Decentralized Runtime Verification of Stream-based Partially-Synchronous Distributed System

ANONYMOUS AUTHOR(S)

Distributed runtime verification is concerned with monitoring of a distributed system with respect to its formal specification. With components of distributed systems often geographically separated, maintaining a common global clock is often not feasible. Additionally, verification of modern distributed system often include monitoring the system specification with respect to aggregated data generated from a number of components poise added challenges. In this paper, we first present a general technique for runtime monitoring of distributed applications whose behavior can be modeled as input/output *streams* with an internal computation module in the partially synchronous semantics, where an imperfect clock synchronization algorithm is assumed. Second, we propose a generalized stream-based decentralized runtime verification technique. We also rigorously evaluate our algorithm on extensive synthetic experiments and several industrial control systems and aircraft SBS message datasets.

CCS Concepts: • **Theory of computation → Formal languages and automata theory**; **Logic and verification**; • **Security and privacy → Software and application security**; • **Computing methodologies → Distributed algorithms**.

Additional Key Words and Phrases: Distributed Systems, Stream-based specification, partially-synchronous systems, industrial control system

ACM Reference Format:
Anonymous Author(s). 2018. Decentralized Runtime Verification of Stream-based Partially-Synchronous Distributed System. In . ACM, New York, NY, USA, 25 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Modern distributed system has grown bigger and more complex over the years. They typically consists of multiple components that are distributed over a large geographical area, solving a common problem. An exhaustive verification technique such as model checking come at a high cost in terms of time, computation resource, and expertise. On the other hand, testing is not exhaustive and only examines a subset of behaviors. Moreover, both model checking and testing approaches often overlook bugs due to unanticipated stimuli from the environment, hardware and external faults.

In this paper, we advocate for a runtime verification (RV) approach, to monitor the behavior of a distributed system with respect to a formal specification. Applying RV to multiple components of a distributed RV, where a centralized or decentralized monitor(s) observe the behavior of a distributed system in which the processes do not share a global clock. Although RV deals with finite executions, the lack of a common global clock prohibits it from having a total ordering of events in a distributed setting. In other words, the monitor can only form a partial ordering of events which may yield different evaluations. Enumerating all possible interleavings of the system at runtime incurs in an exponential blowup, making the approach not scalable.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06...$15.00
https://doi.org/XXXXXXX.XXXXXXX

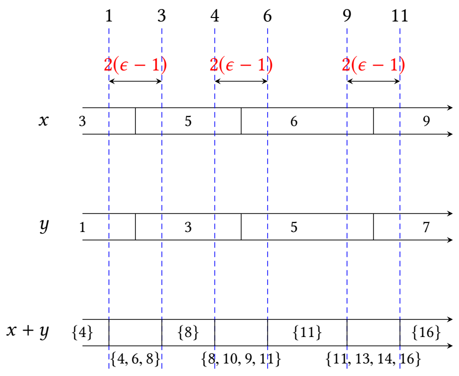


Fig. 1. Partially Synchronous LOLA

Checking for satisfaction of system properties is not usually not enough to study a distributed system to make well-informed, sound decision. For example in a water treatment plant, the level of water in a tank should reflect the net-amount of water flowing-in and flowing-out of the tank in addition to the previously reported water level. To address monitoring of such system, we propose an effective, sound and complete solution to distributed RV for the popular *stream-based* specification language Lola [10]. Compared to other temporal logic, Lola can describe both correctness/failure assertions along with statistical measures that can be used for system profiling and coverage analysis. To present a high level of Lola example, consider two input streams x and y and a output stream, *sum* as shown in Fig. 1. Stream x has the value 3 until time instance 2 when it changes to 5 and so on.

```
input x:int
input y:int
output sum := x+y
```

We consider a fault proof decentralized set of monitors where each monitor only has a partial view of the system and has no access to a global clock. In order to limit the blow-up of states posed by the absence of the global clock, we make a practical assumption about the presence of a bounded clock skew ϵ between all the local clocks, guaranteed by a clock synchronization algorithm (like NTP [21]). This setting is known to be *partially synchronous*. As can be seen in Fig. 1, any two events less than $\epsilon = 2$ time apart is considered to be concurrent and thus the non-determinism of the time of occurrence of each event is restricted to $\epsilon - 1$ on either side. When attempting to evaluate the output stream *sum*, we need to take into consideration all the possible time of occurrence of the values. For example, when evaluating the value of *sum* at time 1, we need to consider the value of x (resp. y) as 3 and 5 (resp. 1 and 3) which evaluates to 4, 6 and 8. The same can be observed for evaluations across all time instances.

Our first contribution in this paper is introducing a partially synchronous semantics for Lola compared to the synchronous semantics in [11]. In other words, we define Lola which takes into consideration a clock-skew of ϵ when evaluating a stream expression. Second, we introduce an SMT-based associated equation rewriting technique over a partially observable distributed system, which takes into consideration the values observed by the monitor and rewrites the associated equation. The monitors are able to communicate within themselves and are able to resolve the partially evaluated equations into completely evaluated ones.

We have proved the correctness of our approach and the upper and lower bound of the message complexity. Additionally, we have completely implemented our technique and report the results of rigorous synthetic experiments, as well as monitoring correctness and aggregated results of several Industrial Control Systems (ICS). As identified in [2], most attacks on ICS components try to alter the value reported to the Programmable Logic Controller (PLC) in-order to make the PLC behave erroneously. Monitoring of ICS is a widely studies subject, with multiple tools from Honeywell [12], Siemens [1, 30], etc. being developed for monitoring large manufacturing and production units. These tools mostly incorporate a machine learning based approach, which is often susceptible to false negative results.

Through our approach, we were able to detect these attacks in-spite of the clock asynchrony among the different components with deterministic guarantee. We also argue that our approach was able to evaluate system behavior aggregates that makes studying these system easier by the human operator. Unlike machine learning based approaches (e.g., [6, 25, 26]), our approach will never raise false negatives. We put our monitoring technique to test, studying the effects of different parameters on the runtime and size of the message sent from one monitor to other and report on each of them.

Organization. Section 2 presents the background concepts. Partially synchronous Lola and the formal problem statement are introduced in Section 3. Our RV technique is collectively presented in Sections Section 4 – 7 followed by the experimental results in Section 8. Related work is discussed in Section 9 before we make concluding remarks in Section 10. Details of syntax of Lola, proofs of correctness and more details about the ICS case studies can be found in the supplement document A.

2 PRELIMINARIES – STREAM-BASED SPECIFICATION LANGUAGE (LOLA) [10]

A Lola [10] specification describes the computation of output streams given a set of input streams. A *stream* α of type T is a finite sequence of values, $t \in \mathsf{T}$. Let $\alpha(i)$, where $i \geq 0$, denote the value of the stream at time stamp i. We denote a stream of finite length (resp. infinite length) by T^* (resp. T^ω).

Definition 2.1. A Lola specification is a set of equations over typed stream variables of the form:

$$\begin{aligned} s_1 &= e_1(t_1, \cdots, t_m, s_1, \cdots, s_n) \\ &\vdots \quad \vdots \\ s_n &= e_n(t_1, \cdots, t_m, s_1, \cdots, s_n) \end{aligned}$$

where $s_1, s_2, \cdots, s_n$ are called the *dependent variables*, $t_1, t_2, \cdots, t_m$ are called the *independent variables*, and $e_1, e_2, \cdots, e_n$ are the *stream expressions* over $s_1, \cdots, s_n, t_1, \cdots, t_m$. □

Typically, *Input* streams are referred to as independent variables, whereas *output* streams are referred as dependent variable. For example, consider the following Lola specification, where t_1 and t_2 are independent stream variables of type boolean and t_3 is an independent stream variable of type integer.

$$\begin{aligned} s_1 &= \texttt{true} \\ s_2 &= t_1 \vee (t_3 \leq 1) \\ s_3 &= \texttt{ite}(s_3, s_4, s_4 + 1) \\ s_4 &= s_4[-1, 0] + (t_3 \mod 2) \end{aligned}$$

where, ite is the abbreviated form of *if-then-else* and stream expressions s_7 and s_8 refers to the stream t_1 with an offset of $+1$ and -1, respectively. Due to space constrains we present the full syntax of LOLA in supplement document Section A.1.

The semantics of LOLA specifications is defined in terms of the evaluation model, which describes the relation between input and output streams.

Definition 2.2. Given a LOLA specification φ over independent variables, $t_1, \cdots, t_m$, of type, $\mathsf{T}_1, \cdots, \mathsf{T}_m$, and dependent variables, $s_1, \cdots, s_n$ with type, $\mathsf{T}_{m+1}, \cdots, \mathsf{T}_{m+n}$, let $\tau_1, \cdots, \tau_m$ be the streams of length $N+1$, with τ_i of type T_i. The tuple $\langle \alpha_1, \cdots, \alpha_n \rangle$ of streams of length $N+1$ is called the *evaluation model*, if for every equation in φ

$$s_i = e_i(t_1, \cdots, t_m, s_1, \cdots, s_n)$$

$\langle \alpha_1, \cdots, \alpha_n \rangle$ satisfies the following associated equations:

$$\alpha_i(j) = val(e_i)(j) \;\text{ for } (1 \leq i \leq n) \wedge (0 \leq j \leq N)$$

where $val(e_i)(j)$ is defined as follows. For the base cases:

$$\begin{aligned} val(c)(j) &= c \\ val(t_i)(j) &= \tau_i(j) \\ val(s_i)(j) &= \alpha_i(j) \end{aligned}$$

For the inductive cases, where f is a function (e.g., arithmetic):

$$\begin{aligned} val\Big(f(e_1, \cdots, e_k)\Big)(j) &= f\Big(val(e_1)(j), \cdots, val(e_k)(j)\Big) \\ val\Big(\mathtt{ite}(b, e_1, e_2)\Big)(j) &= \text{if } val(b)(j) \text{ then } val(e_1)(j) \text{ else } val(e_2)(j) \\ val(e[k,c])(j) &= \begin{cases} val(e)(j+k) & if\, 0 \leq j+k \leq N \\ c & otherwise \end{cases} \blacksquare \end{aligned}$$

The set of all equations associated with φ is noted by φ_α.

Definition 2.3. A *dependency graph* for a LOLA specification, φ is a weighted and directed graph $G = \langle V, E \rangle$, with vertex set $V = \{s_1, \cdots, s_n, t_1, \cdots, t_m\}$. An edge $e : \langle s_i, s_k, w \rangle$ (resp. $e : \langle s_i, t_k, w \rangle$) labeled with a weight w is in E iff the equation for $\alpha_i(j)$ in φ_α contains $\alpha_k(j+w)$ (resp. $\tau_k(j+w)$) as a subexpression. Intuitively, an edge records that s_i at a particular position depends on the value of s_k (resp. t_k), offset by w positions.

Given a set of synchronous input streams $\{\alpha_1, \alpha_2, \cdots, \alpha_m\}$ of respective type $\mathbb{T} = \{\mathsf{T}_1, \mathsf{T}_2, \cdots, \mathsf{T}_m\}$ and a LOLA specification, φ, we evaluate the LOLA specification, given by:

$$(\alpha_1, \alpha_2, \cdots, \alpha_m) \models_S \varphi$$

given the above semantics, where $\models_S$ denotes the synchronous evaluation.

3 PARTIALLY SYNCHRONOUS LOLA

In this section, we extend the semantics of LOLA to one that can accommodate reasoning about distributed systems.

3.1 Distributed Streams

Here, we refer to a global clock which will act as the "real" timekeeper. It is to be noted that the presence of this global clock is just for theoretical reasons and it is not available to any of the individual streams.

We assume a *partially synchronous* system of n streams, denoted by $\mathcal{A} = \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$. For each stream α_i, where $i \in [1, |\mathcal{A}|]$, the local clock can be represented as a monotonically increasing function $c_i : \mathbb{Z}_{\geq 0} \longrightarrow \mathbb{Z}_{\geq 0}$, where $c_i(\mathcal{G})$ is the value of the local clock at global time $\mathcal{G}$. Since we are dealing with discrete-time systems, for simplicity and without loss of generality, we represent time with non-negative integers $\mathbb{Z}_{\geq 0}$. For any two streams α_i and α_j, where $i \neq j$, we assume:

$$\forall \mathcal{G} \in \mathbb{Z}_{\geq 0}.\ |\ c_i(\mathcal{G}) - c_j(\mathcal{G})\ | < \epsilon,$$

where $\epsilon > 0$ is the maximum clock skew. The value of ϵ is constant and is known (e.g., to a monitor). This assumption is met by the presence of an off-the-shelf clock synchronization algorithm, like NTP [21], to ensure bounded clock skew among all streams. The local state of stream α_i at time σ is given by $\alpha_i(\sigma)$, where $\sigma = c_i(\mathcal{G})$, that is the local time of occurrence of the event at some global time $\mathcal{G}$.

Definition 3.1. A *distributed stream* consisting of $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ streams of length $N + 1$ is represented by the pair $(\mathcal{E}, \rightsquigarrow)$, where $\mathcal{E}$ is a set of all local states (i.e., $\mathcal{E} = \cup_{i \in [1,n], j \in [0,N]} \alpha_i(j)$) partially ordered by Lamport's happened-before ($\rightsquigarrow$) relation [16], subject to the partial synchrony assumption:

- For every stream α_i, $1 \leq i \leq |\mathcal{A}|$, all the events happening on it are totally ordered, that is,
$$\forall i, j, k \in \mathbb{Z}_{\geq 0} : (j < k) \rightarrow (\alpha_i(j) \rightsquigarrow \alpha_i(k))$$
- For any two streams α_i and α_j and two corresponding events $\alpha_i(k), \alpha_j(l) \in \mathcal{E}$, if $k + \epsilon < l$ then, $\alpha_i(k) \rightsquigarrow \alpha_j(l)$, where ϵ is the maximum clock skew.
- For events, e, f, and g, if $e \rightsquigarrow f$ and $f \rightsquigarrow g$, then $e \rightsquigarrow g$. □

Definition 3.2. Given a distributed stream $(\mathcal{E}, \rightsquigarrow)$, a subset of events $C \subseteq \mathcal{E}$ is said to form a *consistent cut* if and only if when C contains an event e, then it should also contain all such events that happened before e. Formally,

$$\forall e, f \in \mathcal{E}.(e \in C) \wedge (f \rightsquigarrow e) \longrightarrow f \in C. \blacksquare$$

The frontier of a consistent cut C, denoted by front(C) is the set of all events that happened last in each stream in the cut. That is, front(C) is a set of $\alpha_i(last)$ for each $i \in [1, |\mathcal{A}|]$ and $\alpha_i(last) \in C$. We denote $\alpha_i(last)$ as the last event in α_i such that $\forall \alpha_i(\sigma) \in C.(\alpha_i(\sigma) \neq \alpha_i(last)) \longrightarrow (\alpha_i(\sigma) \rightsquigarrow \alpha_i(last))$.

3.2 Partially Synchronous Lola

We define the semantics of Lola specifications for partially synchronous distributed streams in terms of the evaluation model. The absence of a common global clock among the stream variables and the presence of the clock synchronization makes way for the output stream having multiple values at any given time instance. Thus, we update the evaluation model, so that $\alpha_i(j)$ and $val(t_i)(j)$ are now defined by *sets* rather than just a single value. This is due to nondeterminism caused by partial synchrony, i.e., the bounded clock skew ϵ.

Definition 3.3. Given a Lola [10] specification φ over independent variables, $t_1, \cdots, t_m$ of type $\mathsf{T}_1, \cdots, \mathsf{T}_m$ and dependent variables, $s_1, \cdots, s_n$ of type $\mathsf{T}_{m+1}, \cdots, \mathsf{T}_{m+n}$ and $\tau_1, \cdots, \tau_m$ be the streams of length $N + 1$, with τ_i of type T_i. The tuple of streams $\langle \alpha_1, \cdots, \alpha_n \rangle$ of length $N + 1$

with corresponding types is called the evaluation model in the partially synchronous setting, if for every equation in φ:

$$s_i = e_i(t_1, \cdots, t_m, s_1, \cdots, s_n),$$

$\langle \alpha_1, \cdots, \alpha_n \rangle$ satisfies the following associated equations:

$$\alpha_i(j) = \Big\{ val(e_i)(k) \mid \max\{0, j-\epsilon+1\} \le k \le \min\{N, j+\epsilon-1\} \Big\}$$

where $val(e_i)(j)$ is defined as follows. For the base cases:

$$\begin{aligned} val(c)(j) &= \{c\} \\ val(t_i)(j) &= \Big\{ \tau_i(k) \mid \max\{0, j-\epsilon+1\} \le k \le \min\{N, j+\epsilon-1\} \Big\} \\ val(s_i)(j) &= \alpha_i(j) \end{aligned}$$

For the inductive cases:

$$val\Big(f(e_1, \cdots, e_p)\Big)(j) = \Big\{ f(e'_1, \cdots, e'_p) \mid e'_1 \in val(e_1)(j), \cdots, e'_p \in val(e_p)(j) \Big\}$$

$$val(\texttt{ite}(b, e_1, e_2))(j) = \begin{cases} val(e_1)(j) & \texttt{true} \in val(b)(j) \\ val(e_2)(j) & \texttt{false} \in val(b)(j) \end{cases}$$

$$val(e[k,c])(j) = \begin{cases} val(e)(j+k) & \textit{if}\ 0 \le j+k \le N \\ c & \textit{otherwise} \end{cases}$$

□

Example 3.4. Consider the Lola specification, φ, over the independent boolean variables `read` and `write`:

```
input read:bool
input write:bool
output countRead := ite(read, countRead[-1,0] + 1, countRead[-1,0])
output countWrite := ite(write, countWrite[-1,0] + 1, countWrite[-1,0])
output check := (countWrite - countRead) <= 2
```

In Fig. 2, we have two input stream *read* and *write* which denotes the time instances where the corresponding events take place. It can be imagined that *read* and *write* are streams of type boolean with `true` values at time instances 4, 6, 7 and 2, 3, 5, 6 and `false` values at all other time instances respectively. We evaluate the above mentioned Lola specification considering a time synchronization constant, $\epsilon = 2$. The corresponding associated equations, φ_α, are:

$$countRead(j) = \begin{cases} \texttt{ite}(read, 1, 0) & j = 0 \\ \texttt{ite}\Big(read, countRead(j-1)+1, countRead(j)\Big) & j \in [1, N) \end{cases}$$

$$countWrite(j) = \begin{cases} \texttt{ite}(write, 1, 0) & j = 0 \\ \texttt{ite}\Big(write, countWrite(j-1)+1, countWrite(j)\Big) & j \in [1, N) \end{cases}$$

$$check(j) = \Big(countWrite(j) - countRead(j)\Big) \le 2$$

Similar to the synchronous case, evaluation of the partially synchronous Lola specification involves creating the dependency graph.

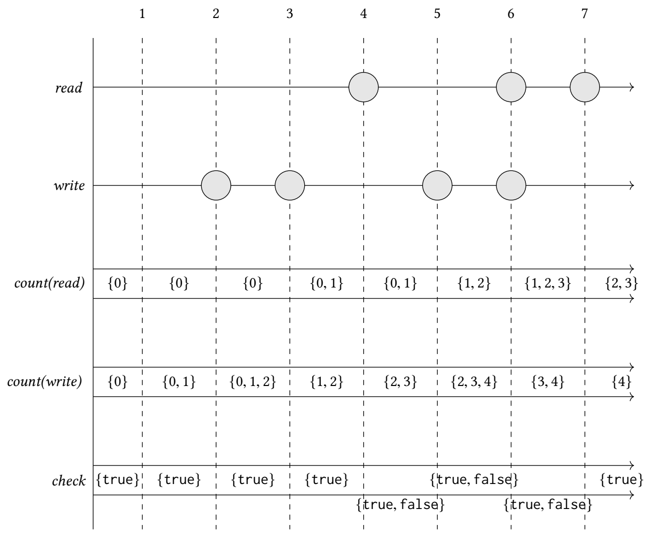


Fig. 2. Partially Synchronous LOLA Example

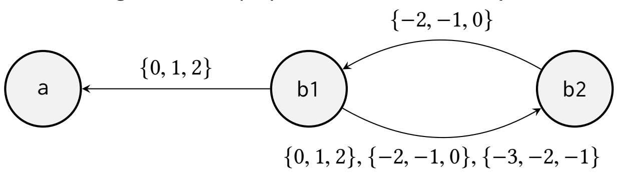


Fig. 3. Dependency Graph Example

Definition 3.5. A dependency graph for a LOLA specification, φ is a weighted directed multi-graph $G = \langle V, E\rangle$, with vertex set $V = \{s_1, \cdots, s_n, t_1, \cdots, t_m\}$. An edge $e : \langle s_i, s_k, w\rangle$ (resp. $e : \langle s_i, t_k, w\rangle$) labeled with a weight $w = \{\omega \mid p - \epsilon < \omega < p + \epsilon\}$ is in E iff the equation for $\alpha_i(j)$ contains $\alpha_k(j + p)$ (resp. $\tau_k(j + p)$) as a sub-expression, for some j and offset p. □

Intuitively, the dependency graph records that evaluation of a s_i at a particular position depends on the value of s_k (resp. t_k), with an offset in w. It is to be noted that there can be more than one edge between a pair of vertex (s_i, s_k) (resp. (s_i, t_k)). Vertices labeled by t_i do not have any outgoing edges.

Example 3.6. Consider the LOLA specification over the independent integer variable a:

```
input a : uint
output b1 := b2[1, 0] + ite(b2[-1,7] <= a[1, 0], b2[-2,0], 6)
output b2 := b1[-1,8]
```

Its dependency graph, shown in Fig. 3 for $\epsilon = 2$, has 1 edge from b1 to a with a weight $\{0, 1, 2\}$. Similarly, there are 3 edges from b1 to b2 with weights $\{0, 1, 2\}$, $\{-2, -1, 0\}$ and $\{-3, -2, -1\}$ and 1 edge from b2 to b1 with a weight of $\{-2, -1, 0\}$

Given a set of partially synchronous input streams $\{\alpha_1, \alpha_2, \cdots, \alpha_{|\mathcal{A}|}\}$ of respective type $\mathbb{T} = \{\mathsf{T}_1, \mathsf{T}_2, \cdots, \mathsf{T}_{|\mathcal{A}|}\}$ and a LOLA specification, φ, the evaluation of φ is given by

$$(\alpha_1, \alpha_2, \cdots, \alpha_{|\mathcal{A}|}) \models_{PS} \varphi$$

Algorithm 1 Behavior of a Monitor M_i, for $i \in [1, |\mathcal{M}|]$

```
1: for $j = 0$ to $N$ do
2:   Let $(\mathcal{E}_i, \rightsquigarrow_i)_j$ be the partial distributed stream view of $M_i$
3:   $LS_j \leftarrow \big[(\mathcal{E}, \rightsquigarrow) \models_{PS} \varphi_\alpha\big]$
4:   Send: broadcasts symbolic view $LS_j$
5:   Receive: $\Pi_j \leftarrow \{LS_j^k \mid 1 \leq k \leq \mathcal{M}\}$
6:   Compute: $LS_{j+1} \leftarrow LC(\Pi_j)$
7: end for
```

where, $\models_{PS}$ denotes the partially synchronous evaluation.

4 DECENTRALIZED MONITORING ARCHITECTURE

4.1 Overall Picture

We consider a decentralized online monitoring system comprising of a fixed number of $|\mathcal{M}|$ reliable monitor processes $\mathcal{M} = \{M_1, M_2, \cdots, M_{|\mathcal{M}|}\}$ that can communicate with each other by sending and receiving messages through a complete point-to-point bidirectional communication links. Each communication link is also assumed to be reliable, i.e., there is no loss or alteration of messages. Similar to the distributed system under observation, we assume the clock on the individual monitors are asynchronous, with clock synchronization constant = ϵ_M.

Throughout this section we assume that the global distributed stream consisting of complete observations of $|\mathcal{A}|$ streams is only partially visible to each monitor. Each monitor process locally executes an identical sequential algorithm which consists of the following steps (we will generalize this approach in Section 7). In other words, an evaluation iteration of each monitor consists of the following steps:

(1) Reads the a subset of $\mathcal{E}$ events (visible to M_i) along with the corresponding time and valuation of the events, which results in the construction of a *partial distributed stream*;
(2) Each monitor evaluates the LOLA specification φ given the partial distributed stream;
(3) Every monitor, broadcasts a message containing rewritten associated equations of φ, denoted LS, and
(4) Based on the message received containing associated equations, each monitor amalgamates the observations of all the monitors to compose a set of associated equations. After a evaluation iteration, each monitor will have the same set of associated equations to be evaluated on the upcoming distributed stream.

The message sent from monitor M_i at time π to another monitor M_j, for all $i, j \in [1, |\mathcal{M}|]$, during a evaluation iteration of the monitor is assumed to reach latest by time $\pi + \epsilon_M$. Thus, the length of an *evaluation iteration* k can be adjusted to make sure the message from all other monitors reach before the start of the next evaluation iteration.

4.2 Detailed Description

We now explain in detail the computation model (see Algorithm 1). Each monitor process $M_i \in \mathcal{M}$, where $i \in [1, |\mathcal{M}|]$, attempts to read $e \in \mathcal{E}$, given the distributed stream, $(\mathcal{E}, \rightsquigarrow)$. An event can either be observable, or not observable. Due to distribution, this results in obtaining a partial distributed stream $(\mathcal{E}_i, \rightsquigarrow)$ defined below.

Definition 4.1. Let $(\mathcal{E}, \rightsquigarrow)$ be a distributed stream. We say that $(\mathcal{E}', \rightsquigarrow)$ is a *partial distributed stream* for $(\mathcal{E}, \rightsquigarrow)$ and denote it by $(\mathcal{E}', \rightsquigarrow) \sqsubseteq (\mathcal{E}, \rightsquigarrow)$ iff $\mathcal{E}' \subseteq \mathcal{E}$ (the happened before relation is obviously preserved). □

We now tie partial distributed streams to a set of decentralized monitors and the fact that decentralized monitors can only partially observe a distributed stream. First, all un-observed events is replaced by $\natural$, i.e., for all $\alpha_i(\sigma) \in \mathcal{E}$ if $\alpha_i(\sigma) \notin \mathcal{E}_i$ then $\mathcal{E}_i = \mathcal{E}_i \cup \{\alpha_i(\sigma) = \natural\}$.

Definition 4.2. Let $(\mathcal{E}, \rightsquigarrow)$ be a distributed stream and $\mathcal{M} = \{M_1, M_2, \cdots, M_{|\mathcal{M}|}\}$ be a set of monitors, where each monitor M_i, for $i \in [1, |\mathcal{M}|]$ is associated with a partial distributed stream $(\mathcal{E}_i, \rightsquigarrow) \sqsubseteq (\mathcal{E}, \rightsquigarrow)$. We say that these monitor observations are consistent if

- $\forall e \in \mathcal{E}.\exists i \in [1, |\mathcal{M}|].e \in \mathcal{E}_i$, and
- $\forall e \in \mathcal{E}_i.\forall e' \in \mathcal{E}_j.(e = e' \wedge e \neq \natural) \oplus \Big((e = \natural \vee e' = \natural)\Big)$,

where $\oplus$ denoted the exclusive-or operator.

In a partially synchronous system, there are different ordering of events and each unique ordering of events might evaluate to different values. Given a distributed stream, $(\mathcal{E}, \rightsquigarrow)$, a sequence of consistent cuts is of the form $C_0C_1C_2\cdots C_N$, where for all $i \geq 0$: (1) $C_i \subseteq \mathcal{E}$, and (2) $C_i \subseteq C_{i+1}$.

Given the semantics of partially-synchronous Lola, evaluation of output stream variable s_i at time instance j requires events $\alpha_i(k)$, where $i \in [1, |\mathcal{A}|]$ and $k \in \Big\{\pi \mid \max\{0, j-\epsilon+1\} \leq \pi \leq \{N, j+\epsilon-1\}\Big\}$. To translate monitoring of a distributed stream to a synchronous stream, we make sure that the events in the frontier of a consistent cut, C_j are $\alpha_i(k)$.

Let $\mathbb{C}$ denote the set of all valid sequences of consistent cuts. We define the set of all synchronous streams of $(\mathcal{E}, \rightsquigarrow)$ as follows:

$$\mathsf{Sr}(\mathcal{E}, \rightsquigarrow) = \Big\{\mathsf{front}(C_0)\mathsf{front}(C_1)\cdots \mid C_0C_1\cdots \in \mathbb{C}\Big\}$$

Intuitively, $\mathsf{Sr}(\mathcal{E}, \rightsquigarrow)$ can be interpreted as the set of all possible "interleavings". The evaluation of the Lola specification, φ, with respect to $(\mathcal{E}, \rightsquigarrow)$ is the following :

$$\Big[(\mathcal{E}, \rightsquigarrow) \models_{PS} \varphi\Big] = \Big\{(\alpha_1, \cdots, \alpha_n) \models_S \varphi \mid (\alpha_1, \cdots, \alpha_n) \in \mathsf{Sr}(\mathcal{E}, \rightsquigarrow)\Big\}$$

This means that evaluating a partially synchronous distributed stream with respect to a Lola specification results in a set of evaluated results, as the computation may involve several streams. This also enables reducing the problem from evaluation of a partially synchronous distributed system to the evaluation of multiple synchronous streams, each evaluating to unique values for the output stream, with message complexity

$$O\big(\epsilon^{|\mathcal{A}|}N|\mathcal{M}|^2\big) \quad \Omega(N|\mathcal{M}|^2)$$

4.3 Problem Statement

The overall problem statement requires that upon the termination of the Algorithm 1, the verdict of all the monitors in the decentralized monitoring architecture is the same as that of a centralized monitor which has the global view of the system

$$\forall i \in [1, m] : \mathsf{Result}_i = \Big[(\mathcal{E}, \rightsquigarrow) \models_{PS} \varphi\Big]$$

where $(\mathcal{E}, \rightsquigarrow)$ is the global distributed stream and φ is the Lola specification with Result_i as the evaluated result by monitor M_i.

5 CALCULATING *LS*

In this section, we introduce the rules of rewriting Lola associated equations given the evaluated results and observations of the system. In our distributed setting, evaluation of a Lola specification involves generating a set of synchronous streams and evaluating the given Lola specification on it (explained in Section 6). Here, we make use of the evaluation of Lola specification into forming our local observation to be shared with other monitors in the system.

Given the set of synchronous streams, $(\alpha_1, \alpha_2, \cdots, \alpha_{|\mathcal{A}|})$, the symbolic locally computed result LS (see Algorithm 1) consists of associated Lola equations, which either needs more information (data was unobserved) from other monitors to evaluate or the concerned monitor needs to wait (positive offset). In either case, the associated Lola specification is shared with all other monitors in the system as the missing data can be observed by either monitors. We divide the rewriting rules into three cases, depending upon the observability of the value of the independent variables required for evaluating the expression e_i for all $i \in [1, n]$. Each stream expression is categorized into three cases (1) completely unobserved, (2) completely observed or (3) partially observed. This can be done easily by going over the dependency graph and checking with the partial distributed stream read by the corresponding monitor.

Case 1 (Completely Observed). Formally, a completely observed stream expression s_i can be identified from the dependency graph, $G = \langle V, E \rangle$, as for all s_k (resp. t_k) $\langle s_i, s_k, w \rangle \in E$ (resp. $\langle s_i, t_k, w \rangle \in E$), $s_k(j+w) \neq \natural$ (resp. $t_k(j+w) \neq \natural$) are observed for time instance j. If yes, this signifies, that all independent and dependent variables required to evaluate $s_i(j)$, is observed by the monitor M, there by evaluating: $s_i(j) = e_i(s_1, \cdots, s_n, t_1, \cdots, t_m)$ and rewriting $s_i(j)$ to LS.

Case 2 (Completely Unobserved). Formally, we present a completely unobserved stream expression, s_i from the dependency graph, $G = \langle V, E \rangle$, as for all s_k (resp. t_k), $\langle s_i, s_k, w \rangle \in E$ (resp. $\langle s_i, t_k, w \rangle \in E$), $s_k(j+w) = \natural$ (resp. $t_k(j+w) = \natural$) are unobserved, for time instance j . This signifies that the valuation of neither variables are known to the monitor M. Thus, we rewrite the following stream expressions

$$s'_k(j) = \begin{cases} s_k(j+w) & 0 \leq j+w \leq N \\ \texttt{default} & \text{otherwise} \end{cases}$$

$$t'_k(j) = \begin{cases} t_k(j+w) & 0 \leq j+w \leq N \\ \texttt{default} & \text{otherwise} \end{cases}$$

for all $\langle s_i, s_k, w \rangle \in E$ and $\langle s_i, t_k, w \rangle \in E$, and include the rewritten associated equation for evaluating $s_i(j)$ as

$$s_i(j) = e_i(s'_1, \cdots, s'_n, t'_1, \cdots, t'_m)$$

It is to be noted that the `default` value of a stream variable, s_k (resp. t_k), depends on the corresponding type T_k (resp. T_{m+k}) of the stream.

Case 3 (Partially Observed). Formally, we present a partially observed stream expression, s_i from the dependency graph, $G = \langle V, E \rangle$, as for all s_k (resp. t_k), they are either observed or unobserved, for time instance j. In other words, we can represent a set $\mathbb{V}_o = \{s_k \mid \exists s_k(j+w) \neq \natural\}$ of all observed stream variable and a set $\mathbb{V}_u = \{s_k \mid s_k(j+w) = \natural\}$ of all unobserved dependent stream variable for all $\langle s_i, s_k, w \rangle \in E$. The set can be expanded to include independent variables as well. For all $s_k \in \mathbb{V}_u$ (resp. $t_k \in \mathbb{V}_u$) that are unobserved, are replaced by:

$$s^u_k(j) = \begin{cases} s_k(j+w) & 0 \leq j+w \leq N \\ \texttt{default} & \text{otherwise} \end{cases}$$

$$t^u_k(j) = \begin{cases} t_k(j+w) & 0 \leq j+w \leq N \\ \texttt{default} & \text{otherwise} \end{cases}$$

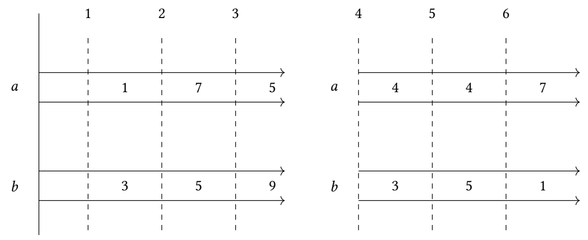


Fig. 4. Example of generating LS

and for all $s_k \in \mathbb{V}_o$ (resp. $t_k \in \mathbb{V}_o$) that are observed, are replaced by:

$$s_k^o(j+w) = \texttt{value}$$
$$t_k^o(j+w) = \texttt{value}$$

and there by partially evaluating $s_i(j)$ as

$$s_i(j) = e_i(s_1^o, \cdots, s_n^o, t_1^o, \cdots, t_m^o, s_1^u, \cdots, s_n^u, t_1^u, \cdots, t_m^u)$$

followed by adding the partially evaluated associated equation for $s_i(j)$ to LS. It is to be noted, that a consistent partial distributed stream makes sure that for all s_k (resp. t_k), can only be either observed or unobserved and not both or neither.

Example 5.1. Consider the Lola specification mentioned below and the stream input of length $N = 6$ divided into two evaluation rounds and $\epsilon = 2$ as shown in Fig. 4 with the monitors M_1 and M_2.

```
input a : uint
input b : uint
output c := ite(a[-1,0] <= b[1, 0], a[1,0], b[-1, 0])
```

The associated equation for the output stream is:

$$c = \begin{cases} \texttt{ite}(0 \leq b(i+1), a(i+1), 0) & i = 1 \\ \texttt{ite}(a(i-1) \leq b(i+1), a(i+1), b(i-1)) & 2 \leq i \leq N-1 \\ \texttt{ite}(a(i-1) \leq 0, 0, b(i-1)) & i = N \end{cases}$$

Let the partial distributed stream read by monitor M_1 include $\{a, (1,1), (3,5)\}, \{b, (2,5), (3,9)\}$ and the partial distributed stream read by monitor M_2 include $\{a, (1,1), (2,7)\}, \{b, (1,3), (3,9)$. Monitor M_1 evaluates $c(2) = 5$ and partially evaluates $c(1)$ and $c(3)$. Thus $LS_1^1 = \{c(1) = a(2), c(2) = 5, c(3) = \texttt{ite}(a(2) \leq b(4), a(4), 5)\}$. Monitor M_2 partially evaluates all $c(1)$, $c(2)$ and $c(3)$ and thus $LS_1^2 = \{c(1) = \texttt{ite}(0 \leq b(2), a(2), 0), c(2) = a(3), c(3) = \texttt{ite}(7 \leq b(4), a(4), b(2))\}$.

Let the partial distributed stream read by monitor M_1 include $\{a, (4,4), (5,4)\}, \{b, (4,3), (6,1)\}$ and the partial distributed stream read by monitor M_2 include $\{a, (5,4), (6,7)\}, \{b, (4,3), (5,5)\}$. Monitor M_1 evaluates $c(4) = 9$ and $c(5) = 3$ and partially evaluates $c(6)$. Thus $LS_2^1 = \{c(4) = 9, c(5) = 3, c(6) = b(5)\}$. Monitor M_2 evaluates $c(6) = 5$ and partially evalues $c(4)$ and $c(5)$ and thus $LS_2^2 = \{c(4) = \texttt{ite}(a(3) \leq 5, 4, 9), c(5) = \texttt{ite}(a(4) \leq b(6), 7, 3), c(6) = 5\}$.

It is to be noted, the after the first round of evaluation, the corresponding local states, LS_1^1 and LS_1^2 will be shared which will enable evaluating the output stream for few of the partially evaluated output stream (will be discussed in Section 7.1). These will be included in the local state of the following evaluation round.

Note that generating *LS* takes into consideration an ordered stream. One where the time of occurrence of events and values are comparable. It can be imagined that generating the same for the distributed system involves generating it for all possible ordering of events. This will be discussed in details in the following sections.s.

6 SMT-BASED SOLUTION

6.1 SMT Entities

SMT entities represent (1) Lola equations, and (2) variables used to represent the distributed stream. Once we have generated a sequence of consistent cuts, we use the laws discussed in Section 5, to construct the set of all locally computer or partially computed Lola equations.

Distributed Stream. In our SMT encoding, the set of events, $\mathcal{E}$, is represented by a bit vector, where each bit corresponds to an individual event in the distributed stream, $(\mathcal{E}, \rightsquigarrow)$. The length of the stream under observation is k, which makes $|\mathcal{E}| = k \times |\mathcal{A}|$ and the length of the entire stream is N. We conduct a pre-processing of the distributed stream where we create a $\mathcal{E} \times \mathcal{E}$ matrix, `hbSet` to incorporate the happen-before relations. We populate `hbSet` as `hbSet[e][f]` = 1 iff $e \rightsquigarrow f$, else `hbSet[e][f]` = 0. In order to map each event to its respective stream, we introduce a function, $\mu : \mathcal{E} \rightarrow \mathcal{A}$.

We introduce a valuation function, $val : \mathcal{E} \rightarrow \mathsf{T}$ (whatever the type is in the Lola specification), in order to represent the values of the individual events. Due to the partially synchronous assumption of the system, the possible time of occurrence of an event is defined by a function $\delta : \mathcal{E} \rightarrow \mathbb{Z}_{\geq 0}$, where $\forall \alpha(\sigma) \in \mathcal{E}.\exists \sigma' \in [\max\{0, \sigma - \epsilon + 1\}, \min\{\sigma + \epsilon - 1\}, N].\delta\big(\alpha(\sigma)\big) = \sigma'$. We update the δ function when referring to events on output streams by updating the time synchronization constant to ϵ_M. This accounts for the clock skew between two monitors. Finally, we introduce an uninterpreted function $\rho : \mathbb{Z}_{\geq 0} \rightarrow 2^{\mathcal{E}}$ that identifies a sequence of consistent cuts for computing all possible evaluations of the Lola specification, while satisfying a number of given constrains explained in Section 6.2.

6.2 SMT Constrains

Once we have defined the necessary SMT entities, we move onto the SMT constraints. We first define the SMT constraints for generating a sequence of consistent cuts, followed by the ones for evaluating the given Lola equations φ_α.

Constrains for consistent cuts over ρ: In order to make sure that the uninterpreted function ρ identifies a sequence of consistent cuts, we enforce certain constraints. The first constraint enforces that each element in the range of ρ is in fact a consistent cut:

$$\forall i \in [0, k].\forall e, e' \in \mathcal{E}.\ \Big((e \rightsquigarrow e') \land (e' \in \rho(i))\Big) \rightarrow (e \in \rho(i))$$

Next, we enforce that each successive consistent cut consists of all events included in the previous consistent cut:

$$\forall i \in [0, k-1].\ \rho(i) \subseteq \rho(i+1)$$

Next, we make sure that the front of each consistent cut constitutes of events with possible time of occurrence in accordance with the semantics of partially-synchronous Lola:

$$\forall i \in [0, k].\forall e \in \mathsf{front}(\rho(i)).\ \delta(e) = i$$

Finally, we make sure that every consistent cut consists of events from all streams:

$$\forall i \in [0, k].\forall \alpha \in \mathcal{A}.\exists e \in \mathsf{front}(\rho(i)).\mu(e) = \alpha$$

Constrains for Lola specification: These constraints will evaluate the Lola specifications and will make sure that ρ will not only represent a valid sequence of consistent cuts but also make sure that the sequence of consistent cuts evaluate the Lola equations, given the stream expressions. As is evident that a distributed system can often evaluate to multiple values at each instance of time. Thus, we would need to check for both satisfaction and violation for logical expressions and evaluate all possible values for arithmetic expressions. Note that monitoring all Lola specification can be reduce to evaluating expressions that are either logical or arithmetic. Below, we mention the SMT constraint for evaluating different Lola equations at time instance j:

$$t_i[p,c] = \begin{cases} val(e) & 0 \le j+p \le N \\ c & \text{otherwise} \end{cases} \qquad \Big(\exists e \in \mathsf{front}(\rho(j+p)).(\mu(e) = \alpha_i)\Big)$$

$$s_i(j) = \texttt{true}\ \mathsf{front}(\rho(j)) \models \varphi_\alpha \qquad \text{(Logical expression, satisfaction)}$$

$$s_i(j) = e_i(\forall e \in \mathsf{front}(\rho(j)).val(e)) \qquad \text{(Arithmetic expression, evaluation)}$$

The previously evaluated result is included in the SMT instance as a entity and a additional constrain is added that only evaluates to unique value, in order to generate all possible evaluations. The SMT instance returns a satisfiable result iff there exists at-least one unique evaluation of the equation. This is repeated multiple times until we are unable to generate a sequence of consistent cut, given the constraints, i.e., generate unique values. It is to be noted that stream expression of the form $\texttt{ite}(s_i, s_k, s_j)$ can be reduced to a set of expressions where we first evaluate s_i as a logical expression followed by evaluating s_j and s_k accordingly.

7 RUNTIME VERIFICATION OF LOLA SPECIFICATIONS

Now that both the rules of generating rewritten Lola equations (Section 5) and the working of the SMT encoding (Section 6) have been discussed, we can finally bring them together in order to solve the problem introduced in Section 4.

7.1 Computing *LC*

Given a set of local states computed from the SMT encoding, each monitor process receives a set of rewritten Lola associated equations, denoted by LS^i_j, where $i \in [1, |\mathcal{M}|]$ for j-th computation round. Our idea to compute LC from these sets is to simply take a prioritized union of all the associated equations.

$$LC(\Pi^i_j) = \biguplus_{i \in [1, |\mathcal{M}|]} LS^i_j$$

The intuition behind the priority is that an evaluated Lola equation will take precedence over a partially evaluated/unevaluated Lola equation, and two partially-evaluated Lola equation will be combined to form a evaluated or partially evaluated Lola equation. For example, taking the locally computed LS^1_1 and LS^2_1 from Example 5.1, $LC(LS^1_1, LS^2_1)$ is computed to be $\{c(1) = a(2), c(2) = 5, c(3) = \texttt{ite}(7 \le b(4), a(4), 5)\}$ at Monitor M_1 and $\{c(1) = 7, c(2) = 5, c(3) = \texttt{ite}(7 \le b(4), a(4), 5)\}$ at Monitor M_2. Subsequently, $LC(LS^1_2, LS^2_2)$ is computed to be $\{c(4) = 9, c(5) = 3, c(6) = 5\}$ at Monitor M_1 and $\{c(4) = 9, c(5) = 3, c(6) = 5\}$ at Monitor M_2.

7.2 Bringing it all Together

As stated in Section 4.1, the monitors are decentralized and online. Since, setting up of a SMT instance is costly (as seen in our evaluated results in Section 8), we often find it more efficient to evaluate the Lola specification after every k time instance. This reduces the number of computation

Algorithm 2 Computation on Monitor M_i

```
1: $LS_1^i[0] = \emptyset$
2: for $r = 1$ to $\lceil N/k \rceil$ do
3:   $(\mathcal{E}_i, \rightsquigarrow_i)_r \leftarrow r$-th Consistent partial distributed stream
4:   $j = 0$
5:   do
6:     $j = j + 1$
7:     $(\alpha_1, \alpha_2, \cdots, \alpha_{|\mathcal{A}|}) \in \mathsf{Sr}(\mathcal{E}_i, \rightsquigarrow_i)$
8:     $LS_r^i[j] \leftarrow LS_r^i[j-1] \cup \left[(\alpha_1, \alpha_2, \cdots, \alpha_{|\mathcal{A}|}) \models_S \varphi_\alpha\right]$
9:   while $(LS_r^i[j] \neq LS_r^i[j-1])$
10:  Send: broadcasts symbolic view $LS_r^i[j]$
11:  Receive: $\Pi_r^i \leftarrow \{LS_r^k \mid 1 \leq k \leq \mathcal{M}\}$
12:  Compute: $LS_{r+1}^i[0] \leftarrow LC(\Pi_r^i)$        ▷ Section 7.1
13: end for
14: Result$^i \leftarrow \bigcup_{r \in [1, \lceil N/k \rceil + 1]} LS_r^i[0]$
```

rounds to $\lceil N/k \rceil$ as well as the number of messages being transmitted over the network as well with an increase to the size of the messages. We update Algorithm 1 to reflect our solution more closely to Algorithm 2.

Each evaluation round starts by reading the r-th partial distributed system which consists of events occurring between the time $\max\{0, (r-1) \times \lceil N/k \rceil\}$ and $\min\{N, r \times \lceil N/k \rceil\}$ (line 3). We assume that the partial distributed system is consistent in accordance with the assumption that each event has been read by atleast one monitor. To account for any concurrency among the events in $(r-1)$-th computation round with that in the r-th computation round, we expand the length by ϵ time, there-by making the length of the r-th computation round, $\max\{0, (r-1) \times \lceil N/k \rceil - \epsilon + 1\}$ and $\min\{N, r \times \lceil N/k \rceil\}$.

Next, we reduce the evaluation of the distributed stream problem into an SMT problem (line 7). We represent the distributed system using SMT entities and then by the help of SMT constraints, and we evaluate the Lola specification on the generated sequence of consistent cuts. Each sequence of consistent cut presents a unique ordering of the events which evaluates to a unique value for the stream expression (line 8). This is repeated until we no longer can generate a sequence of consistent cut that evaluates φ_α to unique values (line 9). Both the evaluated as well as partially evaluated results are included in LS as associated Lola equations. This is followed by the communication phase where each monitor shares its locally computed LS_r^i, for all $i \in [1, |\mathcal{M}|]$ and r evaluation round (line 10-11).

Once, the local states of all the monitors are received, we take a prioritized union of all the associated equation and include them into LS_{r+1}^i set of associated equations (line 12). Following this, the computation shifts to next computation round and the above mentioned steps repeat again. Once we reach the end of the computation, all the evaluated values are contained in Result^i

Lemma 7.1. *Let $\mathcal{A} = \{S_1, S_2, \cdots, S_n\}$ be a distributed system and φ be an* Lola *specification. Algorithm 1 terminates when monitoring a terminating distributed system.*

Theorem 7.2. *Algorithm 2 solves the problem stated in Section 4.*

Theorem 7.3. *Let φ be a* Lola *specification and $(\mathcal{E}, \rightsquigarrow)$ be a distributed stream consisting of $|\mathcal{A}|$ streams. The message complexity of Algorithm 2 with $|\mathcal{M}|$ monitors is*

$$O\left(\epsilon^{|\mathcal{A}|} N |\mathcal{M}|^2\right) \quad \Omega(N|\mathcal{M}|^2)$$

8 CASE STUDY AND EVALUATION

In this section, we analyze our SMT-based decentralized monitoring solution. We note that we are not concerned about data collections, data transfer, etc, as given a distributed setting, the runtime of the actual SMT encoding will be the most dominating aspect of the monitoring process. We evaluate our proposed solution using traces collected from synthetic experiments (Section 8.1) and case studies involving several industrial control systems and RACE dataset (Section 8.2). The implementation of our approach can be found on Google Drive(https://tinyurl.com/2p6ddjnr).

8.1 Synthetic Experiments

8.1.1 Setup. Each experiment consists of two stages: (1) generation of the distributed stream and (2) verification. For data generation, we develop a synthetic program that randomly generates a distributed stream (i.e., the state of the local computation for a set of streams). We assume that streams are of the type `Float`, `Integer` or `Boolean`. For the streams of the type `Float` and `Integer`, the initial value is a random value `s[0]` and we generate the subsequent values by `s[i-1] + N(0, 2)`, for all $i \geq 1$. We also make sure that the value of a stream is always non-negative. On the other hand, for streams of the type `Boolean`, we start with either `true` or `false` and then for the subsequent values, we stay at the same value or alter using a Bernoulli distribution of $B(0.8)$, where a `true` signifies the same value and a `false` denotes a change in value.

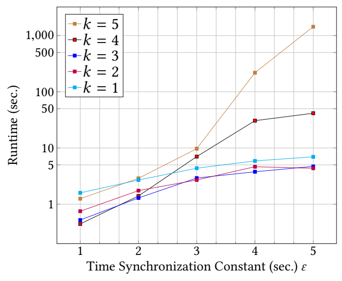
For the monitor, we study the approach using Bernoulli distribution $B(0.2)$, $B(0.5)$ and $B(0.8)$ as the read distribution of the events. A higher readability offers each event to be read by higher number of monitors. We also make sure that each event is read by at least one monitor in accordance with the proposed approach. To test the approach with respect to different types of stream expression, we use the following arithmetic and logical expressions.

```
input a1 : uint
input a2 : uint
output arithExp := a1 + a2
output logicExp := (a1 > 2) && (a2 < 8)
```

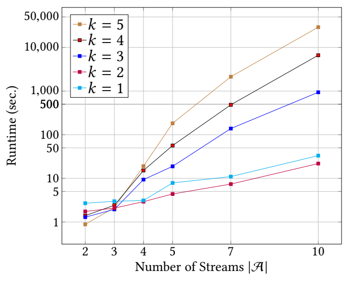
8.1.2 Result - Analysis. We study different parameters and analyze how it effects the runtime and the message size in our approach. All experiments were conducted on a 2017 MacBook Pro with 3.5GHz Dual-Core Intel core i7 processor and 16GB, 2133 MHz LPDDR3 RAM. Unless specified otherwise all experiments consider number of streams, $|\mathcal{A}| = 3$, time synchronization constant, $\epsilon_M = \epsilon = 3s$, number of monitors same as the number of streams, computation length, $N = 100$, with $k = 3$ with a read distribution $B(0.8)$.

Time Synchronization Constant. Increasing the value of the time synchronization constant ϵ, increases the possible number of concurrent events that needs to be considered. This increases the complexity of evaluating the Lola specification and there-by increasing the runtime of the algorithm. In addition to this, higher number of ϵ corresponds to higher number of possible streams that needs to be considered. We observe that the runtime increases exponentially with increasing the value of ϵ in Fig. 5a, as expected. An interesting observation is that with increasing the value of k, the runtime increases at a higher rate until it reaches the threshold where $k = \epsilon$. This is due to the fact, that the number of streams to be considered increases exponentially but ultimately gets bounded by the number of events present in the computation.

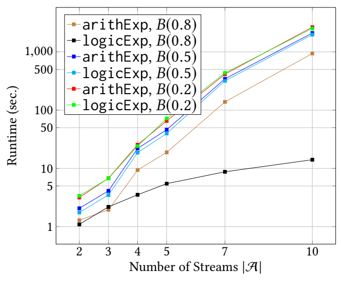
Increasing the value of the time synchronization constant is also directly proportional to the number of evaluated results at each instance of time. This is because, each stream corresponds to a unique value being evaluated until it gets bounded by the total number of possible evaluations, as



(a) Epsilon

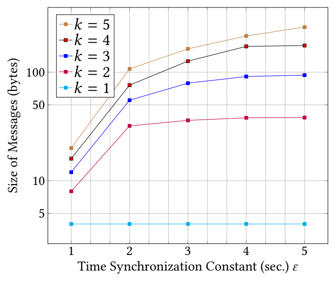


(b) Number of Streams

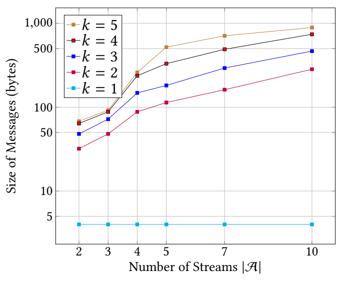


(c) Different LoLa Specification

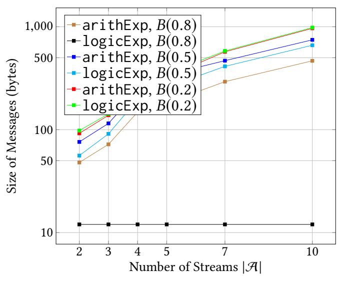
Fig. 5. Impact of different parameters on runtime for synthetic data.



(a) Epsilon



(b) Number of Streams



(c) Different LoLa Specification

Fig. 6. Impact of different parameters on message size for synthetic data.

can be seen in Fig. 6a. However, comparing Figs. 5a and 6a, we see that the runtime increases at a faster rate to the size of the message. This owes to the fact that initially a SMT instance evaluates unique values at all instance of time. However, as we start reaching all possible evaluations for certain instance of time, only a fraction of the total time instance evaluates to unique values. This is the reason behind the size of the message reaching its threshold faster than the runtime of the monitor.

Type of Stream Expression. Stream expressions can be divided into two major types, one consisting of arithmetic operations and the other involving logical operations. Arithmetic operations can evaluate to values in the order of $O(|\mathcal{A}|.\epsilon)$, where as logical operations can only evaluate to either `true` or `false`. When the monitors have high readability of the distributed stream, it is mostly the case, that the monitor was able to evaluate the stream expression. Thus, we observe in Fig. 5c that the runtime grows exponentially for evaluating arithmetic expressions but is linear for logical expressions. However, with low readability of the computation, irrespective of the type of expression, both takes exponential time since neither can completely evaluate the stream expression. So, each monitor has to generate all possible streams.

Similarly, for high readability and logical expressions, the message size is constant given the monitor was was able to evaluate the stream expression. However with low readability, message size for evaluating logical expressions matches with that of its arithmetic counterpart. This can be seen in Fig. 6c and is due to the fact, that with low readability, complete evaluation of the expression is not possible at a monitor and thus needs to send the rewritten expression with the values observed to the other monitors where it will be evaluated.

Number of Streams. As the number of streams increases, the number of events increase linearly and thereby making exponential increase in the number of possible synchronous streams (due to interleavings). This can be seen in Fig. 5b, where the runtime increases exponentially with increase in the number of streams in the distributed stream. Similarly, in Fig. 6b, increase in the number of streams linearly effects the number of unique values that the Lola expression can evaluate to and there-by increasing the size of the message.

8.2 Case Studies: Decentralized ICS and Flight Control RV

Industrial Control Systems (ICS) [28] are information systems to control industrial processes such as manufacturing, product handling, distribution, etc. It includes supervisory control and data acquisition systems used to control geographically dispersed assets (often in the range of few thousands of square kilometers) and distributed control systems using a programmable logic controller for each of the localized processes. ICS is a general term, usually used to refer to Supervisory Control and Data Acquisition (SCADA) systems, Distributed Control Systems (DCS) and Programmable Logic Controller (PLC) that receives data produced by a large number of sensors, fitted across the system. Moreover, typical ICS uses wide-area networks and wireless/radio frequency technologies for communication. This makes the communication prone to delays and data loss and there-by justifying our assumption of a partially-synchronous system.

The data produced by these components are often the target of cyber and ransom-ware attack putting the security of the system in jeopardy. Since these systems are linked to essential services, any attack on these facilities put the users life on the front line. The integrity of the data produced from these distributed components is very important as the PLC's behavior is dictated by it. Recent attacks have shown that an attack on a company's ICS costs the company around $5 million and 50 days of system down time. Additionally, according to a recent report [27], it takes the effected company around 191 days to fully recover and around 54% of all organization are vulnerable to such attacks.

Setup. We put our runtime verification approach to the test with respect to several industrial control system datasets that includes data generated by a (1) Secure Water Treatment plant (SWaT) [15] , comprising of six processes, corresponding to different physical and control components; (2) a Power Distribution system [29] that includes readings from four phaser measurement unit (PMU) that measures the electric waves on an electric grid, and (3) a Gas Distribution system [3] that includes messages to and from the PLC. In these ICS, we monitor for correctness of system properties. Additionally we monitor for mutual separation between all pairs of aircraft in RACE [20] dataset, that consists of SBS messages from aircrafts. For more details about each of the systems along with the Lola specifications refer to the supplement document Section A.3.

For our setting we assume, each component has its own asynchronous local clock, with varying time synchronization constant. Although the event rate in some manufacturing industrial processes can range from few milliseconds to as low as few micro-seconds, due to the public availability of the data, all our experiments consider a data rate of 1 second. Next we discuss the results of verifying different ICS with respect to Lola specifications.

Result Analysis. We employed same number of monitors as the number of components for each of the ICS case-studies and divided the entire airspace into 9 different ones with one monitor responsible for each. We observe that our approach does not report satisfaction of system property when there has been an attack on the system in reality (false-negative). However, due to the assumption of partial-synchrony among the components, our approach may report false positives, i.e., it reports a violation of the system property even when there was no attack on the system. As can be seen in Fig. 7, with decreasing time synchronization constant, the number of false-positives

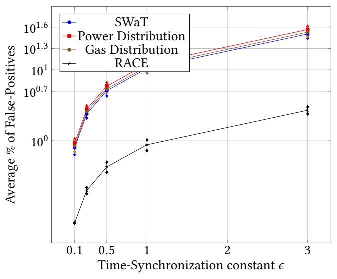


Fig. 7. False-Positives for ICS Case-Studies

reduce as well. This is due to the fact that with decreasing ϵ, less events are considered to be concurrent by the monitors. This makes the partial-ordering of events as observed by the monitor closer to the actual-ordering of events taking place in the system.

We get significantly better result for aircraft monitoring with fewer false-positives compared to the other dataset. This can be attributed towards Air Traffic Controllers maintaining greater separation between two aircrafts than the minimum that is recommended. As part of our monitoring of other ICS, we would like to report that our monitoring approach could successfully detect several attacks which includes underflow and overflow of tank and sudden change in quality of water in SWaT, differentiate between manual tripping of the breaker from the breaker being tripped due to a short-circuit in Power Distribution and Single-point data injection in Gas distribution.

9 RELATED WORK

Online predicate detection for both centralized and decentralized monitoring setting have been extensively studies in [7, 22]. Extensions to more expressive temporal operators are introduced in [23, 24]. Monitoring approaches introduced in [7, 23, 24] considers a fully asynchronous distributed system. An SMT-based predicate detection solution has been introduced in [32]. Runtime Verification for *synchronous* distributed system has been studied in [4, 8, 11]. The assumption of a common global clock shared among all the components act as a major shortcoming of this approach. In [11], the authors propose a decentralized runtime verification technique for synchronous Lola and study the technique for monitors arranged according to multiple topological scenarios. We not only extend the synchronous technique to a non-trivial partially-synchronous semantics, but also show that our approach can reach a lower bound complexity that makes partial evaluation followed by communication an efficient solution. Finally, fault-tolerant monitoring, where monitors can crash, has been investigated in [5] for asynchronous and in [17] for synchronized distributed processes.

Runtime Verification of stream-based specification was introduced in [9, 10], where the occurrence of the events was assumed to be synchronous. To extend the stream-based runtime verification to more complex systems, one where the occurrence of events is asynchronous, a real-time based logic was introduced in [18, 19, 31]. However, these methods fall short to verify large geographically separated distributed system, due to their assumption regarding the presence of a shared global clock. On the contrary, we assume the presence of a clock synchronization algorithm which limits the maximum clock skew among components to a constant. This is a realistic assumption since different components of a large industrial system have their own clock and it is certain to have a skew between them. A similar SMT-based solution was studied for LTL and MTL specifications in [13, 14] respectively, which we extend to include a more expressive stream-based specification.

10 CONCLUSION

In this paper, we studied distributed runtime verification w.r.t. to the popular stream-based specification language Lola. We propose a online decentralized monitoring approach where each monitor takes a set of associated Lola specification and a partial distributed stream as input. By assuming partial synchrony among all streams and by reducing the verification problem into an SMT problem, we were able to reduce the complexity of our approach where it is no longer dependent on the time synchronization constant. We also conducted extensive synthetic experiments, verified system properties of large Industrial Control Systems and airspace monitoring of SBS messages. Comparing to machine learning-based approaches to verify the correctness of these system, our approach was able to produce sound and correct results with deterministic guarantees. As a better practice, one can also use our RV approach along with machine-learning based during training or as a safety net when detecting system violations.

For future work, we plan to study monitoring of distributed systems where monitors themselves are vulnerable to faults such as crash and Byzantine faults. This will let us design a technique with faults and vulnerabilities mimicking a real life monitoring system and thereby expanding the reach and application of runtime verification on more real-life safety critical systems.

REFERENCES

[1] [n.d.].

[2] Tejasvi Alladi, Vinay Chamola, and Sherali Zeadally. 2020. Industrial Control Systems: Cyberattack trends and countermeasures. *Computer Communications* 155 (2020), 1–8. https://doi.org/10.1016/j.comcom.2020.03.007

[3] Justin M. Beaver, Raymond C. Borges-Hink, and Mark A. Buckner. 2013. An Evaluation of Machine Learning Methods to Detect Malicious SCADA Communications. In *2013 12th International Conference on Machine Learning and Applications*, Vol. 2. 54–59. https://doi.org/10.1109/ICMLA.2013.105

[4] B. Bonakdarpour and B. Finkbeiner. 2016. Runtime Verification for HyperLTL. In *Proceedings of the 16th International Conference on Runtime Verification*. 41–45.

[5] B. Bonakdarpour, P. Fraigniaud, S. Rajsbaum, D. A. Rosenblueth, and C. Travers. 2016. Decentralized Asynchronous Crash-Resilient Runtime Verification. In *Proceedings of the 27th International Conference on Concurrency Theory (CONCUR)*. 16:1–16:15.

[6] Raymond C. Borges Hink, Justin M. Beaver, Mark A. Buckner, Tommy Morris, Uttam Adhikari, and Shengyi Pan. 2014. Machine learning for power system disturbance and cyber-attack discrimination. In *2014 7th International Symposium on Resilient Control Systems (ISRCS)*. 1–8. https://doi.org/10.1109/ISRCS.2014.6900095

[7] H. Chauhan, V. K. Garg, A. Natarajan, and N. Mittal. 2013. A Distributed Abstraction Algorithm for Online Predicate Detection. In *Proceedings of the 32nd IEEE Symposium on Reliable Distributed Systems (SRDS)*. 101–110.

[8] C. Colombo and Y. Falcone. 2016. Organising LTL monitors over distributed systems with a global clock. *Formal Methods in System Design* 49, 1-2 (2016), 109–158.

[9] Lukas Convent, Sebastian Hungerecker, Martin Leucker, Torben Scheffel, Malte Schmitz, and Daniel Thoma. 2018. TeSSLa: Temporal Stream-Based Specification Language. In *Formal Methods: Foundations and Applications*, Tiago Massoni and Mohammad Reza Mousavi (Eds.). Springer International Publishing, Cham, 144–162.

[10] B. D'Angelo, S. Sankaranarayanan, C. Sanchez, W. Robinson, B. Finkbeiner, H.B. Sipma, S. Mehrotra, and Z. Manna. 2005. LOLA: runtime monitoring of synchronous systems. In *12th International Symposium on Temporal Representation and Reasoning (TIME'05)*. 166–174. https://doi.org/10.1109/TIME.2005.26

[11] L. M. Danielsson and C. Sánchez. 2019. Decentralized Stream Runtime Verification. In *Proceedings of the 19th International Conference on Runtime Verification (RV)*. 185–201.

[12] Honeywell Distributed Control Systems (DCS). 2023. https://process.honeywell.com/us/en/products/control-and-supervisory-systems/distributed-control-systems-dcs

[13] Ritam Ganguly, Anik Momtaz, and Borzoo Bonakdarpour. 2021. Distributed Runtime Verification Under Partial Synchrony. In *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*, Vol. 184. 20:1–20:17. https://doi.org/10.4230/LIPIcs.OPODIS.2020.20

[14] R. Ganguly, Y. Xue, A. Jonckheere, P. Ljung, B. Schornstein, B. Bonakdarpour, and M. Herlihy. 2022. Distributed Runtime Verification of Metric Temporal Properties for Cross-Chain Protocols. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, Los Alamitos, CA, USA, 23–33. https://doi.org/10.1109/ICDCS54860.2022.00012

[15] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. 2017. A Dataset to Support Research in the Design of Secure Water Treatment Systems. In *Critical Information Infrastructures Security*, Grigore Havarneanu, Roberto Setola, Hypatia Nassopoulos, and Stephen Wolthusen (Eds.). Springer International Publishing, Cham, 88–99.

[16] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (jul 1978), 558–565. https://doi.org/10.1145/359545.359563

[17] L. Lamport and N. Lynch. 1990. *Handbook of Theoretical Computer Science*. Vol. B. Elsevier Science Publishers B. V., Amsterdam, Chapter 18: Distributed Computing: Models and Methods.

[18] Martin Leucker, César Sánchez, Torben Scheffel, Malte Schmitz, and Daniel Thoma. 2019. Runtime Verification for Timed Event Streams with Partial Information. In *Runtime Verification: 19th International Conference, RV 2019, Porto, Portugal, October 8–11, 2019, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 273–291. https://doi.org/10.1007/978-3-030-32079-9_16

[19] Martin Leucker, César Sánchez, Torben Scheffel, Malte Schmitz, and Alexander Schramm. 2020. Runtime verification of real-time event streams under non-synchronized arrival. *Software Quality Journal* 28, 2 (2020), 745–787. https://doi.org/10.1007/s11219-019-09493-y

[20] Peter Mehlitz, Dimitra Giannakopoulou, and Nastaran Shafiei. 2019. Analyzing Airspace Data with RACE. In *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. 1–10. https://doi.org/10.1109/DASC43569.2019.9081664

[21] D. Mills. 2010. *Network Time Protocol Version 4: Protocol and Algorithms Specification.* RFC 5905. RFC Editor.

[22] N. Mittal and V. K. Garg. 2005. Techniques and applications of computation slicing. *Distributed Computing* 17, 3 (2005), 251–277.

[23] M. Mostafa and B. Bonakdarpour. 2015. Decentralized Runtime Verification of LTL Specifications in Distributed Systems. In *Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 494–503.

[24] V. A. Ogale and V. K. Garg. 2007. Detecting Temporal Logic Predicates on Distributed Computations. In *Proceedings of the 21st International Symposium on Distributed Computing (DISC)*. 420–434.

[25] Shengyi Pan, Thomas Morris, and Uttam Adhikari. 2015. Classification of Disturbances and Cyber-Attacks in Power Systems Using Heterogeneous Time-Synchronized Data. *IEEE Transactions on Industrial Informatics* 11, 3 (2015), 650–662. https://doi.org/10.1109/TII.2015.2420951

[26] Shengyi Pan, Thomas Morris, and Uttam Adhikari. 2015. Developing a Hybrid Intrusion Detection System Using Data Mining for Power Systems. *IEEE Transactions on Smart Grid* 6, 6 (2015), 3104–3113. https://doi.org/10.1109/TSG.2015.2409775

[27] Wolfgang Schwab and Mathieu Poujol. 2018. The state of industrial cybersecurity 2018. *Trend Study Kaspersky Reports* 33 (2018).

[28] Keith A. Stouffer, Joseph A. Falco, and Karen A. Scarfone. 2011. *SP 800-82. Guide to Industrial Control Systems (ICS) Security: Supervisory Control and Data Acquisition (SCADA) Systems, Distributed Control Systems (DCS), and Other Control System Configurations Such as Programmable Logic Controllers (PLC).* Technical Report. Gaithersburg, MD, USA.

[29] Chih-Che Sun, Chen-Ching Liu, and Jing Xie. 2016. Cyber-Physical System Security of a Power Grid: State-of-the-Art. *Electronics* 5, 3 (2016). https://doi.org/10.3390/electronics5030040

[30] SIMATIC PCS 7 – the distributed control system with proven performance. 2023. https://www.siemens.com/global/en/products/automation/process-control/simatic-pcs-7.html

[31] Hazem Torfah. 2019. Stream-Based Monitors for Real-Time Properties. In *Runtime Verification*, Bernd Finkbeiner and Leonardo Mariani (Eds.). Springer International Publishing, Cham, 91–110.

[32] V. T. Valapil, S. Yingchareonthawornchai, S. S. Kulkarni, E. Torng, and M. Demirbas. 2017. Monitoring Partially Synchronous Distributed Systems Using SMT Solvers. In *Proceedings of the 17th International Conference on Runtime Verification (RV)*. 277–293.

A SUPPLEMENT DOCUMENT

A.1 Lola Syntax

A stream expression is constructed as follows:

- If c is a constant of type T, then c is an atomic stream expression of type T
- If s is a stream variable of type T, then s is an atomic stream expression of type T.
- If $f : \mathsf{T}_1 \times \mathsf{T}_2 \times \cdots \mathsf{T}_k \rightarrow \mathsf{T}$ is a k-ary operator and for $1 \leq i \leq k$, e_i is an expression of type T_i, then $f(e_1, e_2, \cdots, e_k)$ is a stream expression of type T
- If b is a stream expression of type `boolean` and e_1, e_2 are stream expressions of type T, then $\mathtt{ite}(b, e_1, e_2)$ is a stream expression of type T, where `ite` is the abbreviated form of *if-then-else*.
- If e is a stream expression of type T, c is a constant of type T and i is an integer, then $e[i, c]$ is a stream expression of type T. $e[i, c]$ refers to the value of the expression e offset by i positions from the current position. In case the offset takes it beyond the end or before the beginning of the stream, then the *default* value is c.

Furthermore, Lola can be used to compute *incremental statistics*, where a given a stream, α, a function, $f_\alpha(v, u)$, computes a measure, where u represents the measure thus far and v, the current value. Given a sequence of values, $v_1, v_2, \cdots, v_n$, with a default value d, the measure over the data is given as

$$u = f_\alpha(v_n, f_\alpha(v_{n-1}, \cdots, f_\alpha(v_1, d)))$$

Example of such functions include *count*, $f_{count}(v, u) = u + 1$, *sum*, $f_{sum}(v, u) = u + v$, *max*, $f_{max}(v, u) = \max\{v, u\}$, among others. Aggregate functions like *average*, can be defined using two incremental functions, *count* and *sum*.

A.2 Proofs

Lemma A.1. *Let $\mathcal{A} = \{S_1, S_2, \cdots, S_n\}$ be a distributed system and φ be an Lola specification. Algorithm 1 terminates when monitoring a terminating distributed system.*

Proof. First, we note that our algorithm is designed for terminating system, also, note that a terminating program only produces a finite distributed computation. In order to prove the lemma, let us assume that the system send out a *stop* signal to all monitor processes when it terminates. When such a signal is received by a monitor, it starts evaluating the output stream expression using the terminal associated equations. This might arise to two cases. One where all the values required for the evaluation has been observed or one where the values required for the evaluation has not been observed. Although the termination of the monitor process for the first case is trivial, the termination of the monitor process for the second case is dependent upon replacing such unobserved stream value by the default value of the stream expression. Thus, terminating the monitor process eventually. □

Theorem A.2. *Algorithm 2 solves the problem stated in Section 4.*

Proof. We prove the soundness and correctness of Algorithm 2, by dividing it into three steps. In the first step we prove that given a Lola specification, φ, the values of the output stream when computed over the distributed computation, $(\mathcal{E}, \rightsquigarrow)$, of length N is the same as when the distributed computation is divided into $\frac{N}{k}$ computation rounds of length k each. Second, we prove that for all time instances the stream equation is eventually evaluated after the communication round. Finally we prove the set of all evaluated result is consistent over all monitors in the system.

Step 1: From our approach, we see that the value of a output stream variable, is evaluated on the events present in the consistent cut with time j. Therefore, we can reduce the proof to:

$$\mathsf{Sr}(\mathcal{E}, \rightsquigarrow) = \mathsf{Sr}(\mathcal{E}_1.\mathcal{E}_2 \cdots \mathcal{E}_{\frac{N}{k}}, \rightsquigarrow)$$

- ($\Rightarrow$) Let C_k be a consistent cut such that C_k is in $\mathsf{Sr}(\mathcal{E}, \rightsquigarrow)$, but not in $\mathsf{Sr}(\mathcal{E}_1.\mathcal{E}_2 \cdots \mathcal{E}_{\frac{N}{k}}, \rightsquigarrow)$, for some $k \in [0, |\mathcal{E}|]$. This implies that the frontier of C_k, $\mathsf{front}(C_k) \nsubseteq \mathcal{E}_1$ and $\mathsf{front}(C_k) \nsubseteq \mathcal{E}_2$ and $\cdots$ and $\mathsf{front}(C_k) \nsubseteq \mathcal{E}_{\frac{N}{k}}$. However, this is not possible, as according to the computation round construction in Section 7.2, there must be a $\mathcal{E}_i$, where $1 \leq i \leq \frac{N}{k}$ such that $\mathsf{front}(C_k) \subseteq \mathcal{E}_i$. Therefore, such C_k cannot exist, and $(\alpha_1, \alpha_2, \cdots, \alpha_n) \in \mathsf{Sr}(\mathcal{E}, \rightsquigarrow) \implies (\alpha_1, \alpha_2, \cdots, \alpha_n) \in \mathsf{Sr}(\mathcal{E}_1.\mathcal{E}_2 \cdots \mathcal{E}_{\frac{N}{k}}, \rightsquigarrow)$.
- ($\Leftarrow$) Let C_k be a consistent cut such that C_k is in $\mathsf{Sr}(\mathcal{E}_1.\mathcal{E}_2 \cdots \mathcal{E}_{\frac{N}{k}}, \rightsquigarrow)$ but not in $\mathsf{Sr}(\mathcal{E}, \rightsquigarrow)$ for some $k \in [0, |\mathcal{E}|]$. This implies, $\mathsf{front}(C_k) \subseteq \mathcal{E}_i$ and $\mathsf{front}(C_k) \nsubseteq \mathcal{E}$ for some $i \in [1, \frac{N}{k}]$. However, this is not possible due to the fact that $\forall i \in [1, \frac{N}{k}].\mathcal{E}_i \subset \mathcal{E}$. There, such C_k cannot exist, and $(\alpha_1, \alpha_2, \cdots, \alpha_n) \in \mathsf{Sr}(\mathcal{E}_1.\mathcal{E}_2 \cdots \mathcal{E}_{\frac{N}{k}}, \rightsquigarrow) \implies (\alpha_1, \alpha_2, \cdots, \alpha_n) \in \mathsf{Sr}(\mathcal{E}, \rightsquigarrow)$.

Therefore, $\mathsf{Sr}(\mathcal{E}, \rightsquigarrow) = \mathsf{Sr}(\mathcal{E}_1.\mathcal{E}_2 \cdots \mathcal{E}_{\frac{N}{k}}, \rightsquigarrow)$.

Step 2: Given a output stream expression s_i and the dependency graph $G = \langle V, E \rangle$, for each $\langle s_i, s_k, w \rangle \in E$, evaluating the value at time instance $j \in [1, N]$, $\alpha_k(j + w) \neq \natural$ or $\alpha_k(j + w) = \natural$ or $\alpha_k(w + j)$ not observed.

- If $\alpha_k(j + w) \neq \natural$, then we evaluate the stream expression
- If $\alpha_k(j + w) = \natural$, there exists at-least one other monitor where $\alpha_k(j + w) \neq \natural$. Thereby evaluating the stream expression, followed by sharing the the evaluated result with all other monitors
- If $\alpha_k(w+j)$ not observed, then at some future evaluation round and at some monitor $\alpha_k(j+w) \neq \natural$ and there-by evaluating the stream expression s_i

Similarly, it can be proved for $\langle s_i, t_k, w \rangle \in E$.

Step 3: Each monitor in our approach is fault-proof with communication taking place between all pairs of monitors. We also assume, all messages are eventually received by the monitors. This guarantees all observations are either directly or indirectly read by each monitor.

Together with Step 1 and 2, soundness and correctness of Algorithm 1 is proved. □

Theorem A.3. *Let φ be a Lola specification and $(\mathcal{E}, \rightsquigarrow)$ be a distributed stream consisting of $|\mathcal{A}|$ streams. The message complexity of Algorithm 2 with $|\mathcal{M}|$ monitors is*

$$O\big(\epsilon^{|\mathcal{A}|} N |\mathcal{M}|^2\big) \quad \Omega(N|\mathcal{M}|^2)$$

Proof. We analyze the complexity of each part of Algorithm 2. The algorithm has a nested loop. The outer loop iterates for $\lceil N/k \rceil$ times, that is $O(N)$. The inner loop is dependent on the number of unique evaluations of the stream expression.

- **Upper-bound** Due to our assumption of partial-synchrony, each event's time of occurrence can be off by ϵ. This makes the maximum number of unique evaluations in the order of $O(\epsilon^{|\mathcal{A}|})$.
- **Lower-bound** The minimum number of unique evaluations is in the order of $\Omega(1)$.

In the communication phase, each monitor sends $|\mathcal{M}|$ messages to all other monitors and receives $|\mathcal{M}|$ messages from all other monitors. That is $|\mathcal{M}|^2$. Hence the message complexity is

$$O\big(\epsilon^{|\mathcal{A}|} N |\mathcal{M}|^2\big) \quad \Omega(N|\mathcal{M}|^2)$$

As a side note, we would like to mention that in case of high readability of the monitors and evaluation of logical expression, the complexity is closer to the lower-bound, whereas with low readability and arithmetic expressions, the complexity is closer to the upper bound.

□

A.3 Industrial Control Systems

SWaT Dataset. Secure Water Treatment (SWaT) [15] utilizes a fully operational scaled down water treatment plant with a small footprint, producing 5 gallons/minute of doubly filtered water. It comprises of six main processes corresponding to the physical and control components of the water treatment facility. It starts from process P1 where it takes raw water and stores it in a tank. It is then passed through the pre-treatment process, P2, where the quality of the water is assessed and maintained through chemical dosing. The water then reaches P3 where undesirable materials are removed using fine filtration membranes. Any remaining chlorine is destroyed in the dechlorination process in P4 and the water is then pumped into the Reverse Osmosis system (P5) to reduce inorganic impurities. Finally in P6, water from the RO system is stored ready for distribution.

The dataset classifies different attack on the system into four types, based on the point and stage of the attack: Single Stage-Single Point, Single Stage-Multi Point, Multi Stage-Single Point and Multi Stage-Multi Point. We for the scope of this paper are the most interested in the attacks either covering multiple stages or multiple points. Few of the Lola specifications used are listed below.

```
input FIT-101 : uint
input MV-101 : bool
input LIT-101 : uint
input P-101 : bool
input FIT-201 : uint
output inflowCorr := ite(MV-101 == true, FIT-101 > 0, FIT-101 == 0)
output outflowCorr := ite(P-101 == true, FIT-201 > 0, FIT-201 == 0)
output tankCorr := ite(MV-101 == true || P-101 == true, LIT-101 = LIT-101[-1, 0] +
    FIT-101[-1, 0] - FIT-201[-1, 0])
```

where FIT-101 is the flow meter, measuring inflow into raw water tank, MV-101 is a motorized valve that controls water flow to the raw water tank, LIT-101 is the level transmitter of the raw water tank, P-101 is a pump that pumps water from raw water tank to the second stage and FIT-201 is the flow transmitter for the control dosing pumps. The above Lola specification checks the correctness of the inflow meter and valve pair (resp. outflow meter and pump pair) in inflowCorr (resp. outflowCorr) output expressions. On the other hand, tankCorr checks if the water level in the tank adds up to the in-flow and out-flow meters.

```
input AIT-201 : uint
input AIT-202 : uint
input AIT-203 : uint
output numObv := numObv[-1, 0] + 1
output NaClAvg := (NaClAvg[-1, 0] * numObv[-1, 0] + AIT-201) / numObv
output HClAvg := (HClAvg[-1, 0] * numObv[-1, 0] + AIT-202) / numObv
output NaOClAvg := (NaOClAvg[-1, 0] * numObv[-1, 0] + AIT-202) / numObv
```

where AIT-201, AIT-202 and AIT-203 represents the NaCl, HCl and NaOCl levels in water respectively and NaClAvg, HClAvg and NaOClAvg keeps a track of the average levels of the corresponding chemicals in the water, where as numObv keeps a track of the total number of observations read by the monitor.

Power System Attack Dataset. Power System Attack Dataset [29] consists of three datasets developed by Mississippi State University and Oak Ridge National Laboratory. It consists of readings from four phaser measurement unit (PMU) or synchrophasor that measures the electric waves on an electric grid. Each PMU measures 29 features consisting of voltage phase angle, voltage phase magnitude, current phase angle, current phase magnitude for Phase A-C, Pos., Neg. and Zero. It also measures the frequency for relays, the frequency delta for relay, status flag for relays, etc. Apart from these 116 PMU measurements, the dataset also consists of 12 control panel logs, snort alerts and relay logs of the 4 PMU.

The dataset classifies into either natural event/no event or an attack event. Few of the Lola specifications used are listed below. The first attempts to detect a single-line-to-ground (1LG) fault.

```
input R1-I : float
input R2-I : float
input R1-Relay : bool
input R2-Relay : bool
output R1-I-low := R1-I < 200
output R1-I-high := R1-I > 1000
output R2-I-low := R2-I < 200
output R2-I-high := R2-I > 1000
output 1LG := R1-I-high && R2-I-high && R1-Relay[+2, false] && R2-Relay[+2, false] &&
    R1-I-low[+4, false] && R2-I-low[+4, false]
```

where `R1-I` and `R2-I` represents the current measured at the R1 and R2 PMU respectively. Additionally, `R1-Relay` and `R2-Relay` keeps a track of the state of the corresponding relay. As a part of the 1LG attack detection, we first categorize the current measured as either low or high depending upon the amount of the current measured. We categorize an attack as 1LG if both R1 and R2 detects high current flowing followed by the relay tripping followed by low current.

```
input R1-PA1-I : float
input R1-PA2-I : float
input R1-PA3-I : float
output phaseBal := (R1-PA1-I - R1-PA2-I) <= 10 && (R1-PA2-I - R1-PA3-I) <= 10 &&
    (R1-PA3-I - R1-PA1-I) <= 10
```

where `R1-PA1-I`, `R1-PA2-I` and `R1-PA3-I` are the amount of current measured by R1 PMU at Phase A, B and C respectively. The monitor helps us to check if the load on three phases are equally balanced.

Gas Distribution System. Gas Distributed System [3] is a collection of labeled Remote Terminal Unit (RTU) telemetry streams from a Gas pipeline system in Mississippi State University's Critical Infrastructure Protection Center with collaboration from Oak Ridge National Laboratory. The telemetry streams includes messages to and from the Programmable Logic Controller (PLC) under normal operations and attacks involving command injection and data injection attack. The feature set includes the pipeline pressure, setpoint value, command data from the PLC, response to the PLC and the state of the solenoid, pump and the Remote Terminal Unit (RTU) auto-control.

One of the most common data injection attack is *Fast Change*. Here the reported pipeline pressure value is successively varied to create a lack of confidence in the correct operation of the system. The corresponding Lola specification monitoring against such attack is mentioned below:

```
input PipePress : float
input response : bool
output fastChange := ite(response, mod(PipePress - PipePress[-1, 1000]) <= 10, true)
```

where `PipePress` records the measured pipeline pressure and `response` is a flag variable signifying a message to the PLC. Here we consider the default pressure is 1000 psi and the permitted pressure change per unit time is 10 psi (these can be changed according to the demands of the system). Similarly we have Lola specifications monitoring other data injection attacks such as *Value Wave Injection*, *Setpoint Value Injection*, *Single Data Injection*, etc. and command injection attacks such as *Illegal Setpoint*, *Illegal PID Command*, etc.

RACE Dataset. Runtime for Airspace Concept Evaluation (RACE) [20] is a framework developed by NASA that is used to build an event based, reactive airspace simulation. We use a dataset developed using this RACE framework. This dataset contains three sets of data collected on three different days. Each set was recorded at around 37 N Latitude and 121 W Longitude. The dataset includes all 8 types of messages being sent by the SBS unit by using a Telnet application to listen to port 30003, but we only use the messages with ID 'MSG 3' which is the Airborne Position Message and includes a flight's latitude, longitude and altitude using which we verify the mutual separation of all pairs of aircraft. Furthermore, calculating the distance between two coordinates is computationally expensive, as we need to factor in parameters such as curvature of the earth. In order to speed up distance related calculations, we consider a constant latitude distance of 111.2km and longitude distance of 87.62km, at the cost of a negligible error margin. The corresponding Lola specification is mentioned below:

```
input flight1_alt : float
input flight1_lat : float
input flight1_lon : float
input flight2_alt : float
input flight2_lat : float
input flight2_lon : float
output distDiff := sqrt(pow(flight1_alt - flight2_alt, 2) + pow((flight1_lon -
    flight2_lon)*87620, 2) + pow((flight1_lat - flight2_lat)*111200, 2))
output check := distDiff > 500
```

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009