Runtime Verification of LTL on Lossy Traces

Yogi Joshi
University of Waterloo
y2joshi@uwaterloo.ca

Guy Martin Tchamgoue
University of Waterloo
gmtchamg@uwaterloo.ca

Sebastian Fischmeister
University of Waterloo
sfischme@uwaterloo.ca

ABSTRACT

Runtime verification techniques mostly assume the existence of complete execution traces. However, real-world systems often produce lossy traces due to network issues, partial instrumentation, sampling, and logging failures. A few verification techniques have recently emerged to handle systems with incomplete traces. Some of these techniques sacrifice soundness and may produce imprecise verdicts. The others depend on the recovery of lost events for a sound and meaningful verdict. In this paper, we present an offline algorithm that identifies whether an LTL (Linear Temporal Logic) formula can be *soundly* monitored in the presence of a *transient* loss of events in a trace and constructs a monitor accordingly. More, we introduce the concept of *monotonicity* to express the persistence of the verdicts of a loss-tolerant monitor regardless of the recovery of the lost events. Our evaluation demonstrates the applicability, efficiency and practicality of the technique on common LTL patterns, but also on traces from Google Clusters and MPlayer.

CCS Concepts

•Theory of computation → Linear logic; •Software and its engineering → Formal software verification;

Keywords

Runtime Verification; LTL; Transient Loss

1. INTRODUCTION

Runtime verification (RV) is the problem of, given a program P and an execution trace σ of P along with a specification φ, decide whether σ satisfies φ. A monitor $\mathcal{M}^{\varphi}$ is synthesized for φ. Thus, RV aims to find whether P exhibits the behavior described by φ. Most existing RV techniques assume the existence of complete traces [5, 11, 10]. However, real-world applications often produce lossy traces due to lossy network protocols [12], logging failures [3], sampling-based profilers like OPROFILE [16], and partial instrumentation tools like DIME [2]. Unfortunately, a sound monitor for complete traces may deliver an incorrect verdict on lossy traces. Hence, it is important to develop monitors that yield sound verdicts even on lossy traces.

A few approaches have recently emerged for the verification of lossy traces. Stoller *et al.* [22] proposed the concept of RV with state estimation, where the probability of whether a program satisfies or violates a specification φ is calculated using a Hidden Markov Model (HMM). However, this approach is limited by the requirement of a comprehensive set of traces used to learn the HMM. Thus, if no trace violates φ, an unsound verdict may be delivered. Basin *et al.* [3] showed that not all logging failures can affect the truth-values of formulas in a fragment of metric first-order temporal logic (MFOTL). Thus, in some cases, if the truth-value of a formula cannot be determined, three-valued semantics are used to express the uncertainty about the verdict. Further, if the lost events are recovered, such uncertainty may be resolved. For safety-critical and financial applications for instance, it is crucial that a monitor delivers sound verdicts irrespectively of the recovery of lost events.

In this paper, we present a novel approach to the problem of RV of LTL [17] formulas on lossy traces. We propose an offline algorithm that identifies whether an LTL formula, φ, can be *soundly* monitored in the presence of a *transient* loss of events in a trace σ and constructs a monitor accordingly. Since a transient loss is not permanent, it is required that a system that encounters such a loss *eventually* recovers to observe subsequent valid events. Evaluation of LTL formulas may be loss-sensitive. A finite loss of events results in a failure for a monitor to process the corresponding events. Consequently, such a monitor may deliver an unsound verdict for the monitored LTL formulas, which can be either violated or satisfied over a finite trace. For example, for the LTL formula, $\Diamond\, thread\_exit$ (a thread eventually exits), if a monitor does not observe the event when the thread calls *pthread_exit*, then it would deliver an incorrect verdict.

Some LTL formulas can never be satisfied or violated over finite traces. Bauer *et al.* [5] showed that such formulas count for about 45% of the commonly used LTL formulas. The formula $\varphi = \Box\,(job\_killed \rightarrow \Diamond\, job\_resubmitted)$ captures the idea that any job killed is eventually resubmitted. A monitor for φ can tolerate a finite loss of events. Intu-

SAC 2017, April 03-07, 2017, Marrakech, Morocco

http://dx.doi.org/10.1145/3019612.3019827

itively, even if finitely many events about the job are not observed, the monitor can still determine the verdict based upon the observation of subsequent events with at least one stating that the job is killed or resubmitted. Although such formulas can never be satisfied or violated by finite traces, Falcone *et al.* [9] showed that they are *monitorable*.

The proposed technique extends the monitor construction method of Bauer *et al.* [5] to synthesize loss-tolerant monitors. It is expected that the verdict of a monitor that processes a lossy trace aligns with that of a monitor processing the complete trace. More, this verdict should remain unchanged even with the recovery of the lost events. To capture this notion, we introduce the concept of *monotonicity*. We note that probabilistic monitoring approaches [19, 20, 22] do not exhibit this behavior as the verdict of monitors may change with the recovery of lost events. Finally, the contributions of this paper can be summarized as follows:

- *Monitorability*: We present an offline algorithm that identifies whether an LTL formula φ is monitorable in the presence of a transient loss and constructs a loss-tolerant monitor for φ guaranteeing monotonicity.
- *Monotonicity*: We introduce this concept to capture the fact that the verdict of a monitor on a lossy trace should match that of a monitor processing the complete trace, regardless of the recovery of the lost events.
- *Evaluation*: The evaluation of the technique on the set of common LTL patterns identified by Dwyer *et al.* [8] and on two real-world traces demonstrate the efficiency and effectiveness of the lost-tolerant monitors.

2. OVERVIEW OF LTL

A program P is considered a generator of *computation*, i.e., an infinite sequence of *events* or *states*. We use the symbol Σ to denote the set of states, also known as *alphabet*. Thus, an infinite sequence of elements from an alphabet Σ is called an *infinite word*. Similarly, a finite sequence of elements from Σ is a *finite word*. Further, we use Σ^* and Σ^+ to respectively designate the set of all finite words and that of all nonempty finite words. Also, we use Σ^ω to identify the set of all infinite words. A program P produces an execution trace σ, and a monitor $\mathcal{M}^\varphi$ provides a verdict stating whether σ *satisfies* an LTL specification φ. The concatenation of a finite trace σ with another trace ρ is denoted by $\sigma.\rho$.

DEFINITION 1 (SYNTAX OF LTL). *Let AP be a finite and non-empty set of* atomic propositions*, and $\Sigma = 2^{AP}$ a finite* alphabet*. The set of* LTL *specifications is inductively defined as: $\varphi ::= true \mid p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathbf{U} \varphi_2$, where $p \in AP$, and $\bigcirc$ (next), and $\mathbf{U}$ (until) are the temporal operators. The boolean operators retain their usual meaning.*

DEFINITION 2 (SEMANTICS OF LTL). *Let $\sigma = a_1, a_2, \ldots$ be an infinite word in Σ^ω.* LTL *semantics are inductively defined as per below:*

a) $\sigma, i \models \top$ *b)* $\sigma, i \models p$ iff $p \in a_i$
c) $\sigma, i \models \neg\varphi$ iff $\sigma, i \not\models \varphi$ *d)* $\sigma, i \models \bigcirc\varphi$ iff $\sigma, i+1 \models \varphi$
e) $\sigma, i \models \varphi_1 \vee \varphi_2$ iff $\sigma, i \models \varphi_1 \vee \sigma, i \models \varphi_2$
f) $\sigma, i \models \varphi_1 \mathbf{U} \varphi_2$ iff $\exists k \geq i : \sigma, k \models \varphi_2 \wedge \forall j : i \leq j < k : \sigma, j \models \varphi_1$

where $\models$ denotes the satisfaction relation.

Further, $\sigma \models \varphi$, also referred to as $[\sigma \models \varphi]_\omega$, is *true* iff $\sigma, 1 \models \varphi$. We introduce syntactic sugar in the form of two operators $\Box$ (*always*) and $\Diamond$ (*eventually*). Thus, $\Diamond\varphi$ is defined as $true\,\mathbf{U}\,\varphi$, and $\Box\varphi$ defined as $\neg\Diamond\neg\varphi$. An LTL formula φ defines a set of traces, which we refer to as $L(\varphi)$. A trace σ satisfies φ if $\sigma \in L(\varphi)$. Chang *et al.* [6] classified LTL formulas into six classes: safety, guarantee, obligation, response, persistence, and reactivity.

LTL semantics are defined over infinite paths. However, in practice, a program may only generate a finite word. Thus, multiple *finite path semantics* such as LTL$_3$ [5], FLTL [4, 14], RV-LTL [4] have been defined for LTL. In this paper, we focus primarily on RV-LTL semantics.

DEFINITION 3 (RV-LTL SEMANTICS). *Let $\sigma = a_1, a_2, \ldots \in \Sigma^*$ denote a finite trace. The truth value of a RV-LTL formula φ for the trace σ, denoted $[\sigma \models \varphi]_{RV}$, is an element of the truth domain $\mathbb{B}_4 = \{\top, \top_p, \bot_p, \bot\}$ defined as follows:*

$$[\sigma \models \varphi]_{RV} = \begin{cases} \top & \text{if } \forall\, v \in \Sigma^\omega : [\sigma.v \models \varphi]_\omega \\ \bot & \text{if } \forall\, v \in \Sigma^\omega : [\sigma.v \not\models \varphi]_\omega \\ \top_p & \text{if } [\sigma \models \varphi]_3 = ? \text{ and } [\sigma \models \varphi]_F = \top \\ \bot_p & \text{if } [\sigma \models \varphi]_3 = ? \text{ and } [\sigma \models \varphi]_F = \bot \end{cases}$$

$[\sigma \models \varphi]_3$, $[\sigma \models \varphi]_F$ and $[\sigma \models \varphi]_\omega$ respectively denote the satisfaction relations for LTL$_3$ [5], FLTL [4, 14], and LTL.

An alternate definition by Falcone *et al.* [9] states that an LTL formula φ is *monitorable* iff the constructed monitor can distinguish between *good* and *bad* finite prefixes, i.e.,

$$\forall\, \sigma_{good} \in L^*(\varphi) \cdot \forall\, \sigma_{bad} \in L^*(\neg\varphi) \cdot [\sigma_{good} \models \varphi]_{\mathbb{B}} \neq [\sigma_{bad} \models \varphi]_{\mathbb{B}}$$

$\mathbb{B}$ denotes the truth-domain of the finite path semantics. $L^*(\varphi)$ denotes the set of *good* finite prefixes for φ.

DEFINITION 4 (RV-LTL MONITOR). *Let φ be a RV-LTL formula over an alphabet Σ. The* monitor $\mathcal{M}^\varphi$ *for φ is the final state machine $(\Sigma, Q, q_0, \delta, \lambda)$, where Q is a set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to Q$ is the transition relation, and λ is a function that maps each state in Q to a value in $\{\top, \top_p, \bot_p, \bot\}$, such that, $[\sigma \models \varphi]_{RV} = \lambda(\delta(q_0, \sigma))$.*

3. FORMAL PROBLEM DESCRIPTION

Given an LTL formula φ and a lossy trace σ, our goal is to:

- Decide whether φ can be *soundly* monitored on σ.
- If yes, construct a loss-tolerant monitor $\mathcal{M}^\varphi$ for the formula φ such that $[\sigma \models \varphi]_{RV} = \lambda(\delta(q_0, \sigma))$, i.e., $\mathcal{M}^\varphi$ is *sound* w.r.t. RV-LTL semantics.

By soundness, we mean the truth-value output by $\mathcal{M}^\varphi$ should align with that of a monitor processing the complete trace. More, it is important that $\mathcal{M}^\varphi$ exhibits *monotonicity*, i.e., its verdict remains unchanged with the recovery of loss.

We assume only a transient loss of events, i.e., a finite loss of a sequence of events in a trace σ such that the loss is not permanent, and it is guaranteed that the monitor can observe subsequent valid events after the loss.

DEFINITION 5 (TRANSIENT LOSS). *Let $\sigma = a_1, a_2, \ldots, a_n \in \Sigma^*$ denotes a finite trace of size n. σ exhibits a transient*

loss of events iff there exists a finite, non-empty set of disjoint and bounded integer intervals, $I_s = \{[i_1, j_1], [i_2, j_2], \cdots, [i_m, j_m]\}$ *ordered s.t.* $(\forall k,\ i_k < i_{k+1} < n,\ j_k < j_{k+1} < n,\ i_k \leq j_k \wedge i_k > j_{k-1}) \wedge (\forall I \in I_s, \forall k \in I, a_k = \chi \wedge \forall l \notin I, a_l \neq \chi)$. χ *marks the loss.*

In this paper, a lossy observer O incrementally extracts events from P to feed a monitor $\mathcal{M}$, which verifies whether P satisfies a formula φ. The extracted word may thus contain gaps, but we assume that O transmits the trace to $\mathcal{M}$ without any extra loss. When no event is observed, O outputs a χ symbol to mark the loss. This extraction model is common with sampling-based profilers like OPROFILE [16], and partial instrumentation tools like DIME [2].

In our model, a *loss-tolerant* monitor $\mathcal{M}$ consumes an element of Σ to produce an output in the truth-domain $\mathbb{B}_5 = \{\top, \top_p, ?, \bot_p, \bot\}$, which is a $\mathbb{B}_4$ augmented with '?' and defined by the function $\lambda : Q \to \mathbb{B}_5$, where Q is the set of states of the monitor. Thus, $\mathcal{M}$ is defined similarly to a RV-LTL monitor, except that it yields '?' on the unknown input χ.

4. MONITORABILITY CRITERIA

This section presents our monitorability criteria and uses a few motivating examples to describe the concepts.

4.1 Formal Definitions

Let φ be an LTL formula, and $\mathcal{M} = (\Sigma, Q, q_0, \delta, \lambda)$ the RV-LTL monitor for φ. Let $\sigma \in \Sigma^*$ denote a finite trace, which exhibits transient loss. Let Q_s be a set of states of $\mathcal{M}$ such that $Q_s \subseteq Q$. Let Q_f be the set of final states of $\mathcal{M}$.

DEFINITION 6 (LOSS-TOLERANT ALPHABET). *A loss-tolerant alphabet,* Σ_-*, is a non-empty subset of* Σ*, such that an element* $\alpha \in \Sigma_-$ *iff* $\forall q_i \in Q,\ \delta(q_i, \alpha) = q_\alpha \in Q$.

Each element of a loss-tolerant alphabet forces the monitor to transition into a unique state irrespectively of its current state. This is particularly important when a monitored system is facing transient loss. Thus, at the end of the loss, by processing an element $\alpha \in \Sigma_-$, the monitor can safely move from its current state to its next state q_α.

DEFINITION 7 (LOSS-TOLERANT CLUSTER). *A loss-tolerant cluster,* Q_s*, is a non-empty subset of* Q*, s.t.* $\forall \alpha \in \Sigma_-, \forall (q_i, q_j) | q_i \in Q_s \wedge q_j \in Q_s,\ \delta(q_i, \alpha) = \delta(q_j, \alpha) = q \in Q_s$.

A loss-tolerant cluster ensures that although the monitor cannot make a transition during the loss, it can still reach a state equal to that of an RV-LTL monitor processing the full trace, when it observes subsequent events. Given the same input from Σ_-, states in a loss-tolerant cluster all transit to the same next state within the same loss-tolerant cluster. If a loss happens when the current state of the monitor belongs to a loss-tolerant cluster, the transitions of the cluster ensure that the state would still be one of the same cluster. After the loss, the monitor can process subsequent known events, and it is ensured that the monitor can precisely determine its current state if it processes at least one element of Σ_-.

LEMMA 1 (MONITORABILITY WITH TRANSIENT LOSS). *A formula* φ *can be monitored in the presence of a transient loss if* $\exists \Sigma_-, \exists Q_s$*, such that* $\forall \alpha \in \Sigma, \forall (q_i, q_j) | q_i \in Q_s \wedge q_j \in Q_s, ((\alpha \in \Sigma_-) \to (\delta(q_i, \alpha) = \delta(q_j, \alpha) = q \in Q_s \wedge q \notin Q_f))$.

PROOF. The proof to Lemma 1 trivially follows from Definition 6 and Definition 7. Thus, a formula φ is monitorable in the presence of a transient loss if we can identify a loss-tolerant alphabet Σ_- and a loss-tolerant cluster Q_s such that each element $\alpha \in \Sigma_-$ allows $\mathcal{M}^\varphi$ to transition to the same non-final state in Q_s. □

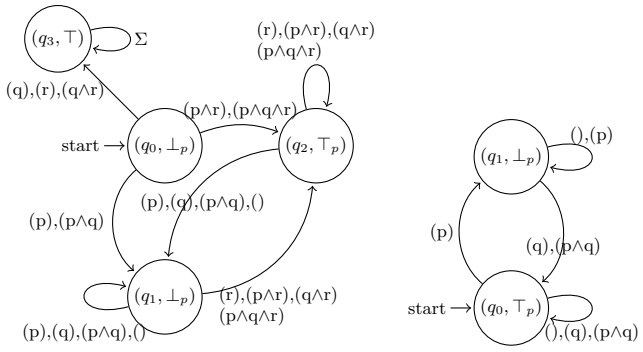
4.2 Motivating Examples

The formulas in the *obligation* [6] class can be satisfied or violated over a finite prefix of a trace. Under transient loss, the monitor may not be able to observe some events in such a finite prefix. Thus, in general, the formulas in obligation cannot be soundly monitored unless a recovery of the lost events is possible. For a safety formula for example, a missing event may cause a violation of the property, but the monitor cannot report the violation unless the lost event is recovered. Similar can be argued for a guarantee formula. However, in some cases, the satisfaction or violation status of obligation formulas can be determined despite the transient loss in the trace. For example, let us consider the LTL formula $\varphi_1 = \Box\, p$, where p is an atomic proposition. If φ_1 is already violated by a finite prefix of a trace, then a transient loss of subsequent events will not change the verdict.

Further, the satisfaction or violation of certain LTL formulas cannot be determined with a finite prefix of a trace. For example, let $\varphi_2 = p \to \Box\Diamond(q \mathbf{U} r)$ be an LTL formula; p, q and r are atomic propositions. Figure 1a represents an RV-LTL [4] monitor for φ_2. Once in state q_1 or q_2, the monitor can never deliver its verdict as $\top$ or $\bot$, because φ_2 can never be satisfied or violated by a finite word. This monitor can provide a sound verdict in the presence of a transient loss, because even though its current state may be unknown, the monitor can still provide a correct verdict for the next element in the trace provided its state immediately before the loss is either q_1 or q_2. Thus, the set of loss-tolerant clusters consists of a single cluster $Q_s = \{q_1, q_2\}$ and the loss-tolerant alphabet $\Sigma_- = \Sigma$. The states in Q_s upon receiving an identical element from Σ_- as the input yield a transition to the same next state. Although its previous state is unknown due to a transient loss, the monitor can reach a correct next state. During the loss, the RV-LTL monitor's verdict is unknown for the formula φ_2, because the current state of the monitor is unknown. However, once the monitor observes the subsequent events, it can determine a verdict, which is same as that of a monitor which processes the complete trace.

Now, let us consider $\varphi_3 = \Box(p \to \Diamond q)$, whose RV-LTL monitor is as shown in Figure 1b. This monitor can deliver a sound verdict for a lossy trace provided the part of the trace that is observed after the loss, contains at least one input element $\sigma_i | \sigma_i \in \{p,\ q,\ p \wedge q\}$. Here, the loss-tolerant alphabet is $\Sigma_- = \{p,\ q,\ p \wedge q\}$. The loss-tolerant alphabet, for instance, does not include the subset $\{\neg p \wedge \neg q\}$.

5. ALGORITHM DESIGN



(a) $\varphi_2 = p \rightarrow \Box\Diamond(q \,\mathbf{U}\, r)$ (b) $\varphi_3 = \Box(p \rightarrow \Diamond q)$

Figure 1: RV-LTL Monitors for LTL Formulas φ_2 and φ_3

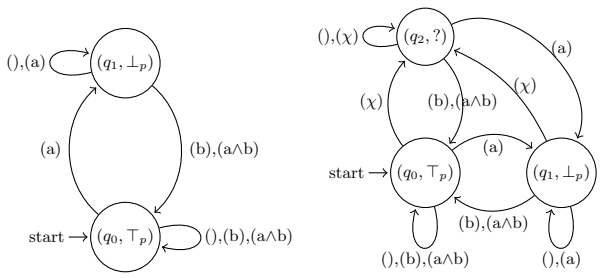
This section describes our algorithm to construct a loss-tolerant monitor for monitorable LTL formulas.

5.1 Synthesis of Loss-Tolerant Monitors

Algorithm 1 describes a process to find whether an LTL formula φ is monitorable on a lossy trace. Thus, if φ is monitorable, the main procedure FINDMONITORABLITY outputs $\top$ and synthesizes a loss-tolerant monitor $\mathcal{M}^{\varphi}$. Otherwise, it returns $\bot$ with the corresponding RV-LTL monitor. The monitor construction of FINDMONITORABLITY, Line 3, is based on the method of Bauer *et al.* [4]. The function checks $\mathcal{M}^{\varphi}$, from Line 4 to Line 16, for possible combinations of a loss-tolerant cluster and a loss-tolerant alphabet that satisfy Lemma 1. These combinations are stored in the ascending order of the size of clusters at Line 13. All clusters with at least two states are verified. The reason being that single-state clusters that satisfy Lemma 1 consist of a final state.

From Line 17, Algorithm 1 processes the list generated at Line 13. If at least one combination satisfying Lemma 1 is found at Line 10, φ may be monitored on a lossy trace. MEETSMONITORABILITY chooses the maximum loss-tolerant alphabet of respective loss-tolerant cluster, which involves checking matching transitions for states of the cluster at Line 45. Further, the function ADDCHITOCLUSTER is called at Line 18 to synthesize a loss-tolerant monitor by augmenting $\mathcal{M}^{\varphi}$ with new states and transitions. For each combination, if a transition to one of the states in the corresponding loss-tolerant cluster is not defined for input χ, a new state is created with output as '?'. The states in the cluster then perform a transition to the newly added state, when the input is χ. Further, for each element in the loss-tolerant alphabet, the new state makes transitions to one of the states in the loss-tolerant cluster. Also, for each element not in the loss-tolerant alphabet, the new state performs a transition to itself. FINDMONITORABLITY connects all other states that are not in the list of clusters identified at Line 13 to a common unknown state as seen at Line 19.

To further understand Algorithm 1, let us consider the LTL formula $\varphi = \Box(a \rightarrow \Diamond b)$ and its two-state monitor, i.e., $Q = \{q_0, q_1\}$, shown in Figure 2a. FINDMONITORABLITY identifies the tuple $(\{q_0, q_1\}, \{a \wedge b, a, b\})$ as the only combination of a loss-tolerant cluster and alphabet that satisfies Lemma 1. ADDCHITOCLUSTER adds a new state labeled as



(a) RV-LTL Monitor (b) Transformed Monitor

Figure 2: Monitor Synthesis for $\varphi = \Box(a \rightarrow \Diamond b)$

'unknown' to the identified cluster. This new state outputs a verdict as '?'. For every state in the cluster $\{q_0, q_1\}$, a new transition is added for the symbol χ, and such transitions yield the new state. The transitions of the new state for all symbols in Σ_- are same as that of any of the states in the cluster. For every element not in Σ_-, but in Σ, a new transition is added from the new state to itself. The transformed RV-LTL monitor for φ appears in Figure 2b.

As discussed by Bauer *et al.* [4], the computation time of SYNTHESIZERVLTLMONITOR is exponential w.r.t. the size of φ in the worst-case. If m is the size of Σ and n the number of state in $\mathcal{M}^{\varphi}$, the complexity of MEETSMONITORABILITY is $O(m \times n)$. Similarly, the complexity of ADDCHITOCLUSTER is $O(m + n)$. Thus, the worst-case time complexity of the inner section of Algorithm 1, i.e, from Line 7 to Line 16, is $O(m \times n \times 2^n)$. Therefore, the time complexity of Algorithm 1 is exponential w.r.t. the number of states of $\mathcal{M}^{\varphi}$.

The number of new states added to the RV-LTL monitor is bounded by n. Therefore, the size of the synthesized loss-tolerant monitor $\mathcal{M}^{\varphi}$ is also in $O(2^{2^p})$, where p is the size of the formula φ. Thus, the size complexity of the loss-tolerant monitor is identical to that of the RV-LTL monitor [4].

5.2 Correctness of Loss-Tolerant Monitors

A loss-tolerant monitor $\mathcal{M}^{\varphi}$ should guarantee soundness. Lemma 2 states that the verdict of $\mathcal{M}^{\varphi}$ matches that of an RV-LTL monitor processing the complete trace, provided for instance that a loss-tolerant alphabet, Σ_-, and a loss-tolerant cluster, Q_s, exist.

LEMMA 2 (CONDITIONAL EQUALITY OF VERDICTS). *Let $\mathcal{M}^{\varphi} = (\Sigma, Q, q_0, \delta, \lambda)$ be a loss-tolerant monitor for φ to process a lossy trace $\sigma_i = a_1, a_2, \ldots, a_n$ and $\mathcal{M}'^{\varphi} = (\Sigma, Q', q'_0, \delta', \lambda')$ the corresponding RV-LTL monitor to process a complete trace $\sigma_j = b_1, b_2, \ldots, b_n$. $\exists! \, I = [i_1, j_1]$ s.t. $\forall k : (a_k = b_k) \vee (i_1 \leq k \leq j_1 \wedge a_k = \chi) \wedge (j_1 < n)$. The verdict of $\mathcal{M}^{\varphi}$ is equal to $\mathcal{M}'^{\varphi}$ iff: $\exists(Q_s, \Sigma_-) \cdot \exists m \in (j_1, n] \cdot \forall k \in [1, n]$ $(((a_{i_1} = \chi \wedge q_{i_1 - 1} \in Q_s) \wedge (a_m \in \Sigma_-)) \iff ((k \geq m \vee k < i_1) \wedge \lambda(\delta(q_{k-1}, a_k)) = \lambda'(\delta'(q'_{k-1}, b_k))) \vee (i_1 \leq k < m \wedge \lambda(\delta(q_{k-1}, a_k)) = \text{'?'})))$.*

PROOF. We start with the first part of Lemma 2, i.e., $\exists(Q_s, \Sigma_-) \cdot \exists m \in (j_1, n] \cdot \forall k \in [1, n] \, (((a_{i_1} = \chi \wedge q_{i_1 - 1} \in Q_s) \wedge (a_m \in \Sigma_-)) \longrightarrow ((k \geq m \vee k < i_1) \wedge \lambda(\delta(q_{k-1}, a_k)) = \lambda'(\delta'(q'_{k-1}, b_k))) \vee (i_1 \leq k < m \wedge \lambda(\delta(q_{k-1}, a_k)) = \text{'?'}))$.

If the current input is χ and M^{φ}'s state at the beginning of the loss belongs to a loss-tolerant cluster, then M^{φ}'s next

```
1  FindMonitorablity(φ){
2   isMonitorable = false
3   M^φ = SynthesizeRvltlMonitor(φ)
4   n = FindNoOfStates(M^φ)
5   if (n ≤ 1)
6    return (isMonitorable, M^φ)
7   for(i = 2; i< n; i++) {
8    for(j = (n choose i); j > 0; j--){
9     Q_s = next Cluster of size i of states of M^φ
10    (ms, Σ_) = MeetsMonitorability(Q_s)
11    if(ms != false){
12     isMonitorable = true
13     clList = AddClToList(Q_s, Σ_)
14    }
15   }
16  }
17  for(each (Q_s, Σ_) ∈ clList)
18   AddChiToCluster(Q_s, Σ_)
19  Create chiState s.t. λ(chiState) = '?'
20  for(q | ∀Q_s ∈ clusterList s.t. q ∉ Q_s)
21  if(q is not a final state)
22   δ(q, χ) = chiState
23  for(each β ∈ Σ ∪ χ)
24   δ(chiState, β) = chiState
25  return (isMonitorable, M^φ)
26 }
```

```
27 AddChiToCluster(Q_s, Σ_){
28  /*Add unknown state to
29  loss-tolerant cluster*/
30  Create chiState such that λ(chiState) = '?'
31
32  q = any state s such that s ∈ Q_s
33
34  for(each β ∈ Σ_)
35   δ(chiState, β) = δ(q, β)
36
37  for(each q ∈ Q_s ∪ chiState)
38   if(δ(q, χ) undefined)
39    δ(q, χ) = chiState
40  for(each β ∉ Σ_ ∧ β ∈ Σ)
41   δ(chiState, β) = chiState
42 }
43
44 MeetsMonitorability(Q_s){
45  if(Q_s satisfies Lemma 1){
46  /*Check performed
47  with elements of Σ*/
48   Find Σ_ for Q_s
49   return (⊤, Σ_)
50  }else
51   return (⊥, null)
52 }
```

Algorithm 1: Synthesis of a loss-tolerant monitor

state is one of the new states with output '?'. This state was earlier added to M^φ by Algorithm 1 via AddChiToCluster, which post-conditions ensure that all states in a cluster transition to a state with output '?' on an input χ. Further, M^φ remains in this state until it can process an element, $a_m \in \Sigma_-$. Hence, its output remains '?' when $i_1 \leq k < m$.

Further, when M^φ observes a_m at the end of the loss, it moves to a state, equal to that of M'^φ as per the transition function defined in AddChiToCluster at Line 34. The post-conditions of procedures MeetsMonitorability and AddChiToCluster ensure that such equality between the states of M^φ and M'^φ exists. Thus, for $k \geq m$, M^φ's output remains equal to that of M'^φ. When $k < i_1$, i.e., before the beginning of the loss, the outputs of M^φ and M'^φ are equal as both perform identical transitions on identical inputs.

Now, we prove the second part of Lemma 2, i.e., $\exists(Q_s, \Sigma_-) \cdot \exists m \in (j_1,\, n] \cdot \forall k \in [1, n]\, ((((k \geq m \vee k < i_1) \wedge \lambda(\delta(q_{k-1}, a_k)) = \lambda'(\delta'(q'_{k-1}, b_k))) \vee (i_1 \leq k < m \wedge \lambda(\delta(q_{k-1}, a_k)) = \text{'?'})) \implies ((a_{i_1} = \chi \wedge\ q_{i_1-1} \in Q_s) \wedge\ (a_m \in \Sigma_-)))$.

Let us assume the output of M^φ is equal to that of M'^φ, when $k < i_1 \vee k \geq m$ and M^φ's output is '?', when $i_1 \leq k < m$. It is evident that $a_{i-1} = \chi$ as per the monitor construction by Algorithm 1 where an unknown state has its output as '?'. The loss begins at $i-1$, because the output of M^φ is '?' starting at index i_1. Further, as the outputs of M^φ and M'^φ are equal for the m^{th} element of the trace, a_m must belong to one of the loss-tolerant alphabets, and the state q_{i_1-1} of M^φ before the processing of the ${i_1}^{th}$ element must belong to the corresponding loss-tolerant cluster. Unless q_{i_1-1} belongs to the same loss-tolerant cluster, M^φ cannot produce an output equal to that of M'^φ upon processing a_m, which belongs to the loss-tolerant alphabet. □

With Lemma 2, a monitor $\mathcal{M}^\varphi$ may provide a sound verdict if its state at the start of the loss belongs to a loss-tolerant cluster. This guarantees that, after the loss, $\mathcal{M}^\varphi$ can move to a correct next state by processing an input from Σ_-. Theorem 1 states the conditions for the correctness of $\mathcal{M}^\varphi$.

Theorem 1 (Correctness of $\mathcal{M}^\varphi$). *Let $\mathcal{M}^\varphi$ be a loss-tolerant monitor for an* Ltl *formula φ. For all $\sigma \in \Sigma^*$, if σ exhibits a transient loss of events, and a run of $\mathcal{M}^\varphi$ on σ satisfies (1) the state of $\mathcal{M}^\varphi$ immediately before each lossy interval belongs to one of the loss-tolerant clusters and (2) the subsequent trace after the end of the loss contains at least one element from the corresponding loss-tolerant alphabet, then the following holds: $\lambda(\delta(q_0, \sigma)) = [\sigma \models \varphi]_{RV}$.*

Proof. The proof directly follows from that of Lemma 2, Definition 5, and the proofs of correctness provided by Bauer *et al.* for Ltl_3 [5] and RV-LTL [4] monitors. □

It is important for the verdict of a monitor not to be invalidated upon the recovery of lost events. To express this idea, we define the *monotonicity* of monitors w.r.t. lost events.

Definition 8 (Monotonicity w.r.t. past events). *A monitor is* monotonic *if its verdict on a lossy finite word cannot be invalidated by the loss recovery in the finite word.*

Corollary 1. *A loss-tolerant monitor $\mathcal{M}^\varphi$ built with Algorithm 1 is monotonic.*

Proof. This proof trivially follows from that of Theorem 1 and Lemma 2. □

5.3 Performance

We measured the performance of Algorithm 1 on common patterns of LTL formulas [5, 8]. A total of 42 formulas were identified as monitorable from the 97 available. We note that some patterns such as $\Box(a \longrightarrow (b \wedge c))$ cannot be soundly monitored under transient loss. Figure 3a compares the number of states of RV-LTL monitors with corresponding loss-tolerant monitors. It shows that the additional size overhead incurred by loss-tolerant monitors is not significant and represents only an increase of at most two from that of the corresponding RV-LTL monitor. Similarly, Figure 3b compares the number of transitions, which depends on the number of states and the size of the alphabet. The number of transitions depicts the additional overhead in terms of memory at run time. We observe a minimum of 5 and a maximum of 534 extra transitions w.r.t. that of RV-LTL monitors. Consequently, the constructed monitors incur minimal extra overhead at run time.

6. EVALUATION

We evaluated our approach on MPlayer 1.1_4.8 [23], a cross-platform media player, and Google cluster traces [24, 18]. We used a computer with 31.4Gb of RAM running Ubuntu 14.04 LTS 64 bit on an 8-core Intel i7-3820 CPU at 3.60GHz. We implemented our loss-tolerant monitor generator in Java using RV-LTL monitors generated with LamaConv [21].

6.1 MPlayer

We monitored MPlayer while playing a high definition, 29.97 fps, 720x480, 1 Mbps bitrate video, to evaluate the following:

Property 1: $\varphi_1 :: \Box play \rightarrow \Box\,(buffer \rightarrow \Diamond decode)$. This property states that whenever MPlayer plays a file, a buffer action is always followed by an eventual decode action.
Property 2: $\varphi_2 :: \Box\,(play \rightarrow \Diamond(audio \vee (video \wedge subtitle) \vee pause))$. This expresses that when MPlayer plays a file, it may be an audio file or a video with subtitle or just paused.

Settings. We use DIME [2], a periodic dynamic binary instrumention (DBI) tool, as a lossy observer. For a period of T time units, DIME instruments only for a predefined time budget $B \leq T$, thus, extracting no data for $T-B$ time units. We modified DIME to output χ when no data is extracted. We use PIN [13], a DBI framework, to generate complete traces for comparison purpose.

Results. We instrumented MPlayer with both DIME and PIN to track function calls related to properties 1 and 2. We varied the budget of DIME by 10 from 10% to 100% and fixed the period to 1 second. We repeated each experiment five times. Figure 4a plots a portion of the truth-values generated by the monitors for the trace obtained from the first runs of DIME with 10% budget and PIN. The x-axis represents the index of each symbol in the input trace, the y-axis, the truth values. We observe that the output of the loss-tolerant monitor matches that of the RV-LTL monitor except for the χ symbol. For example, the verdict of the synthesized monitor for the 55^{th} input symbol is '?' while that of the RV-LTL monitor is $\top_p$, suggesting that the input was a χ. The final verdict is $\top_p$ for both the loss-tolerant and the RV-LTL monitor for the two properties.

Further, we analyzed the run time overhead of the monitors as shown in Figure 4b and Figure 4c for property 1 and 2, respectively. Both figures show that it is more expensive to monitor a complete trace than a lossy trace.

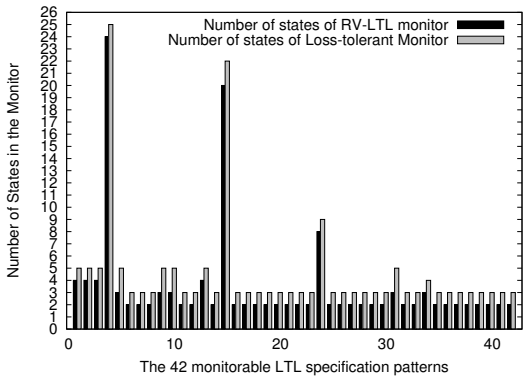
We also compared the number of lost events to that of dropped video frames. Figure 4d shows the results for Property 1. The x-axis represents the budget. The lost events appear of the left y-axis, while the right y-axis represents the dropped frames. The percentage of lost events w.r.t. processed events are also shown in Figure 4d. We observe that more frames are dropped as the budget increases, while on the contrary, the number of lost events decreases as the budget increases. With 80% budget for example, the number of lost events is around 60% with 1000 frames dropped. When the budget is at 100%, DIME still generate about 47% of loss. This is because DIME turns off instrumentation until the budget is next replenished. The video hardly plays with PIN as nearly 3000 frames are dropped. Figure 4d suggests that for a smooth video experience, it is preferable to monitor while instrumenting with a 10% budget per second. This result is interesting as it shows that for monitorable LTL formulas, one can achieve sound and *lightweight* monitoring using a lossy observer like DIME. More, we can drastically reduce the monitoring time of such formulas on large traces by deliberately injecting losses into the traces.

6.2 Google Cluster Traces

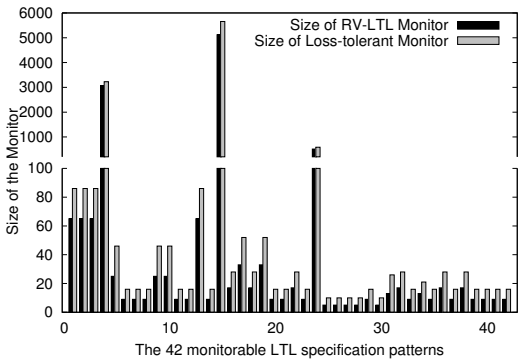
A Google cluster is a set of machines, packed into racks, and connected by a high-bandwidth cluster network [18, 24]. The cluster runs jobs, each comprising of one or more tasks. The ClusterData2011_2 [18, 24] used in this paper, provides data from a 12.5k-machine cell over about a month-long period in May 2011. Data in the traces are periodically collected from the machines. When the monitoring system or cluster gets overloaded, data may not be collected. However, when an event is lost, a reconstruction is attempted and a 'missing info' field set to state the reason. Each task progresses through different states such as 'submitted', 'scheduled', 'failed', 'evicted', and 'killed' throughout its lifecycle. The trace is lossy since some states for some jobs and tasks are not captured due to unknown reasons. We extracted $7,170,572$ events for $16,467$ tasks and verified the following properties on their state transitions.

Property 1: $\Box((fail \vee kill \vee evict) \rightarrow \Box(submit \rightarrow \Diamond finish))$. This states that whenever a task fails or is killed or is evicted, it is always re-submitted, and eventually finishes.
Property 2: $\Box(evict \rightarrow \Diamond submit)$. Means that whenever a task is evicted, it is eventually re-submitted.
Property 3: $\Box(schedule \rightarrow \Diamond(finish \vee fail \vee kill \vee evict))$. This states that whenever a task is scheduled, it will eventually finish, fail, get killed or evicted.
Property 4: $\Box(update\_pending \rightarrow \Diamond schedule)$. This states that whenever a task is updated at runtime before been scheduled, it is eventually scheduled for execution.
Property 5: $\Box(submit \rightarrow \Diamond finish)$. This states that whenever a task is submitted, it eventually gets finished.

The monitoring results appear in Table 1 with the monitored tasks in the first column. Some tasks were skipped as seen in the second column, because their traces did not meet the transience criteria of Definition 5. For Property 1 for instance, the loss-tolerant monitor delivered verdict as $\top_p$

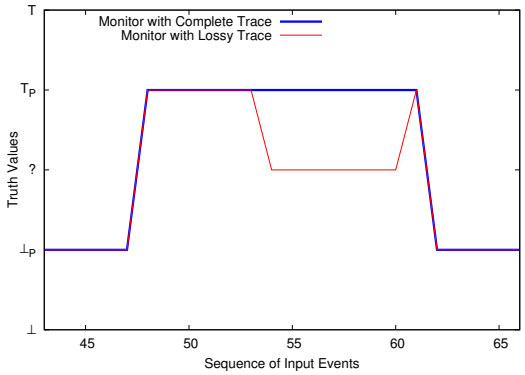


(a) States

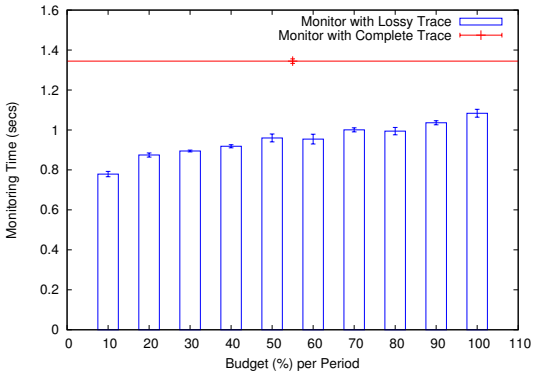


(b) Transitions

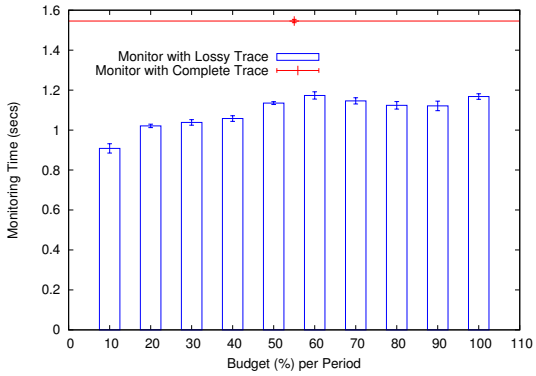
Figure 3: Size Comparison for RV-LTL and Loss-tolerant Monitors



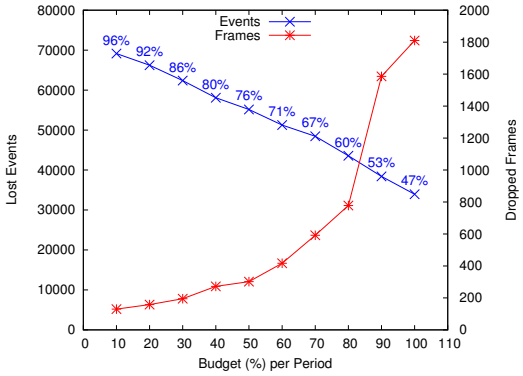
(a) Comparison of Verdicts



(b) Property 1



(c) Property 2



(d) Dropped Frames vs Lost Events

Figure 4: MPlayer's Monitoring Results

for 3,572 tasks and $\perp_p$ for 7,275 out of the 10,847 processed tasks. A total of 5,620 tasks were skipped. These results confirm that important LTL properties can still be soundly monitored with low overhead on lossy traces.

7. RELATED WORK

While most RV techniques [5, 11, 10, 15] assume the existence of complete traces, a few approaches have been developed for lossy traces. However, the probabilistic approach taken by Stoller *et al.* [22] may have its verdict changed with the recovery of lost events. Our loss-tolerant monitor guarantees a sound verdict on lossy traces for monitorable specifications irrespective of the recovery of lost events. Basin *et al.* [3] used a three-valued semantics to express uncertainties about the verdict on lossy traces. However, to resolve such uncertainties, the approach may require a recovery of lost events. Contrarily, our approach identifies monitorable LTL properties for which a recovery of lost events is not needed

Table 1: Verification of Google Cluster Data

Prp.	Mon. Tasks	Skipped Tasks	Verdict $\top_p$	Verdict $\bot_p$	Avg. Time	Stderr
1	10,847	5,620	3,572	7,275	7.68s	0.036
2	10,892	5,575	10,891	1	7.49s	0.047
3	10,892	5,575	6,240	4,652	8.20s	0.026
4	16,467	0	16,461	6	7.15s	0.030
5	10,892	5,575	2,876	8,016	7.63s	0.055

to deliver a sound verdict. Alechina *et al.* [1] developed techniques for monitoring multi-agent systems based upon norms, which are described using obligations and prohibitions. These norms are approximated so that they can be monitored under partial observability. While this approach requires full knowledge of the transition system, our technique only considers runtime execution paths, and does not alter the LTL specifications. Criado and Such [7] also monitored multi-agent systems with norms based on approximate reconstruction of unobserved actions. To our knowledge, our approach is the first that does not depend on the recovery of lost events to produce a sound verdict.

8. CONCLUSIONS

In this paper, we presented an offline algorithm that identifies whether an LTL formula can be *soundly* monitored on a lossy trace and constructs the corresponding loss-tolerant monitor. We analyzed the complexity of the algorithm and also evaluated it against commonly used patterns of LTL formulas to show that the synthesized monitors incur minimal additional overhead in terms of size of monitors. Further, we evaluated the algorithm on traces from two real-world systems to show its effectiveness and applicability. The evaluation shows that our approach increases the applicability of RV to real-world applications which often produce lossy traces. Finally, we define the concept of monotonicity to evaluate the soundness of monitors, especially when lost events are not recovered. In the future, we plan to extend the method to timed traces and distributed systems.

9. REFERENCES

[1] N. Alechina, M. Dastani, and B. Logan. Norm approximation for imperfect monitors. In *Autonomous Agents and Multi-Agent Systems*, pages 117–124, 2014.

[2] P. Arafa, H. Kashif, and S. Fischmeister. DIME: Time-aware Dynamic Binary Instrumentation Using Rate-based Resource Allocation. In *Proc. of Int'l Conf. on Embedded Software*, pages 25:1–25:10. IEEE, 2013.

[3] D. Basin, F. Klaedtke, S. Marinovic, and E. Zalinescu. Monitoring Compliance Policies over Incomplete and Disagreeing Logs. In *Proc. of Runtime Verification*, pages 151–167. Springer, 2013.

[4] A. Bauer, M. Leucker, and C. Schallhart. Comparing LTL Semantics for Runtime Verification. *J. Log. Comput.*, 20(3):651–674, 2010.

[5] A. Bauer, M. Leucker, and C. Schallhart. Runtime Verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, Sept. 2011.

[6] E. Chang, Z. Manna, and A. Pnueli. Characterization of Temporal Property Classes. In *Int'l Colloquium*, pages 474–486. Springer, 1992.

[7] N. Criado and J. M. Such. Norm Monitoring under Partial Action Observability. *IEEE Trans. on Cybernetics*, PP(99):1–13, Jan. 2016.

[8] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in Property Specifications for Finite-State Verification. In *Softw. Eng.*, pages 411–420, 1999.

[9] Y. Falcone, J.-C. Fernandez, and L. Mounier. What can you verify and enforce at runtime? *Int. J. Softw. Tools Technol. Transf.*, 14(3):349–382, 2012.

[10] K. Havelund and G. Rosu. Efficient monitoring of safety properties. *Int. J. Softw. Tools Technol. Transf.*, 6(2):158–173, 2004.

[11] J. Huang, C. Erdogan, Y. Zhang, B. M. Moore, Q. Luo, A. Sundaresan, and G. Rosu. ROSRV: Runtime Verification for Robots. In *Proc. of Runtime Verification*, pages 247–254, 2014.

[12] L. Kong, M. Xia, X.-Y. Liu, G. Chen, Y. Gu, M.-Y. Wu, and X. Liu. Data Loss and Reconstruction in Wireless Sensor Networks. *IEEE Trans. on Parallel and Distributed Systems*, 25(11):2818–2828, Nov 2014.

[13] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. *SIGPLAN Not.*, 40(6), June 2005.

[14] Z. Manna. *Temporal Verification of Reactive systems: Safety*. Springer, 1995.

[15] J. Morse, L. Cordeiro, D. Nicole, and B. Fischer. Model Checking LTL Properties over ANSI-C Programs with Bounded Traces. *Softw. Syst. Model.*, 14(1):65–81, Feb. 2015.

[16] OProfile. http://oprofile.sourceforge.net/news/, 2002.

[17] A. Pnueli. The Temporal Logic of Programs. In *Proc. of Foundations of Computer Science*. IEEE, 1977.

[18] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Nov. 2011.

[19] U. Sammapun, I. Lee, and O. Sokolsky. RT-MaC: Runtime Monitoring and Checking of Quantitative and Probabilistic Properties. In *Proc. of RTCSA*, pages 147–153. IEEE, 2005.

[20] U. Sammapun, I. Lee, O. Sokolsky, and J. Regehr. Statistical Runtime Checking of Probabilistic Properties. In *Proc. of Runtime Verification*, pages 164–175. Springer, 2007.

[21] T. Scheffel, M. Schmitz, S. Hungerecker, M. Kabelitz, C. Krüger, and J. Thorn. LamaConv: Logics and Automata Converter Library. http://www.isp.uni-luebeck.de/lamaconv, 2015.

[22] S. D. Stoller, E. Bartocci, J. Seyster, R. Grosu, K. Havelund, S. A. Smolka, and E. Zadok. Runtime Verification with State Estimation. In *Proc. of Runtime Verification*, pages 193–207. Springer, 2012.

[23] The MPlayer Project. http://www.mplayerhq.hu.

[24] J. Wilkes. More Google cluster data. Google research blog, Nov. 2011. http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.