Adaptable configuration of decentralized monitors

Ennio Visconti, Ezio Bartocci, Yliès Falcone, Laura Nenzi

Adaptable configuration of decentralized monitors

Ennio Visconti[1 [0000−0002−1146−4850]], Ezio Bartocci[1 [0000−0002−8004−6601]], Yliès Falcone[2 [0000−0002−0114−0641]], and Laura Nenzi[3 [0000−0003−2263−9342]]

[1] TU Wien, Austria
[2] Université Grenoble Alpes, France
[3] University of Trieste, Italy

Abstract. Prominent challenges in runtime verification of a distributed system are the correct placement, configuration, and coordination of the monitoring nodes. This work considers state-of-the-art decentralized monitoring practices and proposes a framework to recommend efficient configurations of the monitoring system depending on the target specification. Our approach aims to optimize communication over several features (e.g., minimizing the number of messages exchanged, the number of computations happening overall, etc.) in contexts where finding an efficient communication strategy requires slow simulations. We optimize by training multiple machine learning models from simulations combining traces, formulae, and systems of different sizes. The experimental results show that the developed model can reliably suggest the best configuration strategy in a few nanoseconds, contrary to the minutes or possibly hours required by direct simulations that would be impractical at runtime.

1 Introduction

Computational systems are often spatially distributed and interconnected. To operate correctly and efficiently, they must be carefully verified and tested. The field practitioners know this very well and exploit several verification techniques to guarantee their correctness, both at design time [30] and runtime [8]. Runtime verification [2], in particular, is gaining significant momentum for several reasons: (i) it is a lightweight verification technique, and it allows checking more complex specifications or more complex systems; (ii) at runtime, some properties might be easier to express, as there is more contextual information that can help for the target analysis; (iii) runtime verification is typically more flexible, allowing to analyze system's changes *as they occur* [12,24,34], or changing requirements *on the go* [29].

However, only some techniques can exploit the system's distributed nature to provide guarantees promptly and efficiently. Among these, decentralized monitoring [5,11] is a promising approach, as it allows the distribution of the monitoring task among the components of the system by mapping requirements from the perspective of the global view of the system to the locally observable information. While this technique is promising, it still poses several challenges[5,31]: (i)

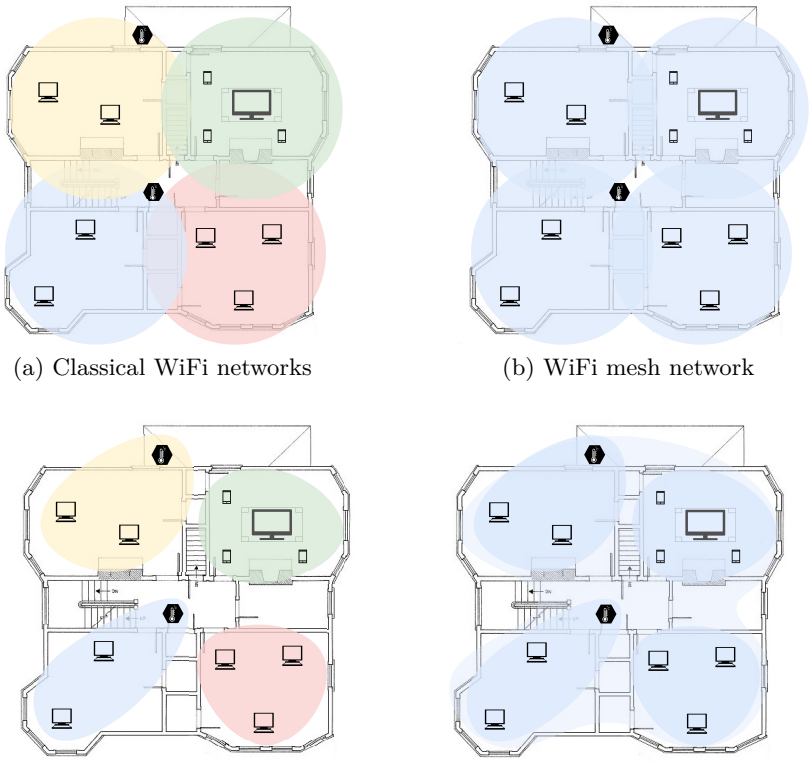
It can be non-trivial to adequately express abstract/global requirements in terms of the individual components that form the system; (ii) A *distributed* monitor is a distributed system and,rep as such, can be challenging to handle by itself; (iii) Current approaches are still very rigid, as they do not encompass alternative network configurations or specification changes, and (iv) The way the monitors are deployed over the network (i.e. their configuration) can significantly impact the system's performances under scrutiny, possibly to a level where monitoring is too costly or hinders the actual observability of the requirements.

Running example Consider a modern WiFi setup for office space, like the one in Fig. 1a, where a different color represents every access point. In this environment, we expect two characteristics:

R1 it should guarantee no disruptions when users are moving around, and data is exchanged in real-time,
R2 it should provide an adaptive bandwidth optimized based on the usage patterns of the connected users.

In this scenario, two technologies are adequate for this task, i.e., a mesh setup (IEEE 802.11s) to avoid disconnections of moving devices (R1, represented in Fig. 1b-1d by using the same color for all the access points), and beamforming (prescribed in IEEE 802.11ac) to optimize based on the network usage patterns (R2, shown in Fig. 1c-1d by using different shapes for the various access points to represent different traffic patterns), which can be used independently (Fig. 1b-1c) or combined (Fig. 1d). When adopting a design like the one in Figure 1d, the WiFi access points (APs) form a fully connected distributed network that exchanges information to assess the optimal distribution of the load coming from the requests of the devices. The requirements to ensure the mesh network is performing beamforming correctly could be easily and efficiently checked by a decentralized monitor deployed on the APs themselves. Still, the way the specific requirements are checked might change depending on the current configuration of the network, as well as the operating mode of the APs, as we will see in the following sections.

Related work Our work is based on the literature on decentralized monitoring of linear temporal logic [27] (LTL) with shared global clock [15,16], that resulted in the DecentMon tool [17] also employed in our experiments. In [7], the authors address the problem of fault-tolerance in decentralized monitoring, extending the ideas presented in [18]. However, their work does not tackle the problem of optimal configuration (or placement). To the best of our knowledge, no other work addresses this problem. In the last year, runtime verification has gained increased attention in the context of artificial intelligence, both because it plays a central role in ensuring the safety and correctness of AI systems during runtime [32], and because of the benefits logic encodings can bring to word embeddings [26,28,1] (the term more commonly used to address encodings in natural language processing literature). On the other side, Machine Learning methods were exploited in several contexts related to runtime verification: for



(a) Classical WiFi networks

(b) WiFi mesh network

(c) WiFi networks with beamforming

(d) Wifi mesh network + beamforming

Fig. 1. Example of four WiFi setups, from a traditional one (a) to a modern one (d). Subfigures from top to bottom show how beamforming transforms the networks' shapes, while subfigures from left to right show the impact of mesh networks in creating a unified continuous network. Original floorplans from [22].

predictive monitor [9], for learning temporal logic specifications [3,6,25], and as a similarity function on formulae of temporal logic [10].

Contributions of this paper This paper proposes a novel approach to provide recommendations for efficiently distributing a decentralized monitor for temporal traces. We do that by (i) formalizing the decision of configurations of decentralized monitors as an optimization problem over several cost functions (ii) introducing a way to encode LTL formulae to approximate the search for the best configuration of decentralized monitors, (iii) presenting a classification and a regression task that approximate this search, together with multiple machine learning models solving these tasks over a dataset of 480 thousand combinations of monitoring traces generated over 40 hours of simulation.

Paper organization The rest of the paper is organized as follows. In Sec. 2, we introduce the decentralized monitoring background necessary to develop our work. In Sec. 3, we present the concept of distribution strategies, we formalize it as an optimal synthesis one, and we present possible ways of finding a good one in Sec. 4. In Sec. 5, we present the results of the models we trained, showing how they can achieve excellent performance predictions in almost no time once trained. Lastly, we conclude the paper in Sec. 6, presenting several directions for future works.

2 Decentralized systems, monitoring and traces

Let $\mathbb{N}$ be the set of non-negative natural numbers, and let $[a, b)$ denote an interval closed on a and open on b, and 2^A denote the powerset of A. We will denote by $\mathcal{T} = \{0, 1, \ldots, T\}$ the global time domain of reference for the system and traces (finite or infinite, i.e., when $T \to \infty$). Based on this fundamental notion of time, we clarify the meaning of *decentralized systems*, how their respective *events* are aggregated in timed *traces*, and the kind of *monitoring* we are interested in.

2.1 System & Events

Let $\mathcal{C}$ be some range of numbers from $1, \ldots, n$ denoting the set of components i. A system is a *complete graph* i.e., a graph where a unique edge connects each pair of distinct vertices. We denote by $\mathcal{E}_i$ the set of *locally observable events* by some component $i \in \mathcal{C}$. A notable element of $\mathcal{E}_i$ is the *no event* ε, which denotes the fact that no action has been observed.

Definition 1 (Local Trace). *We call local trace the sequence of events u_i that happen at all time points $t \in \mathcal{T}$ for the i-th component, i.e. $u_i : \mathcal{T} \to \mathcal{E}_i$.*

Running example We can consider the network of Fig. 1 as a system of four components ($\mathcal{C} = \{1, 2, 3, 4\}$), representing the WiFi Access Points (APs). Several locally observable events can be considered from a WiFi AP: an increase in the number of packets requested (e.g., a big file is being downloaded), an increase in packet priority is asked for (a video call is starting), etc. In particular, our event sets are defined as follows: all events start with a numeric id i corresponding to a unique identifier of the AP (the component) within the system; then they are optionally followed by `_D_`j when the event refers to a specific device

communicating to the APs, where j will be used as a unique numeric id (in reality the MAC address of the device would be seen), and optionally followed by `_A_`k when the event is related to a specific antenna (we assume each AP has two antennas for simplicity), and completed by $\_n$ where n belongs to a finite set of specific events observable, e.g. `CONNECTED`, `HIGH_POWER`, `HIGH_TRAFFIC`, `IN_RANGE`, etc. For example, a local trace of the component 1 is the sequences of events $(e_0, \varepsilon, e_2, \ldots)$, where e_0 is the event `1_D_1_CONNECTED`, observed at time $t = 0$, no event is observed at time $t = 1$, e_2 is the event `1_D_1_WEAK_SIGNAL` observed at time $t = 2$, and so on.

Definition 2 (Global Trace). *Let $\mathcal{E} := \mathcal{E}_1 \times \ldots \times \mathcal{E}_n$ denote the set of n-vectors of observable events $\mathbf{e}$, with the i-th component e_i of the vector $\mathbf{e}$ corresponding to an event observed at the component $i \in \mathcal{C}$ of the system. We call (global) trace a sequence of n-vectors of events $(\mathbf{e}^0, \ldots, \mathbf{e}^T)$ observed by all the components of the system, i.e., $\mathbf{u} : \mathcal{T} \to \mathcal{E}$.*

Within the global trace $\mathbf{u}$, the symbol $\boldsymbol{\varepsilon} := (\varepsilon_1, \ldots, \varepsilon_n)$ will be used to denote that no event is observed in any of the components. In some cases, we will expect the system to run for an infinite amount of time, and we mark this characteristic by denoting with w and $\mathbf{w}$ respectively local and global infinite traces, and with Ω, the set of global traces.

2.2 Working assumptions

Before addressing the problems of specifying system requirements and monitoring them, several working assumptions must be discussed, some of which are instrumental for keeping the presented work easily understandable, and some of which are oriented in keeping the scope of this work at a manageable size, but might be lifted in the future.

Fixed number of components We assume the system's number of components in $\mathcal{C}$ to be known and constant.

Perfect synchrony By perfect synchrony, we mean that all system components share a global clock [19], i.e., they all observe the same time. This is the most prominent restriction, present in several definitions, and frequently entailed by the word *decentralized* (in contrast with *distributed*, which often also addresses the problem of time synchronization). While this assumption can seem too restrictive, it is a common strategy in the literature to abstract away the synchronization details in contexts where the system is *"fast enough"* in synchronizing without affecting the correctness of the specifications or the monitoring process. See [23] for a discussion on this topic.

Perfect communication By perfect communication, we mean the combination of several features of the message-exchanging interface that result in (i) messages always delivered, (ii) no transfer delays, and (iii) negligible transfer times, which combined imply that messages are always immediately delivered. These features can be significantly limiting in some cases. Still, they are perfectly reasonable for a large set of applications, e.g., in a use case like the one of Fig. 1, other layers of the networking protocols stack are responsible for guaranteeing the delivery,

and the communication usually happens in the order of milliseconds, while the network optimization can typically occur within several seconds. However, future work might be directed at relaxing part of this assumption to widen the applicability to other scenarios, possibly along the lines of [4] that addressed this topic for another temporal logic.

Disjoint events Without loss of generality, we require that for any $i, j \in \mathcal{C}$, $\mathcal{E}_i \cap \mathcal{E}_j = \emptyset$, where $\mathcal{E}_i, \mathcal{E}_j$ are the set of events observed by i and j, respectively. In practice, this means we assume the events observed by different components are disjoint, i.e., no two components can observe the same event. This assumption is frequently made in other works on decentralized monitoring (see [5]).

Single event per component As presented in Defs. 1-2, we require that each component can observe at most one event at each time point. Similarly to *Disjoint events*, this assumption does not bring any loss of generality, because for any set of shared events $\mathcal{E}$, a new set $2^{\mathcal{E}}$ can be constructed where all the possible combinations can be considered as unique events, and from it a new $\mathcal{E}' := \{(i, e) \mid i \leq n, e \in 2^{\mathcal{E}}\}$ can be derived.

2.3 Specifications

Several requirements could be needed for a system like the one in Section 1. For example, the designers of a mesh network might require that, within some time after a device connects, the orientation of the beam must change to a specific value. A simple yet powerful language for expressing such requirements is *linear temporal logic* (LTL) [27], although with some adaptations to support a decentralized evaluation.

Definition 3 (*LTL* Syntax [27]). *A (LTL) formula is any φ belonging to the set Φ, formed according to the following grammar:*

$$\varphi ::= p \quad | \quad \neg\varphi \quad | \quad \varphi \vee \varphi \quad | \quad \mathbf{X}\ \varphi \quad | \quad \varphi\ \mathbf{U}\ \varphi$$

where $p \in \mathcal{E}_i$ for some component $i \in \mathcal{C}$ of the system, denoting an observable local *event. In addition, the following derived operators are commonly defined:* $\top \equiv a \vee \neg a$, $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, $\mathbf{F}\varphi \equiv \top\mathbf{U}\varphi$, $\mathbf{G}\varphi \equiv \neg\mathbf{F}(\neg\varphi)$.

Running example LTL formulae can be used effectively to describe the requirements of the system presented in Fig. 1, e.g., R1 can be decomposed in the following Formulae (1) - (2):

$$\begin{gathered}\texttt{1\_D\_1\_CONNECTED} \wedge \texttt{1\_D\_1\_WEAK\_SIGNAL} \wedge \texttt{2\_D\_1\_IN\_RANGE} \\ \longrightarrow \\ \mathbf{X}\ \texttt{2\_D\_1\_CONNECTED}\end{gathered} \tag{1}$$

meaning that whenever a device is close to AP 2 and the signal of the current AP (component 1) is weak, it should connect to AP 2 as quickly as possible (in the next ($\mathbf{X}$) time unit). Or:

$$\texttt{1\_D\_1\_CONNECTED}\ \mathbf{U}\ (\texttt{2\_D\_1\_CONNECTED} \wedge \neg\texttt{1\_D\_1\_CONNECTED}) \tag{2}$$

to denote that there is no service interruption until (**U**) the device switches connection. Similarly, a possible requirement for beamforming (R2) can be the following:

$$\begin{aligned} &\mathrm{G}(\texttt{1\_A\_1\_HIGH\_TRAFFIC} \wedge \neg \texttt{1\_A\_2\_HIGH\_TRAFFIC}) \\ &\quad \rightarrow \ (\texttt{1\_A\_1\_HIGH\_POWER} \wedge \neg \texttt{1\_A\_2\_HIGH\_POWER}) \end{aligned} \tag{3}$$

where `A_k` denotes the k-th antenna of the AP, meaning that as long as high traffic is experienced on one antenna and not on the other, the AP should set the power delivery of the antenna accordingly. The precise interpretation of *LTL* formulae can be described by the following relation:

Definition 4 (*LTL* semantics [27]). *By* $\models: \Omega \times \mathcal{T} \times \Phi$*, we denote traces and time instants in which a formula holds, in the following way:*

$$\begin{aligned}
(\mathbf{w}, t) &\models p && \textit{iff} && \exists\ i \ \textit{s.t.}\ \mathbf{w}(t)_i = p \\
(\mathbf{w}, t) &\models \neg\varphi && \textit{iff} && (\mathbf{w}, t) \not\models \varphi \\
(\mathbf{w}, t) &\models \varphi_1 \vee \varphi_2 && \textit{iff} && (\mathbf{w}, t) \models \varphi_1 \ \textit{or}\ (\mathbf{w}, t) \models \varphi_2 \\
(\mathbf{w}, t) &\models \mathbf{X}\varphi && \textit{iff} && (\mathbf{w}, t+1) \models \varphi \\
(\mathbf{w}, t) &\models \varphi_1 \mathbf{U} \varphi_2 && \textit{iff} && \exists\ j \in [t, \infty) \textit{s.t.}\ (\mathbf{w}, j) \models \varphi_2 \ \textit{and}\ \forall k \in [t, j) : (\mathbf{w}, k) \models \varphi_1
\end{aligned}$$

Note that, in the monitoring context, we typically consider infinite traces for the semantics, as this is more coherent to monitor a continuously evolving system.

2.4 Monitoring

The fundamental behavior expected by a monitoring system is to provide a verdict as soon as there is enough information to get one. In the context of the specifications we defined, such a verdict is usually a simple 'yes/no' answer to whether the trace of observed events satisfies the formula. This intuitive idea requires special care when the requirements incorporate temporal aspects, as we might not be able to provide such an answer by just considering the events observed so far, nor ever (if the trace is infinite). In the following, we consider the *definitive interpretation* of this idea: we say a trace τ of length T – (i) satisfies a formula φ when the formula would be satisfied for any trace τ' extending the current trace (i.e., such that τ' has the same events in the same order of τ and is of length $T' > T$); conversely (ii) it does not satisfy the formula φ when the formula would not be satisfied for any trace τ' extending the current trace. In other cases, we cannot provide an answer since we do not have enough information to give a definitive answer. Alternative interpretations are commonly used in the literature [14], depending on what best fits the targeted application. To clarify the previous intuitive description, we introduce the concept of *progression function* P, which is a function that maps a formula φ, a trace u and a time point t, to a three-valued logical verdict:

Definition 5 (Progression Function [5]). *We call* progression *the function* $P : \Phi \times \mathcal{E} \rightarrow \Phi_?$ *that maps formulae and events to new formulae, possibly containing the character '?' in place of some subformulae:*

$$\begin{aligned}
P(\top, \mathbf{e}) &= \top & P(\bot, \mathbf{e}) &= \bot \\
P(p, \mathbf{e}) &= \top \ \textit{if}\ \exists\ i \ \textit{s.t.}\ e_i = p, \ \textit{otherwise '?'} \\
P(\neg\varphi, \mathbf{e}) &= \neg P(\varphi, \mathbf{e}) \\
P(\varphi_1 \vee \varphi_2, \mathbf{e}) &= P(\varphi_1, \mathbf{e}) \vee P(\varphi_2, \mathbf{e}) \\
P(X\varphi, \mathbf{e}) &= P(\varphi, \mathbf{e}) \\
P(\varphi_1 \mathrm{U} \varphi_2, \mathbf{e}) &= P(\varphi_2, \mathbf{e}) \vee P(\varphi_1, \mathbf{e}) \wedge \varphi_1 \mathrm{U} \varphi_2
\end{aligned}$$

The character ' **?**' is adopted in Def. 5 to denote the distinguishing aspect of decentralized monitoring against the classical centralized one. In a centralized monitor, instead of ' **?**' we would see $\bot$ every time $\nexists i$ s.t. $\mathbf{e}_i = p$. In a decentralized context, we would continue to see $\bot$ when $\mathbf{e}_i \neq p$, for some component i where $p \in \mathcal{E}_i$. At the same time, for all the others $j \neq i$, we must wait for communication to happen in order to have a Boolean evaluation.

3 Monitors' distribution strategies

The specific distribution strategy of the progression function computations of Def. 5 can significantly impact the resources required to reach a verdict, and, therefore, it can be crucial to determine whether or not a given formula can conveniently be monitored at a given system node. In doing so, all the subformulae of the specification must be mapped to at least one component of the system, and possibly more than one formula is mapped to the same component. We clarify this intuition in the following definition.

Definition 6 (Distribution strategy). *Let $Stfm(\varphi)$ denote the set of subformulae of φ. A monitor distribution strategy is a function $d_\varphi : \mathcal{C} \to 2^{Stfm(\varphi)}$ that maps every component $i \in \mathcal{C}$ to a set of subformulae of φ that must be monitored at that component. Moreover, every subformula must be mapped to at least one component, i.e., $\forall \psi \in Stfm(\varphi)$, $\exists i$ s.t. $\psi \in d_\varphi(i)$.*

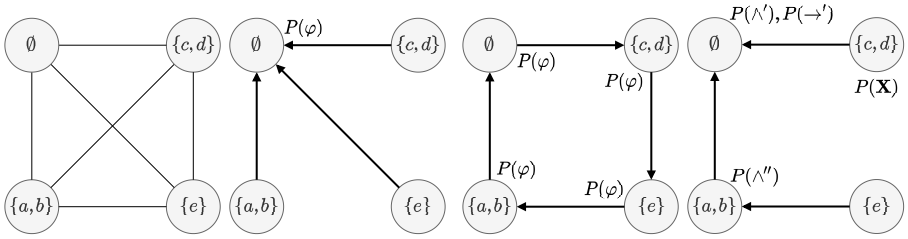
In principle, any use case might have an ideal d_φ of Def. 6 (e.g. a notable case is when a given sub-formula is mapped to multiple components – like it is often the case when developing resilient or traffic-efficient systems). For that reason, the efficiency of the actual run can be significantly impacted by a poor choice of a distribution strategy, for some chosen cost function γ. That said, most of the times practitioners consider a restricted set of cost functions[5]; in the rest of our work, we focus on the following ones:

Definition 7 (Monitoring cost). *We call monitoring cost any of the following:*

- `N. of progressions`*: Computational iterations of the P function required to reach a final verdict.*
- `N. of messages`*: Number of messages exchanged to complete the monitoring task.*
- `Avg. message size`*: Average size of the messages exchanged.*
- `Trace length`*: Evaluation delay between the observed events and the actual result.*

Running example In the network we are considering, we might want to minimize the number of progressions to reduce the computational load on the APs (e.g. when several devices often do real-time calls), or maybe we want to minimize the average message size (e.g., when most of the traffic is for intensive downloads and uploads, like in a video-making setting).

In the experiments of Sec. 5, we consider a specific monitoring cost function for any evaluation. In the following, we denote by γ the chosen monitoring cost function. For a fixed formula φ and a system $\mathcal{C}$, it is not trivial in general to decide which monitoring distribution strategy d_φ to pursue since the variability of the input traces can significantly affect any monitoring cost one is willing to optimize. For example, consider the case of Fig. 2 where alternative strategies are compared for a system similar to the one described in Sec. 1: when observing a trace having $\neg c$ at time 1, orchestration and choreography will lead to a result immediately, while migration will require some extra messages and processing steps, as it has to go through the node $\{e\}$ first. Generally, the *best* distribution strategy can be defined as follows.



(a) Base system (b) Orchestration (c) Migration (d) Choreograpy

Fig. 2. Example of the different distribution strategies for the system of Fig. 1 and Formula 1, i.e. $\varphi = (a \wedge b \wedge c) \rightarrow \mathbf{X}d$, where, for brevity, a replaces `1_D_1_CONNECTED`, b replaces `1_D_1_WEAK_SIGNAL`, c replaces `2_D_1_IN_RANGE`, d replaces `2_D_1_CONNECTED`, and e some other generic event not affected by the current formula. The sets denote the events locally observable by each node; P represents the progression function evaluation, while $'$ and $''$ are used to distinguish the two occurrences (in order of occurrence) of the symbol $\wedge$ in the formula. Arrows denote the communication flow.

Definition 8 (Best distribution strategy). *The best distribution strategy of formula φ, if it exists, is the distribution strategy d_φ that minimizes the accumulated monitoring cost function γ, over all the components i of the system $\mathcal{C}$, for any trace* $\mathbf{u}$.

$$\hat{d_\varphi} := \arg\min_{d_\varphi} \sum_{i \leq n} \gamma(d_\varphi(i), \mathbf{u})$$

Unfortunately, the variability of formulae and observable traces can be so high that the exact computation of Def. 8 becomes prohibitive for any realistic use cases. Therefore, some approximations must be developed to answer this question promptly so that the monitoring task can start without significant penalties. To face this problem, focusing on specific monitoring strategies frequently adopted in practice is convenient (see [11][15]).

- $\mathcal{D}_1$ (*Orchestration*): the class of traditional centralized monitors that assumes (or waits for) global observability. It corresponds to the mapping $0 \mapsto \mathit{Stfm}(\varphi)$.
- $\mathcal{D}_2$ (*Migration*): the class of monitors where the state of a monitor is progressed by a node when some relevant local event is observed or otherwise transferred to the *next* node. It maps the whole specification to all components, i.e., $i \mapsto \varphi, \forall i$.
- $\mathcal{D}_3$ (*Choreography*): the class of monitoring functions where the formula is broken down in a directed acyclic graph over several nodes of the system, and only results are communicated, such that $i \mapsto \{\psi \in Stfm(\varphi) \mid i \in \arg\max_{j \leq n} s_\psi(j)\}$, where $s_\psi(j)$ denotes some scoring rule for ψ at node j (e.g. the number of atomic propositions in the formula that are locally observable).

Fig. 2 shows graphically how messages are distributed according to these strategies. Note that several other communication flows could correspond to migration and choreography. Despite considering a limited set of distribution strategies, Def. 6 is still significantly expensive to compute because of the high variability of the observable traces $\mathbf{u}$. Therefore, some techniques to approximate this function are necessary in practice. We will present them in the next section.

4 Choosing good distribution strategies

We know that the optimal distribution strategy $\hat{d_\varphi}$ depends on the specific number of components of the system n, on the cost function of interest γ, and the formula φ being monitored. In use cases like the one described in our running example, the number of components of the system is fixed for the lifespan of the system, therefore we drop the dependency on the constant n to simplify the presentation. Still, the exact computation of the optimal distribution strategy is costly and too slow to take the timely decisions that are needed to accommodate the system and the requirements of Formulae 1-3. In the following, we consider an alternative approach to approximate the best distribution strategy, where we extract some ahead-of-time knowledge of our system to select the distribution strategy $\mathcal{D}_k$ with the highest probability of being the best for the current system $\mathcal{C}$ and specification φ, more precisely:

$$\text{If } \hat{d_\varphi} = d' \text{ then } d' = \arg\max_{d_j} Pr(\hat{d_\varphi} = d_j | \varphi) \tag{4}$$

The part of (4) after *then* introduces the probability space over distribution functions for a given formula φ on a system of n components. While an exact computation of this probability does not save us from having to consider all the possible traces $\mathbf{u}$, a statistical estimation when d_j is restricted to the strategies $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$ is more easily achievable, and it might also get us very close to the optimal distribution strategy. To pursue this approach, some aspects must be clarified:

- what information we can extract from the formula φ to approximate the best distribution strategy $\hat{d_\varphi}$
- whether we can recommend a good/better strategy for some cost function γ when some information about the current system is available (e.g., we know the cost for some previous trace and distribution strategy of the system);
- whether we can predict the cost function γ for different distribution strategies, given some observations.

We explore these aspects respectively in Sec. 4.1-4.3, while we postpone the actual implementation details to Sec. 5.

4.1 Formula encoding

To properly compare the possible distribution strategies, it is crucial to encode the target formula φ so that some characteristics are kept. (i)It must communicate the primary aspects that could affect the performance of the distribution strategies. (ii) It must provide a compact representation that allows treating in a similar way formulae that are expected to behave similarly in terms of time required to complete the evaluation. (iii) It must be easily exploitable for the recommendation and prediction tasks we are interested in.

Definition 9 (Encoding). *We call encoding $E : \Phi \to \mathbb{R}^k$ a function that maps a specification φ to a vector of k real numbers.*

In [5], the authors define an *urgency level* to support the definition of an evaluation priority among the subformulae to monitor, with the idea that some temporal operators might affect more significantly the time required to complete the evaluation of a formula. That intuition inspires the structural delay encoding presented in Def. 10.

Definition 10 (Structural Delay). *Let T_φ be the syntax tree of the formula φ, and let $T_\varphi[i]$ denote the i-th node of T_φ given a left-to-right top-to-bottom ordering of its nodes. We will call $\overline{E}$:*

$$\overline{E}(\varphi)[i] = \begin{cases} 0 & \text{if } T_\varphi[i] = \top \mid \bot \mid \neg \mid \wedge \\ 1 & \text{if } T_\varphi[i] = p \text{ s.t. } \exists j, p \in \mathcal{E}_j \\ 1 & \text{if } T_\varphi[i] = \mathbf{X} \\ 2 & \text{if } T_\varphi[i] = \mathbf{U} \end{cases} \tag{5}$$

The structural delay encoding of φ.

For a formula $\varphi = (a \vee b)\mathrm{U}\neg c$, the encoding steps are the following (summarized in Fig. 3):

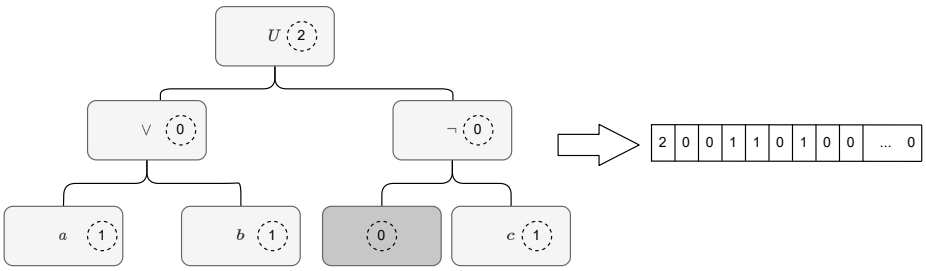


Fig. 3. Encoding steps for a formula $\varphi = (a \vee b)\mathrm{U}\neg c$. First, the tree is built (left), then numbers are extracted (rounded numbers on left), and lastly, the numbers are linearized (right).

1. The formula is first rewritten so only to contain the symbols in Def. 3.
2. A binary tree is generated from φ, where all atomic propositions are leaves, and all logical operators are internal nodes of the tree. Unary operators are represented as binary operators with a dummy leaf.
3. The encoding of Def. 10 is applied, with the respective numeric encoding associated with every symbol of the formula. Dummy leaves are encoded as zeros.
4. The binary tree is linearized as a sequence of dataset features in a classical left-to-right breadth-first traversal. In this process, all the formulae are normalized as if they were of the maximum length by padding the shorter ones with zeros.

4.2 Recommending a better distribution strategy

In scenarios where the monitoring cost $\overline{\gamma}$ of some distribution strategy was observed ahead of time (e.g. when a similar trace was monitored in the past), one might want to check whether a better distribution strategy could be chosen. In this case, the goal is to improve the monitoring cost, thanks to the recent experience. We can formalize the following classification problem:

Definition 11 (Classification problem). *Given the encoding $\overline{E}(\varphi)$ of the formula φ, and a set of distribution strategies (or* classes*) like $\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$, we want to select $\hat{\mathcal{D}}_k$ in the following way:*

$$\hat{\mathcal{D}_k} = \underset{\mathcal{D}_k \;\; k=1,2,3}{\arg\max} \; Pr(\gamma(\mathcal{D}_k, \mathbf{u}) \leq \overline{\gamma} \mid \overline{E}(\varphi), \overline{\gamma})$$

$\hat{\mathcal{D}}_k$ denotes the "best" distribution strategy as the one that maximizes the probability of yielding a lower cost γ for some trace $\mathbf{u}$ that will be observed in future monitoring instances. Note that these traces are unknown, so a training stage is needed to estimate the costs correctly.

4.3 Predicting the monitoring cost

A more challenging problem is predicting the cost of a distribution strategy for a given system without having any information about previous traces. In this case, the only information for the input would be the specification φ, and the goal is to have some prediction of the monitoring cost, which can be used to decide directly which of the options provides the best-predicted cost, and therefore, which strategy to adopt. We can cast this as a regression problem:

Definition 12 (Regression problem). *We want to predict the monitoring cost for a given distribution strategy $\mathcal{D}_k$.*

$$\gamma(\mathcal{D}_k, \mathbf{u}) \approx f_k(\overline{E}(\varphi))$$

The f_k function approximates the γ function that predicts the cost of a given distribution strategy $\mathcal{D}_k$ without simulating it. To correctly approximate it, a proper training stage is required.

Running example Consider our network where R1 and R2 are being monitored as the Formulae (2) and (3) respectively. Which strategy should they follow when the system starts? A good guess could come from substituting Def 12 in Def.11, where, in the absence of information on the trace, an estimation of the monitoring cost γ informs the strategy to follow (say, e.g. an orchestration choreography where the monitoring is happening only on the AP with `id` $= 3$). At some point, a sequence of events `1_A_1_HIGH_TRAFFIC` is recorded (e.g. $\overline{\mathbf{u}} = \{..., \texttt{1\_A\_1\_HIGH\_TRAFFIC}, \texttt{1\_A\_1\_HIGH\_TRAFFIC}\}$). Should the APs change the monitoring strategy to accommodate the new traffic? To decide, one could use Def.11 to assess whether the orchestration is the best strategy for the currently observed traffic (they now can directly observe $\gamma(\mathcal{D}_1, \overline{\mathbf{u}})$). A simulation-based approach, on the contrary, could require several seconds to evaluate the cost for the three distribution strategies we selected, for a single trace, a result that should be replicated with some variation of the trace to make the choice more robust. In the following, we will show how our approach can significantly outperform a direct simulation.

5 Experimental evaluation

We have seen in Section 4 that the problem of deciding a decentralized distribution strategy over a system can be interpreted in several ways, and it might be constrained over several dimensions (e.g., number of messages or message size). In a use case like the one described in Section 1, a change to the decentralization strategy of the monitor has to happen in very little time, not more than a few seconds, and should not impact the computational power of the components, that could be already under-stress for satisfying the bandwidth demand of the connected devices.

5.1 Studied Dataset

To generate a suitable dataset for our models, we used DecentMon [17], a tool for benchmarking decentralized monitors. The data was collected by running DecentMon over randomly generated formulae containing between 2 and 50 logical operators, over

systems of 3, 5, 7, and 9 components, and traces of 100-time steps. The traces were generated by independently sampling each event from a uniform distribution for any component of the system, and the monitor was run over them until an overall simulation memory bound was reached. If the bound is reached without a verdict, the combination of formula-trace is discarded. Our formulae have been randomly generated following the nine popular formula patterns from [13], which were selected for covering 92% of a dataset of 555 real-world LTL specifications and that are frequently adopted as a reference in the literature (see [21,5]). Common examples of patterns from [13], are $\mathrm{G}(q \rightarrow \mathrm{G}(\neg p))$ to denote the "absence" of p (i.e. it is false), or $\mathrm{G}((\neg q) \vee \mathrm{F}(q \wedge \mathrm{F}p))$ to denote the "existence" of p (i.e. at some point it becomes true). The monitoring of each generated formula is then simulated for the three analyzed strategies (orchestration, migration, choreography) over a randomly extracted trace. The final dataset contains approximately 480 thousand entries (i.e., each entry being a combination of *strategy*, *formula*, and *trace*), over a feature space of 709 numerical dimensions, that we used to train and test our models. The dataset has been generated over approximately 40 hours of multi-thread simulations on 12 cores of a 2.1GHz Intel Xeon (Cascade Lake), L1 cache 384KB, L2 cache 48MB, L3 cache 192MB, and 16GB of RAM. For our experiments, an independent model was trained for any combination of the system size and the cost function, using 70% of the dataset for training and the rest for testing the models. For the classification problem, after some preprocessing, the final training dataset contains, on average, 10250 rows per model, while for the regression problem, the final training dataset contains, on average, 76000 rows per model. Check the accompanying artifact for a replication package[4].

5.2 Classification problem

For the classification problem of Sec. 4.2, all *monitoring cost* functions from previous evaluations can be observed, and, therefore, a more informed decision can be made. We used a K-Nearest Neighbors classifier implemented in Python's `scikit-learn` over the three distribution strategies presented on our target dataset. We collected the accuracy and $F1$ scores for the different system sizes. The results are shown in Table 1. The results of Table 1 clearly show that the classification problem is somehow straightforward (contrary to the regression one that we will see later), as is apparent from the very high accuracy scores. This is unsurprising, as it can exploit the information from an actual evaluation for subsequent computations. Nevertheless, the combination with the $F1$ score uncovers certain aspects: (i) Some classes have very low $F1$ scores despite the high accuracy, showing that the dataset was very unbalanced. Note that unbalanced classes mean that the number of entries (e.g., the number of combinations of *formula-trace*) where the best number of progressions is observed with choreography is $>>$ than the one of migration. This is even more apparent by looking at the $-$ signs (particularly when pursuing migration), which report that the difference in the cardinality of that class versus the others was so big that no reasonable $F1$ score could be extracted. This means the optimal strategy consistently leaned towards one distribution strategy for most cost functions (except for the average message size). (ii) We do not observe a significant change in the scores as the number of the components in the system increases; this means that, from the data we have and the hypotheses we made, we can conclude that the size of the system does not play a significant role in moving the convenience from one strategy to another.

[4] Live repository at `https://github.com/ennioVisco/predicting-decentmon`, paper snapshot at [33]

|System| # comp.	Cost function	Accuracy score	F1 score Orchestration	Migration	Choreography
3	N. of progressions	0.956	0.166	-	0.978
	N. of messages	0.873	0.574	0.926	-
	Avg. message size	0.951	0.760	0.235	0.975
	Trace length	0.976	0.043	-	0.924
5	N. of progressions	0.976	0.043	-	0.988
	N. of messages	0.880	0.647	0.928	-
	Avg. message size	0.955	0.757	0.471	0.976
	Trace length	0.895	0.155	-	0.944
7	N. of progressions	0.983	0.172	-	0.991
	N. of messages	0.881	0.680	0.927	-
	Avg. message size	0.955	0.765	0.489	0.977
	Trace length	0.906	0.115	-	0.950
9	N. of progressions	0.980	0.084	-	0.990
	N. of messages	0.878	0.746	0.920	-
	Avg. message size	0.948	0.717	0.575	0.973
	Trace length	0.905	0.180	-	0.950

Table 1. Classification model *accuracy* and $F1$ scores for each system size in terms of the number of components (denoted as **|System|**), and for each class. The columns show the prediction accuracy and the $F1$ metric for each class (we chose not to take the average because of how unbalanced the classes are). Both accuracy and $F1$ can range from 0 (very bad) to 1 (perfect). A '−' sign denotes insufficient data points in that class to compute a reliable $F1$ score.

5.3 Regression problem

Only the formula's encoding is available for the regression problem of Sec. 4.3. On our target dataset, we tested two alternative approximation techniques, a simple linear regression, also regularized via Ridge and Lasso, and a multi-layer perceptron of 1 hidden layer, 100-neurons with a ReLU [20] activation function and no activation function for the output – the default settings from the implementation we used from Python's `scikit-learn`, over the *monitoring cost* functions previously presented. We collected the R^2 scores for the different sizes of the system and for the adopted regression techniques, which denote the proportion of the variation in the monitoring cost that can be predicted from the input data. The results are shown in Table 2.

The results of the regression problem in Table 2, instead, give some insights into the challenges in predicting the actual cost: (i) The R^2 scores of the linear regressors become quickly worse as the system size increases; this fact provides a clear benchmark of the nonlinearity of the problem, as it becomes significantly more challenging to predict it by linear approximation (e.g. for the n. of messages R^2 goes from 0.478 to 0.134 selecting the best performing linear models). (ii) A Multi-Layer Perceptron (MLP) without any specific structuring of the layers performs already very well in several dimensions (e.g., the `Avg. Message Size` is always above 0.900, the `Trace Size` is always above 0.780, and the `# Messages` is above 0.670 except for a system of 3 nodes). The previous points are more clearly depicted in Fig. 4, which compares the predictions for the number of messages for a system of 5 components: the linear models, even the one regularized via Lasso, are not able to define a good prediction model. In contrast, the Multi-Layer Perceptron captures the metric more precisely (they are more spread along the identity diagonal, representing $R^2 = 1$). (iii) The number of progressions

|System| # comp.	Technique	# Progressions	# Messages	Avg. |Message|	|Trace|
3	Linear Regressor	0.697	0.478	0.322	0.738
	L.R. + Ridge Regularization	0.443	0.345	0.608	0.584
	L.R. + Lasso Regularization	0.650	0.284	0.558	0.383
	Multi-Layer Perceptron	0.501	0.564	0.828	0.785
5	Linear Regressor	-1.318	$-4.097 * 10^2$	-9.056	0.607
	L.R. + Ridge Regularization	0.395	0.275	0.666	0.620
	L.R. + Lasso Regularization	0.682	0.209	0.651	0.381
	Multi-Layer Perceptron	0.656	0.679	0.903	0.868
7	Linear Regressor	0.062	0.113	0.130	-1.452
	L.R. + Ridge Regularization	0.628	0.232	0.494	0.638
	L.R. + Lasso Regularization	0.844	0.163	0.479	0.386
	Multi-Layer Perceptron	0.477	0.762	0.910	0.900
9	Linear Regressor	$-5.479 * 10^3$	$-6.064 * 10$	$-1.526 * 10^2$	$-9.567 * 10^2$
	L.R. + Ridge Regularization	0.664	0.204	0.521	0.637
	L.R. + Lasso Regularization	0.797	0.134	0.500	0.388
	Multi-Layer Perceptron	0.397	0.761	0.901	0.906

Table 2. Regression model R^2 scores for the system's different sizes and regression techniques. The R^2 score denotes the proportion of the variation in the monitoring cost that can be predicted from the input data. It ranges from $-\infty$ (very bad) to 1 (perfect).

needed to complete the monitoring is inherently harder to predict for non-regularized models, (e.g. for a system of 9 components $R^2 = 0.397$, is significantly lower than the next worse predicted measure, i.e. $R^2 = 0.761$ for `# Messages`). (iv) While the number of progression gets harder to predict as the system grows, the results seem to get better with Lasso regularization, suggesting that the growth in size helps the model in selecting some dominant features over the others. Lastly, (v) in line with the scores from the classification problem, the size of the system seems not to significantly affect the prediction score (all values are in a range never larger than ± 0.200 as the system grows).

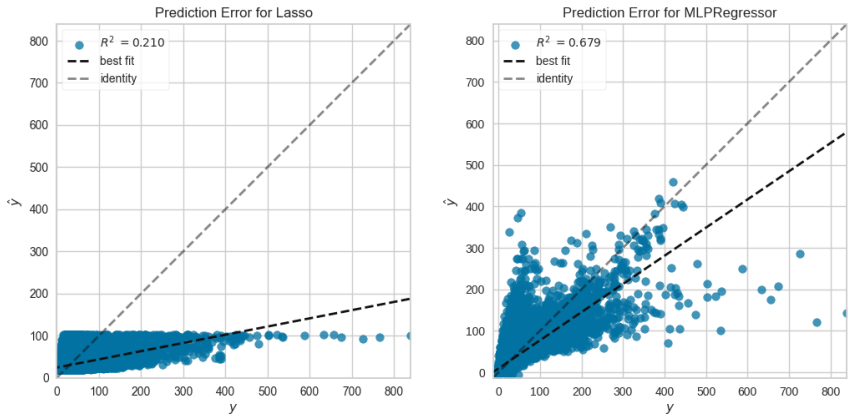


Fig. 4. Comparison of Lasso (left) vs Multi-Layer Perceptron (right) regressors scores for predicting the number of messages exchanged in a system of 5 nodes. y denotes the real value while $\hat{y}$ is the predicted one. In a perfect predictor, the values would be all along the identity line.

5.4 Computational performances

The memory allocation of the trained K-Nearest Neighbors model is approximately 30MBs, while for the Multi-layer Perceptron model, it is 0.9MBs. The simulation time is harder to estimate accurately because of the high variation from the specific parameters in a particular scenario. In our experiments, for a system of 5 components, the simulation time for extracting 5276 combinations of formulae and traces for a single formula pattern from [21] is 134mins (1.52s on average for a single formula/trace combination), while our regressor models can evaluate 35 thousands combinations in 200ms.

5.5 Discussion

The principal argument in support of the proposed learning-based approach, in contrast to a direct simulation-based approach, comes from the runtime benefit of the recommendations, and this is very clear when addressing the classification problem, as well as the regression one via MLP, except perhaps for the `# of Progressions` that proved to be the most challenging metric for both problems. The memory required and testing time from the trained models are negligible (among all the models we tested, the testing time was very small for large sets of values, i.e. 200ms for 35k entries). In contrast, a simulation of the system would require much larger amounts of memory and several seconds in the optimal case where a few traces are sufficient. We now briefly mention some of the pitfalls we encountered during the development of our approach, as they could be guiding directions for future work.

Alternative encodings Before reaching the final encoding of the input data, we tested several alternatives:

- **Formula operators cardinality**: the simplest encoding we found for the input formula φ is to report the number of occurrences of a given logical operator for that formula (e.g. in $\varphi = \mathrm{F}(a \wedge b) \vee (c \wedge d)$, the encoding would be F `= 1,` $\wedge$ `= 2,` $\vee$ `= 1`). While the difference in performance was minimal for elementary formulae, the results were becoming much worse when the formulae had several levels of nesting.
- **Unique operators encoding**: an encoding we tested providing very similar results to the ones showed in Table 1-2 is one where each operator of the formula and each atomic proposition is uniquely identified according to a shared dictionary (e.g., in $\varphi = \mathrm{F}(a \wedge b) \vee (c \wedge d)$, the encoding would be F `= 1,` $\wedge$ `= 2,` $\vee$ `= 3,` a `= -1,` b `= -2,` c `= -3`). The numbers were generally slightly worse, although in that case, the increase in the model's size showed even more minor negative effects. Perhaps this information can be exploited in future, using more clever encodings.
- **Trace events**: To incorporate the information about the evaluated trace (particularly relevant for the Classification problem), we included the exact sequence of events that occurred in every component at every time step. While this encoding was quite costly (we observed an average 30% increase in the preprocessing time of the dataset), it made the classifiers perform worse overall. We explain this by noting that the trace space is much larger than the formula space (and grows faster as the system gets larger), and the classifier cannot generalize well enough to the unseen traces. That said, we do not exclude that some information from the traces could be helpful, perhaps in the form of general statistics (e.g., average number of events per component, average number of events per time step, events frequency, etc.).

6 Conclusions

We proposed a new formalization for optimally distributing decentralized monitors in the context of unknown ideal network configurations framed on the use case of a WiFi mesh network with beamforming. We developed two models based on state-of-the-art simulators and machine-learning techniques to approximate this problem and provide reliable answers instantly at runtime. Several directions could be followed in future work: firstly, some of the initial scenario assumptions could be lifted, most notably considering a weighted network would allow to account for *imperfect communication* and to extend the methodology to several other scenarios. A clear direction of study could come from an end-to-end test of our solution, where a real system is systematically stressed to test how beneficial our approach can be when compared to simulation. Another interesting direction would be to allow for dynamic distribution strategies. These strategies could exploit contextual information (e.g., deviations from the expected costs or extra not-predicted information) to guide a reconfiguration, effectively reacting to environmental changes. Lastly, in the direction of perfecting the predictions, alternative layouts of the Multi-Layer Perceptron (e.g. more hidden layers), or alternative learning techniques (e.g. Support Vector Machine-based ones) could provide substantial improvements.

Acknowledgments

The authors acknowledge funding from the Austrian Science Fund (FWF) for the project "High-dimensional statistical learning: New methods to advance economic and sustainability policies" (ZK 35), jointly carried out by the University of Klagenfurt, the University of Salzburg, TU Wien, and the Austrian Institute of Economic Research (WIFO). The work was also partially supported by the WWTF project ICT22-023. The work was also partially supported by SEVERITAS ANR-20-CE39-0009 of the French national research agency.

References

1. Asudani, D.S., Nagwani, N.K., Singh, P.: Impact of word embedding models on text analytics in deep learning environment: a review. Artificial Intelligence Review pp. 1 – 81 (2023), https://api.semanticscholar.org/CorpusID:257098478
2. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification. LNCS, Springer (2018)
3. Bartocci, E., Mateis, C., Nesterini, E., Nickovic, D.: Survey on mining signal temporal logic specifications. Inf. Comput. **289**(Part), 104957 (2022)
4. Basin, D., Klaedtke, F., Zalinescu, E.: Failure-aware Runtime Verification of Distributed Systems. In: Harsha, P., Ramalingam, G. (eds.) 35th IARCS Conf. on Found. of Soft. Tech. and Theor. Comp. Sc. (FSTTCS 2015). Dagstuhl, Germany (2015)
5. Bauer, A., Falcone, Y.: Decentralised LTL monitoring. In: Proc. of FM 2012. pp. 85–100. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
6. Bombara, G., Belta, C.: Offline and online learning of signal temporal logic formulae using decision trees. ACM Trans. Cyber Phys. Syst. **5**(3), 22:1–22:23 (2021)
7. Bonakdarpour, B., Fraigniaud, P., Rajsbaum, S., Rosenblueth, D., Travers, C.: Decentralized asynchronous crash-resilient runtime verification. J. ACM **69**(5) (2022)
8. Bornholt, J., et al.: Using lightweight formal methods to validate a key-value storage node in amazon s3. In: SOSP 2021 (2021)

9. Bortolussi, L., Cairoli, F., Paoletti, N., Smolka, S.A., Stoller, S.D.: Neural predictive monitoring and a comparison of frequentist and bayesian approaches. Int. J. Softw. Tools Technol. Transf. **23**(4), 615–640 (2021)
10. Bortolussi, L., Gallo, G.M., Kretínský, J., Nenzi, L.: Learning model checking and the kernel trick for signal temporal logic on stochastic processes. In: Proc. of TACAS 2022. LNCS, vol. 13243, pp. 281–300. Springer (2022)
11. Colombo, C., Falcone, Y.: Organising LTL monitors over distributed systems with a global clock. In: Proc. of RV 2014. pp. 140–155. Springer (2014)
12. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Formal Methods in Sys. Des. (2015)
13. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st international conference on Software engineering. ICSE '99, Association for Computing Machinery (1999)
14. Eisner, C., et al.: Reasoning with temporal logic on truncated paths. In: Proc. of CAV 2003. pp. 27–39. Springer Berlin Heidelberg (2003)
15. El-Hokayem, A., Falcone, Y.: On the monitoring of decentralized specifications: Semantics, properties, analysis, and simulation. ACM Trans. Softw. Eng. Methodol. (2020)
16. Falcone, Y.: On decentralized monitoring. In: Verification and Evaluation of Computer and Communication Systems. Springer International Publishing (2022)
17. Falcone, Y.: DecentMon: an OCaml benchmark for decentralised monitoring of LTL (2023), `https://gricad-gitlab.univ-grenoble-alpes.fr/falconey/decentmon`
18. Fraigniaud, P., Rajsbaum, S., Travers, C.: On the number of opinions needed for fault-tolerant run-time monitoring in distributed systems. In: Proc. of RV 2014. Springer International Publishing (2014)
19. Francalanza, A., Pérez, J.A., Sánchez, C.: Runtime Verification for Decentralised and Distributed Systems, pp. 176–210. Springer International Publishing, Cham (2018)
20. Fukushima, K.: Cognitron: A self-organizing multilayered neural network. Biological Cybernetics **20**(3), 121–136 (Sep 1975)
21. Gruhn, V., Laue, R.: Patterns for timed property specifications. Electronic Notes in Theoretical Computer Science **153**(2), 117–133 (2006), proc. of QAPL 2005
22. HABS VA, 2-ALB, .s..o..: Company house, state route 719, alberene, albemarle county, va, habs va,2-alb,1- (sheet 2 of 8)
23. Jantsch, A.: chapter four - the synchronous model of computation. In: Modeling Embedded Systems and SoC's. Systems on Silicon, Morgan Kaufmann (2003)
24. Mamouras, K., Chattopadhyay, A., Wang, Z.: A compositional framework for quantitative online monitoring over continuous-time signals. In: Proc. of RV 2021. Springer International Publishing (2021)
25. Nenzi, L., Silvetti, S., Bartocci, E., Bortolussi, L.: A robust genetic algorithm for learning temporal specifications from data. In: Proc. of QEST 2018. Springer (2018)
26. Neto, W.L., Moreira, M.T., Amarù, L.G., Yu, C., Gaillardon, P.E.: Read your circuit: Leveraging word embedding to guide logic optimization. 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC) pp. 530–535 (2021), `https://api.semanticscholar.org/CorpusID:231730639`
27. Pnueli, A.: The temporal logic of programs. 18th Annual Symposium on Foundations of Computer Science (sfcs 1977) pp. 46–57 (1977)
28. Racharak, T.: On approximation of concept similarity measure in description logic elh with pre-trained word embedding. IEEE Access **9**, 61429–61443 (2021), `https://api.semanticscholar.org/CorpusID:233433689`

29. Rufino, J.: Towards integration of adaptability and non-intrusive runtime verification in avionic systems. SIGBED Rev. **13**(1), 60–65 (mar 2016)
30. Rungta, N.: A billion SMT queries a day (invited paper). In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification. Springer International, Cham (2022)
31. Sánchez, C., et al.: A survey of challenges for runtime verification from advanced application domains (beyond software). Formal Methods in System Design **54**(3), 279–335 (Nov 2019)
32. Seshia, S.A., Sadigh, D., Sastry, S.S.: Toward verified artificial intelligence. Commun. ACM **65**(7) (jun 2022)
33. Visconti, E., Bartocci, E., Falcone, Y., Nenzi, L.: Predicting Decentmon (Source code + Docker Image) (3 2024). https://doi.org/10.6084/m9.figshare.25465243.v2
34. Visconti, E., Bartocci, E., Loreti, M., Nenzi, L.: Online monitoring of spatio-temporal properties for imprecise signals. In: Proc. of MEMOCODE 2021. ACM (2021)