina, N., Dastani, M., Logan, B.: Norm approximation for imperfect monitors. In: Bazzan, A.L.C., Huhns, M.N., Lomuscio, A., Scerri, P. (eds.) International conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2014, Paris, France, 5–9 May 2014, pp. 117–124. IFAAMAS/ACM (2014). http://dl.acm.org/citation.cfm?id=2615753
4. Bartocci, E., Grosu, R.: Monitoring with uncertainty. In: Bortolussi, L., Bujorianu, M., Pola, G. (eds.) Proceedings Third International Workshop on Hybrid Autonomous Systems, HAS 2013, Rome, Italy, 17th March 2013. EPTCS, vol. 124, pp. 1–4 (2013). https://doi.org/10.4204/EPTCS.124.1
5. Bauer, A., Falcone, Y.: Decentralised LTL monitoring. Formal Methods Syst. Design **48**(1–2), 46–93 (2016)
6. Bonakdarpour, B., Fraigniaud, P., Rajsbaum, S., Rosenblueth, D.A., Travers, C.: Decentralized asynchronous crash-resilient runtime verification. J. ACM **69**(5), 34:1–34:31 (2022)
7. Chauhan, H., Garg, V.K., Natarajan, A., Mittal, N.: A distributed abstraction algorithm for online predicate detection. In: Proceedings of the 32nd IEEE Symposium on Reliable Distributed Systems (SRDS), pp. 101–110 (2013)

8. Colombo, C., Falcone, Y.: Organising LTL monitors over distributed systems with a global clock. Formal Methods Syst. Design **49**(1–2), 109–158 (2016)
9. El-Hokayem, A., Falcone, Y.: On the monitoring of decentralized specifications: semantics, properties, analysis, and simulation. ACM Trans. Softw. Eng. Methodol. **29**(1), 1:1–1:57 (2020)
10. Ganguly, R., Momtaz, A., Bonakdarpour, B.: Runtime verification of partially-synchronous distributed system. Formal Methods Syst. Design (FMSD) (2024, to appear)
11. Ganguly, R., Momtaz, A., Bonakdarpour, B.: Distributed runtime verification under partial synchrony. In: Bramas, Q., Oshman, R., Romano, P. (eds.) 24th International Conference on Principles of Distributed Systems, OPODIS 2020, 14–16 December 2020, Strasbourg, France (Virtual Conference). LIPIcs, vol. 184, pp. 20:1–20:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)https://doi.org/10.4230/LIPIcs.OPODIS.2020.20
12. Garg, V.K.: Predicate detection to solve combinatorial optimization problems. In: Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 235–245. ACM (2020)
13. Henzinger, T.A., Mazzocchi, N., Saraç, N.E.: Abstract monitors for quantitative specifications. In: Dang, T., Stolz, V. (eds.) RV 2022. LNCS, vol. 13498, pp. 200–220. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-17196-3_11
14. Henzinger, T.A., Mazzocchi, N., Saraç, N.E.: Quantitative safety and liveness. In: Kupferman, O., Sobocinski, P. (eds.) FoSSaCS 2023. LNCS, vol. 13992, pp. 349–370. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30829-1_17
15. Henzinger, T.A., Saraç, N.E.: Quantitative and approximate monitoring. In: 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, 29 June–2 July 2021, pp. 1–14. IEEE (2021). https://doi.org/10.1109/LICS52264.2021.9470547
16. Maler, O., Nickovic, D.: Monitoring properties of analog and mixed-signal circuits. Int. J. Softw. Tools Technol. Transf. **15**(3), 247–268 (2013). https://doi.org/10.1007/s10009-012-0247-9
17. Mittal, N., Garg, V.K.: Techniques and applications of computation slicing. Distrib. Comput. **17**(3), 251–277 (2005)
18. Momtaz, A., Abbas, H., Bonakdarpour, B.: Monitoring signal temporal logic in distributed cyber-physical systems. In: Mitra, S., Venkatasubramanian, N., Dubey, A., Feng, L., Ghasemi, M., Sprinkle, J. (eds.) Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems, ICCPS 2023, (with CPS-IoT Week 2023), San Antonio, TX, USA, 9–12 May 2023, pp. 154–165. ACM (2023). https://doi.org/10.1145/3576841.3585937
19. Mostafa, M., Bonakdarpour, B.: Decentralized runtime verification of LTL specifications in distributed systems. In: Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 494–503 (2015)
20. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
21. Ogale, V.A., Garg, V.K.: Detecting temporal logic predicates on distributed computations. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 420–434. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75142-7_32
22. Pant, Y.V., Abbas, H., Mangharam, R.: Smooth operator: control using the smooth robustness of temporal logic. In: 2017 IEEE Conference on Control Technology and Applications (CCTA), pp. 1235–1240. IEEE (2017)

23. Stoller, S.D., et al.: Runtime verification with state estimation. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 193–207. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_15
24. USNRC: Pressurized water reactor systems (2021). https://www.nrc.gov/reading-rm/basic-ref/students/for-educators/04.pdf