Model-Based Testing of Probabilistic Systems with Stochastic Time

Marcus Gerhold and Mariëlle Stoelinga(✉)

University of Twente, Enschede, The Netherlands
m.gerhold@utwente.nl, marielle@cs.utwente.nl

Abstract. This paper presents a model-based testing framework for black-box probabilistic systems with stochastic continuous time. Markov automata are used as an underlying model. We show how to generate, execute and evaluate test cases automatically from a probabilistically timed requirements model. In doing so, we connect classical **ioco**-theory with statistical hypothesis testing; our **ioco**-style algorithms test for functional behaviour, while χ^2 hypothesis tests and confidence interval estimations assess the statistical correctness of the system.

A crucial development are the classical soundness and completeness properties of our framework. Soundness states that test cases assign the correct verdict, while completeness states that our methods are powerful enough to discover each discrepancy in functional or statistical misbehaviour, up to arbitrary precision.

We illustrate our framework via the Bluetooth device discovery protocol.

1 Introduction

The role of computer-based systems is ever increasing: robots, drones and autonomous cars will soon pervade our lives. Attuning to this progress, verification and validation techniques of these systems have grown to a field of crucial importance. They provide methods that show whether the actual and the intended behaviour of a system differ, or give confidence that they do not.

Conversely, the progressively intricate design of embedded systems continuously brings new challenges to the field of verification engineers. The key question of whether a system works as intended therefore has a variety of angles: Was the functional behaviour correctly implemented? Does the system continue to operate under a work overload? Is the average lifetime within safety regulations? Can requirements be met on time?

Probabilistic aspects in many computer applications naturally add one of those angles. Security protocols use random bits in their encryption methods [9], control policies in robots lead to the emerging fields of probabilistic robotics [46],

This research has been partially funded by STW and ProRail under the project ArRangeer (12238), STW under the project SEQUOIA (15474), NWO under the project BEAT (612.001.303), NWO under the project SamSam (628.005.015), and the EU under the project SUCCESS (509-18240).

S. Gabmeyer and E.B. Johnsen (Eds.): TAP 2017, LNCS 10375, pp. 77–97, 2017.
DOI: 10.1007/978-3-319-61467-0_5

hidden Markov chains are used in speech recognition [39] and communication protocols are often equipped with a stochastic delay [15,44]. Therefore, there is a natural demand for a pendant in the verification and validation community that accounts for probabilistic aspects.

Testing. To investigate such questions, probabilistic verification has become a mature research field with techniques like stochastic model checking (SMC) [38] based on models like probabilistic automata [40], Markov decision processes [37], generalised stochastic Petri nets [33] or stochastic automata [12]. These techniques are complemented with tools like PRISM [29], PLASMA [27] or the MODEST toolset [20].

In practice, however, the most common validation technique is testing. The system is subjected to many well-designed test cases and the outcome is compared to a specification. A verdict, i.e. *pass* or *fail*, is then given based on the expectations.

This paper presents a model-based testing (MBT) approach that can handle probabilistic and stochastic-time aspects in systems. MBT gained a lot of traction in recent years in both academia and industry. It mirrors the faster development of systems, by providing access to faster test methods due to automation. Test cases are automatically generated, executed and evaluated based on a requirements specification. A number of industrial and academic MBT tools have been developed, such as TorXakis [35], MaTeLo [19], UPPAAL Tron [22] or SpecExplorer [51].

There is a large body of different frameworks that accommodate a variety of requirements aspects, like functional properties [50], real-time [2,6,30], quantitative aspects [3,5] and coverage [7]. Surprisingly, only few papers are concerned with the testing of probabilistic systems[1], with some notable exceptions being [24–26].

We present an applicable framework in an MBT setting, that is capable of verifying if probabilistic choices made by the system itself were implemented correctly. Furthermore, the approach also accommodates stochastic-time aspects of systems, such as specified delays, degradation rates or intended waiting periods. This is of particular interest, if only the mean duration of an activity is known.

Our Approach. The foundation of our methodology are Markov automata (MAs). MAs are equipped with both probabilistic and nondeterministic choices. The first represent choices made by the system (e.g. coin tosses or random seeds) or the environment (e.g. degradation rates or failure probabilities). The latter model choices that are not under its control. As widely agreed [40,43], nondeterminism is crucial for implementation freedom, scheduling choices and interleaving. Complementary, they are of particular interest because of their memoryless exponential distributed timed transitions. These give a highly appropriate stochastic approximation, if only the mean duration of an activity is known, as is often the case in a practical setting. Mathematically, MAs arise as the

[1] The topic of statistical testing, e.g. [1,52], is concerned with choosing test inputs probabilistically; it does not check the correctness of the random choices made by a system itself.

conservative extension of both probabilistic automata (PAs) [40] and interactive Markov chains (IMCs) [21].

An important contribution are our algorithms that automatically generate, execute and evaluate probabilistic test cases from a specification MA. They check the functional, probabilistic and stochastic-time behaviour of the system. Probabilities are observed via frequencies, hence, test cases need to be repeated multiple times. We use statistical hypothesis testing, in particular χ^2 testing, to assess whether a test should *pass* or *fail*.

To account for the correctness of our framework, we prove it to be *sound* and *complete*. Soundness states that each test case assigns the correct verdict, while completeness (a.k.a. exhaustiveness) guarantees that the test method is powerful enough to discover each deviation from the requirements. Phrasing these results requires a mathematical notion of conformance. We propose the **Mar-ioco** relation, an implementation relation that pins down precisely when an implementation modelled as an MA conforms to a requirements specification model. We prove **Mar-ioco** to be a conservative extension to the **ioco** relation known from MBT literature [47,50]. Lastly, we provide a case study on the Bluetooth device discovery protocol showing the applicability of our framework.

While test efficiency is essential, this paper focusses on the methodological set up and correctness. Imperative future research is to optimize the statistical verdicts we give and provide fully fledged tool support.

We summarize our key contributions:

1. The general input output Markov automata model comprising discrete probability distributions, non-deterministic choices and exponentially delayed transitions,
2. a behavioural description for Markov automata based on trace semantics,
3. solid definitions of probabilistic test cases, test execution and verdicts,
4. the treatment of the absence of outputs in a stochastically time delayed setting and
5. the soundness and completeness results of our framework.

Related Work. There is a large body of work on testing real-time systems [2,6,28,30]. Briones and Brinksma [6] extend the framework to incorporate the notion of quiescence, i.e. the absence of outputs.

Conversely, probabilistic testing preorders and equivalences are well-studied [10,14,40]. Distinguished work by [31] introduces the concept of probabilistic bisimulation via hypothesis testing. Largely influential work is given by [8], presenting how to observe trace frequencies during a sampling process. Executable probabilistic test frameworks are suggested for probabilistic finite state machines in [23,26] and Petri nets [4].

Closely related to our work is the study of Markovian bisimulation. The foundation of an observational equivalence is presented in [16] in the form of weak bisimulation for Markov automata, and was refined by introducing late-weak bisimulation [13,42] and branching bisimulation [49].

This paper is an extension of earlier work [17] that investigated the test process in the probabilistic setting and a workshop paper [18] sketching how

stochastic time and exponential delays can be incorporated. Novel contributions of the current version are the complete integration of stochastic-time delays and the treatment of quiescence.

Overview Over the Paper. In Sect. 2 we recall definitions of Markov automata. Section 3 describes how Markov automata are used in the testing process and Sect. 4 shows that our framework is sound and complete. We show experimental results in Sect. 5. The paper ends in Sect. 6 with conclusions and future work.

2 Markov Automata

We recall properties of Markov automata and show how nondeterminism is resolved. We assume that the reader is acquainted with the basics of probability theory, but recall integral definitions. In particular, we borrow the standard construction of probability spaces via σ-fields. See [11] for an excellent overview and further reading.

Probability. A *discrete probability distribution* over a set X is a function $\mu : X \to [0,1]$, such that $\sum_{x \in X} \mu(x) = 1$. The set of all distributions over X is denoted $Distr(X)$ and subdistributions $SubDistr(X)$ respectively.

Let Ω be a set, $\mathcal{F}$ a σ-field of Ω and $(\Omega, \mathcal{F})$ the resulting measurable space. A σ-additive function $\mu : \mathcal{F} \to [0,1]$ is called a *probability measure*, if $\mu(\Omega) = 1$. We denote the set of all probability measures over X by $Meas(X)$.

A *probability space* is a triple $(\Omega, \mathcal{F}, Pr)$, where Ω is a set, $\mathcal{F}$ is a σ-field of Ω and $Pr : \mathcal{F} \to [0,1]$ is a probability measure, such that $Pr(\Omega) = 1$ and $Pr(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} Pr(A_i)$ for $A_i \in \mathcal{F}$, $i = 1, 2 \ldots$ pairwise disjoint.

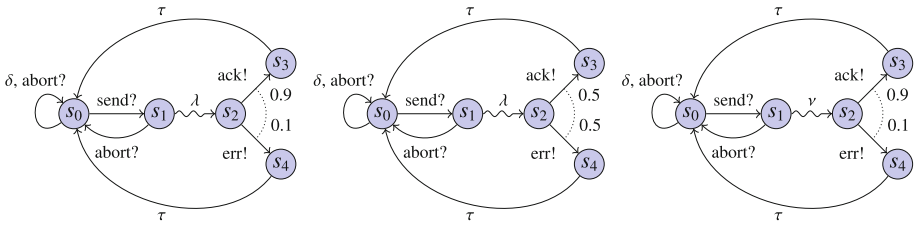
2.1 The Markov Automaton Model

Markov automata [48] comprise nondeterministic choices, discrete probability distributions and exponentially delayed transitions. They allow modelling choices made by the system (e.g. coin tosses) or the environment (e.g. degradation rates) and are an appropriate stochastic approximation, if only the mean duration of an activity is known.

Definition 1. *A* Markov automaton $\mathcal{M} = \langle S, s_0, L, \to, \rightsquigarrow \rangle$ *is a five-tuple, consisting of*

- S *a set of states, with* s_0 *the unique starting state,*
- L *a set of actions,*
- $\to \subseteq S \times L \times Distr(S)$*, the probabilistic transition relation and*
- $\rightsquigarrow \subseteq S \times \mathbb{R}_{\geq 0} \times S$*, the Markovian transition relation.*

An IOMA is an MA, where $L = L_i \sqcup L_o \sqcup L_\tau$ *is the disjoint union of input, output, and internal actions respectively, containing a special quiescence label* $\delta \in L_o$.



(a) Specification (b) Unfair Implementation (c) Slow Implementation

Fig. 1. Protocol specification IOMA and two erroneous implementations. After the input *send?* there is an exponentially delayed transition, followed by an acknowledgement or error output.

If we replace $\rightarrow$ *by* $\rightarrow' \subseteq S \times Distr\,(L \times S)$ *with the requirement that for all* $(s, \mu) \in \rightarrow'$ *if* $\mu\,(s, a) > 0$ *for an input* $a \in L_i$*, then* $\mu\,(s, b) = 0$ *for all* $b \neq a$*, the input output MA becomes* input-reactive *and* output-generative.

An action a is *enabled* in state s, if there is a distribution μ, such that $(s, \mu) \in \rightarrow$ and $\mu\,(a, s') > 0$ for some $s' \in S$. We write *enabled* $\{s\}$ for the set of enabled actions in s. A state is called *probabilistic*, if at least one action of L is enabled. A state is called *input-enabled*, if all actions of the set L_i are enabled. A state is called *Markovian*, if it has at least one transition $(s, \lambda, s') \in \rightsquigarrow$. The Markovian actions are parameters for the exponentially delayed transitions and therefore deemed invisible.

A distinctive feature of Markov automata are their exponentially distributed timed transitions, i.e. the set $\rightsquigarrow$. The *rate* to go from a state s to a state s' is the sum of all λ, such that $(s, \lambda, s') \in \rightsquigarrow$ and is denoted $\mathbf{R}\,(s, s')$. The sum of all rates in a state s is called *exit rate* of s and denoted by $\mathbf{E}\,(s)$. We require $\mathbf{E}\,(s) < \infty$ for all $s \in S$. The *delay* associated with a Markovian state is exponentially distributed with its exit rate. Multiple Markovian transitions in one state thus lead to a *race condition*. The probability to move from s to a successor s' equals the probability that (one of) the Markovian transitions leading from s to s' wins the race. This induces the *discrete branching probability distribution* $\mathbb{P}_s$ for s given by $\mathbb{P}_s\,(s') = \mathbf{R}\,(s, s')\,/\mathbf{E}\,(s)$.

A state is called *stable*, if it enables no internal action. We employ the maximal progress assumption, meaning that time is not allowed to progress in unstable states. This renders Markovian transitions in unstable states unnecessary [32].

Example 1. Fig. 1 shows three input-reactive output-generative IOMA. The model describes a protocol that associates a delay with every sent action, followed by an acknowledgement or error. Input is suffixed with "*?*" and output with "*!*". Discrete probability distributions are denoted with a dotted arc, together with the action label and corresponding probabilities. Markovian actions are presented as staggered arrows.

After the *send?* input is received, there is an expected delay indicated by the Markovian action λ. The delay is exponentially distributed, thus, the probability to go from s_1 to s_2 in T time units is $1 - e^{-\lambda T}$. In state s_2 there is one outgoing discrete probability distribution. The specification in Fig. 1a implies that only 10% of all messages should end in an error report and the remaining 90% get delivered correctly. After a message is delivered, the automaton goes back to its initial state where it stays quiescent until input is provided. This is denoted with the δ self-loop, marking the desired absence of outputs.

2.2 Paths and Traces

Let $\mathcal{M} = \langle S, s_0, L, \rightarrow, \rightsquigarrow \rangle$ be an IOMA. We define the usual language theoretic concepts. A *path* π of $\mathcal{M}$ is a (possibly) infinite sequence of the form

$$\pi = s_1\, t_1\, \mu_1\, \alpha_1\, s_2\, t_2\, \mu_2\, \alpha_2 \ldots,$$

where $s_i \in S$, $t_i \in \mathbb{R}_{\geq 0}$, $\mu_i \in \rightarrow \cup\, \mathbb{P}_{s_i}$ and $\alpha_i \in L \cup \mathbb{R}_{\geq 0}$ for $i = 1, 2, \ldots$. We require that each finite path ends in a state. The sequence $s_i\, t_i\, \mu_i\, \alpha_i\, s_{i+1}$ means that $\mathcal{M}$ resided t_i time units in state s_i before moving to s_{i+1} via α_i using the distribution μ_i. The *length* of a finite path, denoted $|\pi|$, is the number of input and output actions occurring on it.

Note that measuring a single time point in continuous time results in probability zero. Hence, it is necessary to talk about time intervals instead of individual time values. An *abstract path* is a path, where each occurrence of single time values t_i is replaced by intervals $I_i \subseteq \mathbb{R}_{\geq 0}$. However, we limit our interested to intervals of the form $[0, t]$ with $t \in \mathbb{R}_{\geq 0}$. Consequently, any path can be replaced with its abstract path by changing t_i to $[0, t_i]$ or vice versa. This convention lets us use both notions interchangeably.

The *trace* of a path $tr\,(\pi)$ only records its visible behaviour, i.e. time and input/output actions. It is given by the (possibly) infinite sequence of the form

$$\sigma = tr\,(\pi) = t_1\, a_1\, t_2\, a_2 \ldots,$$

where $t_i \in \mathbb{R}_{\geq 0}$ and $a_i \in L_i \cup L_o$ for $i = 1, 2, \ldots$. The length of a trace is the length of its corresponding paths. Note that a path fragment $s_1\, t_1\, \mu_1\, \lambda\, s_2\, t_2\, \mu_2\, a\, s_3$ collapses to $(t_1 + t_2)\, a$ if λ is a Markovian action. Technically, Markovian actions are just parameters for an exponential delay and therefore invisible. Similar to abstract paths, an *abstract trace* is given, if all $t_i \in \mathbb{R}_{\geq 0}$ of a trace are replaced by intervals $I_i \subseteq \mathbb{R}_{\geq 0}$. Again, we limit ourselves to abstract traces only using intervals of the form $[0, t]$ with $t \in \mathbb{R}_{\geq 0}$. This enables us to use traces and abstract traces interchangeably.

We denote the set finite paths $Paths^*\,(\mathcal{M})$ ($Traces^*\,(\mathcal{M})$ resp.) and abstract paths as $AbsPaths^*\,(\mathcal{M})$ ($AbsTraces^*\,(\mathcal{M})$ resp.) and omit the asterisk to include the infinite case. We use $ctraces\,(\mathcal{M})$ to denote the set of traces ending in a deadlock state. Lastly, let the operator $act\,(\pi)$ return the *action path* of π by removing all time values t_i and distributions μ_i. For traces $act\,(\sigma)$ returns visible actions only.

2.3 Traces and Their Probabilities

Similar to how the visible behaviour of a labelled transition system (LTS) is given by its traces, the visible behaviour of an IOMA is given by its trace distributions. A trace distribution is a probability space, that assigns probabilities to all traces. A trace of an LTS is obtained by removing all states and internal actions from a given path. We do the same in the IOMA case: First we resolve all nondeterministic choices via an adversary and then remove all invisible information. The resolution of nondeterministic behaviour leads to a purely probabilistic structure.

The mathematical framework for infinite abstract paths is technically more involved, but completely standard [43]. A classical result in measure theory [11] shows, that it is impossible to assign a probability to *all* sets of traces in nontrivial scenarios. To illustrate: the probability of always rolling a 6 with a die is 0, but the probability of rolling a 6 within the first 100 tries is positive. To resolve this, we use a cone construction of sets of traces.

Adversaries and Path Probability. Similar to [40,43], adversaries form the core concept of our framework. Given any finite piece of history leading to the current state, an adversary returns a distribution over the available transitions.

Definition 2. *An* adversary A *of an IOMA* $\mathcal{M} = \langle S, s_0, L, \to, \rightsquigarrow \rangle$ *is a function*

$$A : Paths^*(\mathcal{M}) \longrightarrow Distr\left(Distr\left(L \times S\right) \cup \{\bot\}\right),$$

such that for each finite path π *only available distributions are scheduled, i.e.*

$$\forall \pi \in Paths^*(\mathcal{M}) : A(\pi)(\mu) > 0, \text{ then } (last(\pi), \mu) \in \to .$$

The value $A(\pi)(\bot)$ *is the probability to* interrupt/halt *the process. An adversary* A halts *on path* π*, if* $A(\pi)(\bot) = 1$*. We say an adversary is of* length $k \in \mathbb{N}$*, if it halts for all paths* π *with length greater or equal to* k*. We denote this set by* $adv(\mathcal{M}, k)$ *and the set of all adversaries by* $adv(\mathcal{M})$ *respectively.*

An adversary resolves all nondeterministic choices of an IOMA making it possible to calculate the probability for each path via the probabilistic execution function. Probabilistic executions assign the unique starting state probability 1 and each following transition either multiplies the probability that the scheduler assigned to an action or, if no action was scheduled, the probability of a Markovian action taking place in a certain time interval.

Definition 3. *Let* A *be an adversary of an IOMA* $\mathcal{M}$*, then we define the* probabilistic execution function $P_A : AbsPaths(\mathcal{M}) \to [0,1]$ *inductively by* $P_A(s_0) = 1$ *and*

$$P_A(\Pi \cdot I\alpha\mu s) = P_A(\Pi) \cdot \begin{cases} A(\pi)(\mu) \cdot \mu(\alpha, s) & \text{if } \alpha \in L \\ \int_I \mathbf{R}(last(\Pi), s)\, e^{-\mathbf{E}(last(\Pi))t} dt & \text{if } \alpha \in \mathbb{R}_{\geq 0} \end{cases},$$

where $I = [0, T] \subseteq \mathbb{R}_{\geq 0}$ *and* π *is the corresponding path to the abstract path* Π*.*

The probability space of an adversary is constructed based on *cones* of paths [40]. The cone C_π of a path π contains all paths that have π as prefix. Given $A \in adv(\mathcal{M})$, let $\Omega_A := Paths(\mathcal{M})$ be the sample set and $\mathcal{F}_A$ be the smallest σ-field generated by the set of cones $\{C_\Pi \subseteq Paths(\mathcal{M}) \mid \Pi \in AbsPaths^*(\mathcal{M})\}$. Standard measure theory arguments [11] ensure that P_A induces a unique probability measure on the measurable space $(\Omega_A, \mathcal{F}_A)$. Hence, an adversary induces a probability space $(\Omega_A, \mathcal{F}_A, P_A)$ on a Markov automaton.

Trace Distributions. A trace distribution is obtained from (the probability space of) an adversary, in the way a trace is obtained from a path; all invisible behaviour is removed. Intuitively, the probability assigned to a set of abstract traces X, is defined as the probability assigned to all abstract paths whose abstract trace is an element of X.

Definition 4. *The trace distribution D of an adversary A is the probability space $(\Omega_D, \mathcal{F}_D, Pr_D)$ given by $\Omega_D = Traces(\mathcal{M})$, $\mathcal{F}_D$ as the smallest σ-field generated by the set of cones $\{C_\sigma \subseteq Traces(\mathcal{M}) \mid \sigma \in AbsTraces^*(\mathcal{M})\}$ and P_D as the unique probability measure on $\mathcal{F}_D$, such that $P_D(X) = P_A\left(tr^{-1}(X)\right)$ for $X \in \mathcal{F}_D$*

A trace distribution is of length $k \in \mathbb{N}$, if it based on an adversary of length k. We denote the set of all such trace distributions by $Trd(\mathcal{M}, k)$. The set of all trace distributions is denoted by $Trd(\mathcal{M})$. This naturally induces an equivalence relation, denoted $=_{TD}$, that equates two IOMAs, if they have the same set of trace distributions.

3 Testing with Markov Automata

Model-based testing entails automatic test case generation, execution, and evaluation. We formalize the notion of offline tests and show how they can be generated in batch or on-the-fly. The functional correctness of a system under test (SUT) is assessed upon test execution. To evaluate the probabilistic correctness of the system, tests are executed multiple times and recorded in a sample. The trace frequencies observed in a sample are then compared to their expectations. Consequently, an implementation is deemed correct, if these frequencies are within certain confidence intervals given by the requirements.

3.1 Test Generation

Test Cases. We consider test cases as sets of traces based on an action signature consisting of inputs and outputs (L_i, L_o). These traces describe the possible behaviour of a tester. In each state of a test, a tester may decide to stimulate the SUT, observe its possible outputs or stop the process altogether.

Mathematically, we consider test cases as input-reactive and output-generative probabilistic automata, i.e. Markov automata with $\rightsquigarrow = \emptyset$. This enables us to model the choices of stimulating, observing or stopping probabilistically. Note that, even in the non-probabilistic case, the test cases are often

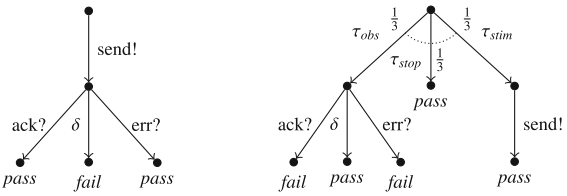
created probabilistically in practice. However, this is rarely ever supported in theory. Thus, our definition fills a small gap here.

Definition 5. *A* test *over an action signature* (L_i, L_o) *is an IOMA of the form* $t = (S, s_0, L_o \setminus \{\delta\}, L_i \cup \{\delta\}, \{\tau_{stop}, \tau_{stim}, \tau_{obs}\}, \rightarrow, \emptyset)$*, such that*

- *t is internally deterministic and does not contain an infinite path;*
- *t is acyclic and connected;*
- *For every state* $s \in S$*, either*
  - *enabled* $\{s\} = \emptyset$
  - *enabled* $\{s\} = \{\tau_{stop}, \tau_{stim}, \tau_{obs}\}$
  - *enabled* $\{s\} = L_i \cup \{\delta\}$
  - *enabled* $\{s\} \subseteq L_o \setminus \{\delta\}$

A test for a specification IOMA $\mathcal{M} = \langle S, s_0, L, \rightarrow, \rightsquigarrow \rangle$ *is a test over its action signature.*

Note that the action signature of tests has switched input and output labels. This is to allow for synchronisation in a parallel composition with an implementation IOMA.



(a) Nonprobabilistic Test (b) Probabilistic Test

Fig. 2. A regular test and a probabilistic test derived for the specification of Fig. 1.

Example 2. Fig. 2 shows two test cases for the specification IOMA in Fig. 1. The probabilistic test case models the possible behaviour of a tester. Here, a probabilistic choice is made with $\frac{1}{3}$ on whether to stop, stimulate or wait for responses of the system. Traces in the tests are labelled *pass* or *fail* according to Definition 6.

Annotations. To state whether observed functional behaviour is deemed correct, each trace of a test is annotated with a verdict; *pass* for correct and *fail* for erroneous behaviour. The classical **ioco** test case annotation [47] suffices here. Informally, all traces of a test, that are also present in the specification, get annotated as correct.

Definition 6. *For a test t, a* test annotation *is a function* $a : ctraces(t) \longrightarrow \{pass, fail\}$*. A pair* $\hat{t} = (t, a)$ *consisting of a test and an annotation is called an* annotated test*. If t is a test for a specification* $\mathcal{S}$ *we define the annotation* $a_{\mathcal{S},t} : ctraces(t) \longrightarrow \{pass, fail\}$ *by*

$$a_{\mathcal{S},t}(\sigma) = \begin{cases} \mathit{fail} & \mathit{if}\ \exists\, \varrho \in \mathit{Traces}(\mathcal{S}), a! \in L_O^\delta : \varrho a! \sqsubseteq \sigma \wedge \varrho a! \notin \mathit{Traces}(\mathcal{S}); \\ \mathit{pass} & \mathit{otherwise}, \end{cases}$$

where $\sqsubseteq$ *denotes the prefix relation for traces.*

Algorithms. Algorithm 1 presents the batch test generation according to Definition 5. The inputs are a specification IOMA $\mathcal{M}$ and a history trace, which is initially empty. At each step of the computation, the algorithm decides probabilistically to stop with $p_{\sigma,1}$, stimulate with $p_{\sigma,2}$ or observe with $p_{\sigma,3}$. The latter two choices recursively call the batch-gen algorithm again with updated trace history. Note that $p_{\sigma,1} + p_{\sigma,2} + p_{\sigma,3} = 1$.

Algorithm 2 describes the on-the-fly test case derivation for a given specification $\mathcal{S}$, implementation $\mathcal{I}$ and upper limit for the test length n. It returns a verdict within the first n steps. The verdict is *fail* if unexpected output was encountered and *pass* otherwise. With probability $p_{\sigma,1}$ the algorithm observes the output of the implementation and with probability $p_{\sigma,2}$ it stimulates it with a new input. Note that $p_{\sigma,1} + p_{\sigma,2} = 1$.

Algorithm 1: Batch test generation for *Mar-ioco*.

Input: Specification IOMA $\mathcal{S}$ and history $\sigma \in \mathit{traces}(\mathcal{S})$.
Output: A test case t for $\mathcal{S}$.

Procedure batch($\mathcal{S}, \sigma$)
  $p_{\sigma,1}\cdot$[true] $\rightarrow$
  return $\{\tau_{stop}\}$
  $p_{\sigma,2}\cdot$[true] $\rightarrow$
  result := $\{\tau_{obs}\}$
  forall $b! \in L_o$ **do:**
    if $\sigma b! \in \mathit{traces}(\mathcal{S})$ **:**
      result := result $\cup \{b!\sigma' \mid \sigma' \in \text{batch}(\mathcal{S}, \sigma b!)\}$
    else:
      result := result $\cup \{b!\}$
    end
  end
  return result
  $p_{\sigma,3}\cdot[\sigma a? \in \mathit{traces}(\mathcal{S})] \rightarrow$
  result := $\{\tau_{stim}\} \cup \{a?\sigma' \mid \sigma' \in \text{batch}(\mathcal{S}, \sigma a?)\}$
  forall $b! \in L_O$ **do:**
    if $\sigma b! \in \mathit{traces}(\mathcal{S})$ **:**
      result := result $\cup \{b!\sigma' \mid \sigma' \in \text{batch}(\mathcal{S}, \sigma b!)\}$
    else:
      result := result $\cup \{b!\}$
    end
  end
  return result

Algorithm 2: On-the-fly test case derivation for *Mar-ioco*.

Input: Specification IOMA $\mathcal{S}$, an implementation $\mathcal{I}$ and an upper bound for the test length $n \in \mathbb{N}$.
Output: Verdict pass if Impl. was ioco conform in the first n steps and fail if not.

$\sigma := \epsilon$
while $|\sigma| < n$ **do:**
  $p_{\sigma,1}\cdot$[true] $\rightarrow$
  observe next output $b!$ *(possibly* δ*) of* $\mathcal{I}$
  $\sigma := \sigma b!$
  if $\sigma \notin \mathit{traces}(\mathcal{S})$ **:**
    return fail
  $p_{\sigma,2} \cdot [\sigma a? \in \mathit{traces}(\mathcal{S})] \rightarrow$
  try:
    atomic
      stimulate I with $a?$
      $\sigma := \sigma a?$
    end
  catch *an output* $b!$ *occurs before* $a?$ *could be applied*
    $\sigma := \sigma b!$
    if $\sigma \notin \mathit{traces}(\mathcal{S})$ **:**
      return fail
  end
end
return pass

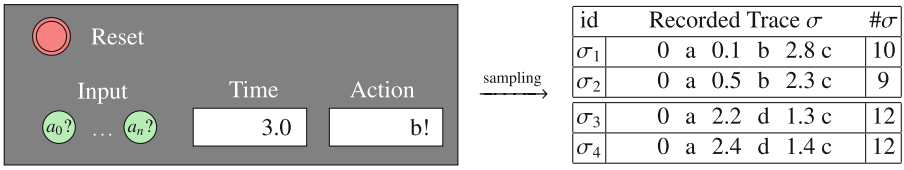
Theorem 7. *All test cases generated by Algorithm 1 are test cases according to Definition 5. All test cases generated by Algorithm 2 assign the correct functional verdict according to Definition 6.*

3.2 Test Execution

Since discrete probabilistic choices and stochastic time delay are integral parts of Markov automata, there is a twofold evaluation process of functional and statistical behaviour. While functional behaviour is assessed via the test annotation as in classic **ioco**-test theory [50], we focus on describing the sampling process to validate statistical correctness.

Sampling. In order to reason about probabilistic correctness, a single test execution is insufficient. Rather, we collect a sample via multiple test runs. The sampling process consists of a push-button experiment in the sense of [34]. Assume a black-box timed trace machine is given with inputs, time and action windows, and a reset button as illustrated in Fig. 3.

At the beginning of the experiment, we set the parameters for sample length $k \in \mathbb{N}$, sample width $m \in \mathbb{N}$ and a level of significance $\alpha \in (0, 1)$. That is, we choose the length of individual runs, how many runs should be observed and a limit for the statistical *error of first kind*, i.e. the probability of rejecting a correct implementation.



id	Recorded Trace σ	$\#\sigma$
σ_1	0 a 0.1 b 2.8 c	10
σ_2	0 a 0.5 b 2.3 c	9
σ_3	0 a 2.2 d 1.3 c	12
σ_4	0 a 2.4 d 1.4 c	12

Fig. 3. Black box timed trace machine with input alphabet $a_0?, \ldots, a_n?$, reset button, and time and action windows. Running the machine m times and observing traces of length k yields a sample. The ID together with the trace and the respective number of occurrences are noted down.

We assume that the timer resets to 0 after every visible action and that two consecutive occurrences of the same action are distinguishable. An external observer records each individual execution before the reset button is pressed and the machine starts again. Thus, we collect m traces of length k, which are summarized as a *sample* O.

During each run the black-box $\mathcal{I}$ is governed by a trace distribution $D \in \mathit{Trd}(\mathcal{I})$. In order for any statistical reasoning to work, we assume that D is the same in every run. Thus, the SUT chooses a trace distribution D and D chooses a trace σ to execute.

Frequencies and expectations. We evaluate the deviation of a collected sample to the expected distribution. The latter is given for any underlying trace distribution D of the specification IOMA. Since the trace distribution is assumed

to be the same for all runs, the expected probability to observe a trace σ is given by $\mathbb{E}^D(\sigma) = P_D(\sigma)$.

Depending on the accuracy of time measurement, it is unlikely to record the *exact* same timed trace more than once. Therefore, we group traces in classes based on the same visible action behaviour. For a given abstract trace σ, its class Σ_σ is the set of all abstract traces $\varrho \in O$, such that $act(\sigma) = act(\varrho)$. A sample of length k and width m then induces a frequency measure, given by

$$freq(O)(\sigma) = \frac{|\Sigma_\sigma|}{m} \Pi_{i=1}^{k} \frac{|\{\varrho \in \Sigma_\sigma \mid I_i^\varrho \subseteq I_i^\sigma\}|}{|\Sigma_\sigma|},$$

where I_i^ϱ denotes the i-th time interval of trace ϱ, for any abstract trace σ. The implementation is rejected for statistical reasons, should the deviation of the measure $freq(O)$ to $\mathbb{E}^D$ exceed a certain threshold based on α.

Acceptable outcomes. Conversely, we accept a sample O if $freq(O)$ lies within some distance, say r_α, of the expected distribution $\mathbb{E}^D$. Recall the definition of a closed ball centred at $x \in X$ with radius r as $B_r(x) = \{y \in X \mid dist(x,y) \leq r\}$. All measures deviating at most by r from the expected distribution are contained within the ball $B_r(\mathbb{E}^{\boldsymbol{D}})$, where $dist(u,v) := \sup_{\sigma \in (\mathbb{R}_{\geq 0} \times L)^k} \mid u(\sigma) - v(\sigma) \mid$ is the total variation distance of measures.

To limit the error of accepting an erroneous sample, we choose the smallest radius, such that the error of rejecting a correct sample is not greater than the a priori chosen level of significance $\alpha \in (0,1)$ by [2]

$$r_\alpha := \inf\left\{r \in \mathbb{R}_{>0} \mid P_{\boldsymbol{D}}\left(freq^{-1}\left(B_r\left(\mathbb{E}^{\boldsymbol{D}}\right)\right)\right) > 1 - \alpha\right\}.$$

Definition 8. *For $k, m \in \mathbb{N}$ and an IOMA $\mathcal{M}$ the* acceptable outcomes *under a trace distribution $D \in Trd(\mathcal{M}, k)$ of significance level $\alpha \in (0,1)$ are given by the set*

$$Obs(D, \alpha, k, m) = \left\{O \in \left((\mathbb{R}_{\geq 0} \times L)^k\right)^m \mid dist\left(freq(O), \mathbb{E}^D\right) \leq r_\alpha\right\}.$$

The set of observations *of $\mathcal{M}$ of significance level $\alpha \in (0,1)$ is given by*

$$Obs(\mathcal{M}, \alpha, k, m) = \bigcup_{D \in Trd(\mathcal{M},k)} Obs(D, \alpha, k, m).$$

The set of observations therefore guarantees two properties, reflecting the error of false rejection and false acceptance respectively:

1. If a sample O was truthfully generated by $\mathcal{M}$ or a behaviourally equivalent IOMA, then there is a trace distribution D such that $P_D(O) \geq 1 - \alpha$; and
2. if a sample O was generated by a behaviourally different MA, then for all trace distributions D' we have $P_{D'}(O) \leq \beta_m$,

[2] Note that $freq(O)$ is not a bijection, but used here for ease of notation.

where α is the predefined level of significance and β_m is unknown but minimal by construction. Note that $\beta_m \to 0$ as $m \to \infty$, thus the error of falsely accepting an erroneous sample decreases with increasing sample width. Here, behavioural equivalence is induced by trace distribution equivalence, cf. Definition 4.

Goodness of fit. In order to state whether a given sample O is a truthful observation of $\mathcal{M}$, we need to find a trace distribution $D \in \mathit{Trd}(\mathcal{M})$ such that $O \in \mathit{Obs}(D, m, k, \alpha)$. It guarantees that the error of rejecting a truthful sample is at most α. These sets are crucial for the soundness and completeness proofs. However, they are computationally intractable to gauge for every D, since there are uncountably many.

Instead, we use χ^2 hypothesis testing to assure that a sample is acceptable. The χ^2 score is calculated as:

$$\chi^2 = \sum_{i=1}^{l} \frac{\left(n\left(\Sigma_{\sigma_i}\right) - m\mathbb{E}^D\left(\Sigma_{\sigma_i}\right)\right)^2}{m\mathbb{E}^D\left(\Sigma_{\sigma_i}\right)} \quad \text{with} \quad l \leq m. \tag{1}$$

To find a trace distribution that gives a high likelihood to an observed sample, we need to find D, such that $\chi^2 < \chi^2_{crit}$. The critical value depends on α and the degrees of freedom in the statistical test. In this case the degrees of freedom are given by the number of trace classes minus one, i.e. the probability of one class is determined, if we know all others. The critical value for χ^2 tests can be calculated or universally looked up in a table.

By construction of adversaries, cf. Definition 2, we are interested in the resolution of the nondeterministic choices. Consequently, (1) turns into a satisfaction problem over a probability vector p in a rational function of two polynomials f and g as $f(p)/g(p)$. As [36] shows, optimization over rational functions and inequality constraints is **NP-hard**.

Since (1) neglects time stamps, we need to assure that the recorded time intervals correspond to the α confidence intervals of specified Markovian actions. That is

$$\forall \lambda \in \mathbb{R}_{\geq 0} \text{ with } (s, \lambda, s') \in \rightsquigarrow \text{ for } s, s' \in S : \lambda \in \left[\frac{2\sum_{i=1}^{n} t_i}{\chi^2_{2n}(1-\alpha/2)}, \frac{2\sum_{i=1}^{n} t_i}{\chi^2_{2n}(\alpha/2)}\right],$$

The confidence intervals depend on a scheduler that solves the satisfaction problem.

3.3 Test Evaluation and Verdicts

An implementation should pass the test suite, if it passes the two verdicts for functional behaviour and probabilistic behaviour. This is reflected in the mathematical verdicts.

Definition 9. *Given a specification* $\mathcal{S} = \langle S, s_0, L, \rightarrow, \rightsquigarrow \rangle$*, an annotated test* $\hat{t}$ *for* $\mathcal{S}$*,* $k, m \in \mathbb{N}$ *where* k *is given by the trace length of* $\hat{t}$ *and a level of significance* $\alpha \in (0, 1)$*, we define the* functional verdict *as the function* $v_{func} : \mathit{IOMA} \longrightarrow \{\mathit{pass}, \mathit{fail}\}$*, with*

$$v_{func}(\mathcal{I}) = \begin{cases} pass & \text{if } \forall \sigma \in ctraces(\mathcal{I} \,||\, t) \cap ctraces(t) : a(\sigma) = pass \\ fail & \text{otherwise,} \end{cases}$$

and the probabilistic verdict *as the function* $v_{prob} : IOMA \longrightarrow \{pass, fail\}$*, with*

$$v_{prob}(\mathcal{I}) = \begin{cases} pass & \text{if } \exists D \in Trd(\mathcal{S}, k) : P_D(Obs(\mathcal{I} \,||\, t, \alpha, k, m)) \geq 1 - \alpha \\ fail & \text{otherwise,} \end{cases}$$

where $||$ *denotes the parallel composition. The overall verdict is* pass*, iff an implementation passes both verdicts.*

A note on quiescence. A test case needs to assess if an SUT is allowed to be unresponsive when output was expected [45]. Quiescence δ models the absence of output for indefinite time. Therefore, it should be regarded with caution in practical test scenarios. Earlier work assumes a global fixed time-out value set by a user [6].

Time progress of Markov automata is exponentially delayed, hence, a global time-out value has two disadvantages: 1. a time-out might occur, before a specified Markovian action takes place and 2. a *global* time-out value might unnecessarily prolong the test process. Therefore, our interest is to minimize the probability of erroneously declaring quiescence, while keeping the overall testing time as low as possible.

Assume a level of significance $\alpha \in (0, 1)$ is given. Let λ be the exit rate of a state s. Then the exit rate of s is a random variable T that is exponentially distributed with parameter λ. The probability, that a Markovian action is executed before a state-specific maximum waiting time t_{max} expires should be greater than $(1 - \alpha)$, i.e.

$$P(T < t_{max}) > 1 - \alpha$$

Hence, choosing $t_{max} > -\frac{\log \alpha}{\lambda}$ minimizes the probability of assigning quiescence, when the SUT makes progress. Since the sum of exponential distributions is not exponentially distributed, we resort to less sharper bounds for consecutive Markovian transitions.

Example 3. Fig. 4 shows a simplistic specification of a file exchange protocol. An exponential distribution is used to model the time delay between sending a file and acknowledging its reception. Note that different expected delays are associated with sending a small or a large file respectively.

After a file was send, there is a chance that it gets lost and we do not receive the *acknowledge!* output. In this case the system is judged as quiescent, and therefore faulty. However, since $\nu \ll \lambda$ a test should wait at least $-\frac{\log \alpha}{10}$ time units in s_1 and $-\log \alpha$ in s_2, to minimize the probability to erroneously judge the system as quiescent, while also keeping the testing time as low as possible.

Regardless, for a sufficiently large sample size, an MBT-tool eventually erroneously observes quiescence. The right hand side of Fig. 4 therefore allows for some amount of quiescence observations depending on α.

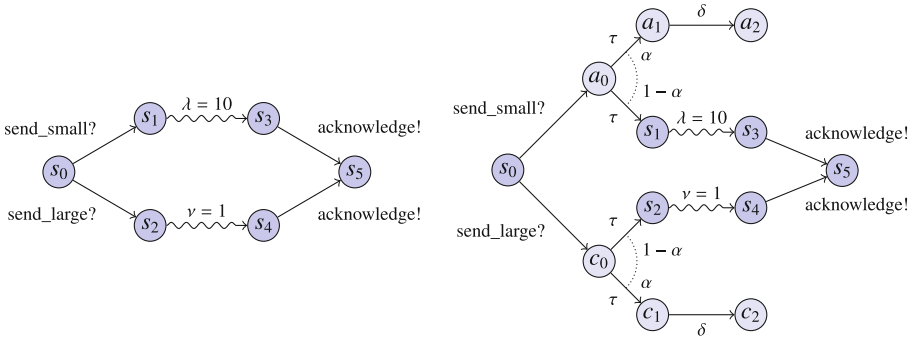


Fig. 4. Specification of a file exchange protocol. Sending a small file is expected to take less time. The right hand side models the possibility to erroneously declare quiescence probabilistically.

4 Conformance, Soundness and Completeness

A fundamental result of our work is the correctness of our framework, phrased as soundness and completeness. *Soundness* ensures that test cases assign the correct verdict. *Completeness* postulates that the framework is powerful enough to discover each deviation from the specification. In order to formulate these properties, we require a formal notion of conformance that we define as the **Mar-ioco** relation [18].

4.1 The Mar-Ioco Relation

The **ioco** relation as defined in [50] states, that an implementation conforms to a specification, if it never provides any unspecified output or quiescence. Mathematically, for two input-output transition systems $\mathcal{I}$ and $\mathcal{S}$, with $\mathcal{I}$ *input enabled*, we say $\mathcal{I} \sqsubseteq_{ioco} \mathcal{S}$, iff

$$\forall \sigma \in \mathit{Traces}\,(\mathcal{S}) : \mathit{out}_{\mathcal{I}}\,(\sigma) \subseteq \mathit{out}_{\mathcal{S}}\,(\sigma)\,.$$

This restricts the theory to functional behaviour in the case of classic transition systems. To generalize **ioco** to Markov automata, we need two auxiliary concepts:

Trace Distribution Prefix. Given a trace distribution D of length k and a trace distribution D' of length greater or equal than k, we say D is a *prefix* of D', written $D \sqsubseteq_k D'$, if both assign the same probability to all traces of length k.

Output Continuation. Given a trace distribution D of length k, its *output continuation* is the set of trace distributions of length $k+1$ such that $D \sqsubseteq_k D'$, assigning probability zero to traces of length $k+1$ ending in inputs. This set is denoted by $\mathit{outcont}_{\mathcal{M}}\,(D)$.

We are now able to define the conformance relation **Mar-ioco**. Intuitively, an implementation is conforming, if the probability of every output trace can be matched by the specification. This includes the three factors: 1. functional behaviour, 2. probabilistic behaviour and 3. stochastic timing.

Definition 10. *Let $\mathcal{I}$ and $\mathcal{S}$ be IOMA with $\mathcal{I}$ input-enabled. We write $\mathcal{I} \sqsubseteq_{Mar-ioco} \mathcal{S}$, if for all $k \in \mathbb{N}$*

$$\forall D \in Trd(\mathcal{S}, k) : outcont_{\mathcal{I}}(D) \subseteq outcont_{\mathcal{S}}(D).$$

The **Mar-ioco** relation conservatively extends the **ioco** relation to Markov automata. That is, both relations coincide for classic input output transition systems (IOTSs).

Theorem 11. *For two IOTSs $\mathcal{I}, \mathcal{S}$ with $\mathcal{I}$ input enabled, we have*

$$\mathcal{I} \sqsubseteq_{ioco} \mathcal{S} \iff \mathcal{I} \sqsubseteq_{Mar-ioco} \mathcal{S}.$$

In **ioco** theory, the implementation is always assumed to be input enabled, to model that a tester can give any input at any moment. If the specification is input enabled too, **ioco** coincides with trace inclusion [50]. Assuming an input enabled specification, our results show, that **Mar-ioco** coincides with trace distribution inclusion. Moreover, the relation is transitive, just like **ioco** [50].

Theorem 12. *Let $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$ be IOMAs and let $\mathcal{A}$ and $\mathcal{B}$ be input enabled, then*

- *$\mathcal{A} \sqsubseteq_{Mar-ioco} \mathcal{B}$ if and only if $\mathcal{A} \sqsubseteq_{TD} \mathcal{B}$.*
- *$\mathcal{A} \sqsubseteq_{Mar-ioco} \mathcal{B}$ and $\mathcal{B} \sqsubseteq_{Mar-ioco} \mathcal{C}$ imply $\mathcal{A} \sqsubseteq_{Mar-ioco} \mathcal{C}$.*

4.2 Soundness and Completeness

Since the underlying model is probabilistic, there remains a degree of uncertainty known as the *errors of first and second kind*. For MBT of probabilistic systems this translates to the likelihood to reject a correct implementation, and to accept an erroneous one respectively. Hence, a test suite can only be considered sound and complete with a guaranteed (high) probability.

Soundness expresses for a given $\alpha \in (0, 1)$, that there is a $(1 - \alpha)$ probability, that a correct system passes the test suite for sufficiently large sample width m.

Theorem 13. *Each annotated test for an IOMA $\mathcal{S}$ is sound for every level of significance $\alpha \in (0, 1)$ with respect to **Mar-ioco**.*

Completeness of a test suite is inherently a theoretical result. Possible loops and infinite behaviour in the SUT require a test suite of infinite size. Further, there is the chance of accepting an erroneous implementation, i.e. the error of second kind. However, the latter is bound from above and decreases with larger sample size.

Theorem 14. *The set of all annotated tests for an IOMA $\mathcal{S}$ is complete for every level of significance $\alpha \in (0, 1)$ with respect to **Mar-ioco**.*

5 Experiments on the Bluetooth Device Discovery Protocol

Bluetooth is a wireless communication technology standard [41] specifically aimed at low-powered devices that communicate over short distances. To cope with inference, the protocol uses a frequency hopping scheme in its initialisation period. Before any communication can take place, Bluetooth devices organise themselves into small networks called *piconets* consisting of one *master* and up to seven *slave* devices.

To illustrate our framework, we study the discovery phase for one master and one slave device. We give a high level overview of the protocol in this case. The reader is referred to [15] for a detailed description on the protocol in a more general setting.

To resolve possible interference, the master and slave device communicate on a previously agreed sequence of 32 frequencies. Both devices have a 28-bit clock that ticks every $312.5\mu s$. Every two consecutive ticks, the master device sends packages on two frequencies, followed by a two-tick listening period on the same frequencies. It picks the broadcasting frequency according to the formula:

$$\mathit{freq} = [CLK_{16-12} + \mathit{off} + (CLK_{4-2,0} - CLK_{16-12}) \mod 16] \mod 32,$$

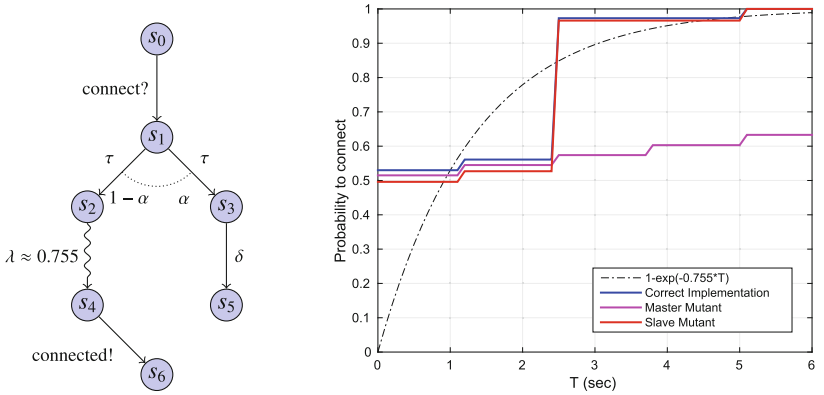
where CLK_{i-j} marks the bits $i, \ldots, j$ of the clock and $\mathit{off} \in \mathbb{N}$ is an offset. The master device chooses one of two tracks and switches to the other every $2.56s$. Moreover, every $1.28s$, i.e. every time the 12th bit of the clock changes, a frequency is swapped between the two tracks. For simplicity, we chose $\mathit{off} = 1$ for track one and $\mathit{off} = 17$ for track two, such that the two tracks initially comprise frequencies $1, \ldots, 16$ and $17, \ldots, 32$.

Conversely, the slave device periodically scans on the 32 frequencies and is either in a sleeping or listening state. To ensure the eventual connection, the hopping rate of the slave device is much slower. Every $0.64s$ it listens to one frequency in a window of $11.25ms$ and sleeps during the remaining time. It cycles to the next frequency after $1.28s$. This is enough for the master device to broadcast on 16 different frequencies.

We implemented the protocol and two mutants in Java 7; 1. the *master mutant* never switches between tracks one and two, therefore covering far less different frequencies than the correct protocol in the same time and 2. the *slave mutant* only listens for $5.65ms$ every $1.28s$ and therefore has a much longer sleeping period.

Since the time to connect two devices is deterministic for any initial state, we assumed that the clocks are desynchronized, i.e. the master sends out packages, while the slave starts listening after a uniformly chosen random waiting time. The expected waiting time $1/\lambda$ for an established connection is therefore estimated as $1.325s$, i.e. $\lambda \approx 0.755$.

Figure 5a shows the high level specification of the protocol. The request for both devices to synchronise is either followed by an acknowledgement or a time-out. Note that the amount of allowed time-outs is part of the specification and depends on α. A collected sample therefore consisted of the traces



(a) Protocol Specification IOMA (b) Probability to establish connection over time

Fig. 5. High level specification of the Bluetooth device discovery protocol for one master and one slave device. The time to establish a connection for a correct implementation and two mutants is compared to the assumed underlying exponential distribution with parameter $\lambda \approx 0.755$.

$\sigma_1 = 0\, \mathit{connect}?\, t\, \mathit{connected}!$ and $\sigma_2 = 0\, \mathit{connect}?\, t\, \delta$. Figure 5b shows the cumulative probability distribution to connect within T seconds of the assumed underlying distribution $1 - \mathit{exp}\,(-0.755T)$ and sample data collected for 10^3 runs of the correct implementation and the two mutants.

To mitigate statistical deviations, we collected 10^3 samples of the size 10^3 to calculate the average confidence intervals for $\alpha = 0.05$. The confidence interval of the correct implementation resulted in $[0.721, 0.824]$, containing the assumed value $\lambda = 0.755$ and was therefore judged as correct. The average connection time of the master mutant was $30.2s$ with a confidence interval of $[0.030, 0.034]$ and was therefore rejected. Dividing the listening time of the slave mutant into half had a less significant impact and gave a confidence interval of $[0.781, 0.887]$. It was consequently rejected with a small margin.

6 Conclusions and Future Work

We presented a sound and complete framework to test probabilistic systems with sotchastic-time delays based on a model. We defined a conformance relation in the **ioco** tradition called **Mar-ioco** pinning down precisely what *correctness* means. Our algorithms provide test cases that are sound with respect to this notion. Probabilistic correctness is assessed after a sampling process that counts frequencies of traces and compares them to statistical requirements.

Future work should comprise the practical aspects of our work: more powerful statistical methods facilitating efficient tool support. Lastly, we plan to apply our framework to a case study of larger size.

Acknowledgements. We would like to thank David Huistra for his aid on the case study.

References

1. Beyer, M., Dulz, W.: Scenario-based statistical testing of quality of service requirements. In: Leue, S., Systä, T.J. (eds.) Scenarios: Models, Transformations and Tools. LNCS, vol. 3466, pp. 152–173. Springer, Heidelberg (2005). doi:10.1007/11495628_9
2. Bohnenkamp, H., Belinfante, A.: Timed testing with TorX. In: Fitzgerald, J., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 173–188. Springer, Heidelberg (2005). doi:10.1007/11526841_13
3. Bohnenkamp, H., Stoelinga, M.: Quantitative testing. In: Proceedings of the 8th International Conference on Embedded Software, (EMSOFT), pp. 227–236. ACM (2008)
4. Böhr, F.: Model-based statistical testing of embedded systems. In: IEEE 4th International Conference on Software Testing, Verification and Validation, pp. 18–25 (2011)
5. Bozga, M., David, A. Hartmanns, H., Hermanns, H., Larsen, K.G., Legay, A., Tretmans, J.: State-of-the-art tools and techniques for quantitative modelling and analysis of embedded systems. In: DATE, pp. 370–375 (2012)
6. Briones, L.B., Brinksma, E.: A test generation framework for *quiescent* real-time systems. In: Grabowski, J., Nielsen, B. (eds.) FATES 2004. LNCS, vol. 3395, pp. 64–78. Springer, Heidelberg (2005). doi:10.1007/978-3-540-31848-4_5
7. Briones, L.B., Brinksma, E., Stoelinga, M.: A semantic framework for test coverage. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 399–414. Springer, Heidelberg (2006). doi:10.1007/11901914_30
8. Cheung, L., Stoelinga, M., Vaandrager, F.: A testing scenario for probabilistic processes. J. ACM **54**(6), 29:1–29:45 (2007). Article 29
9. Choi, S.G., Dachman-Soled, D., Malkin, T., Wee, H.: Improved non-committing encryption with applications to adaptively secure protocols. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 287–302. Springer, Heidelberg (2009). doi:10.1007/978-3-642-10366-7_17
10. Cleaveland, R., Dayar, Z., Smolka, S.A., Yuen, S.: Testing preorders for probabilistic processes. Inf. Comput. **154**(2), 93–148 (1999)
11. Cohn, D.L.: Measure Theory. Birkhäuser, Basel (1980)
12. D'Argenio, P.R., Katoen, J.-P.: A theory of stochastic systems part I: stochastic automata. Inf. Comput. **203**(1), 1–38 (2005)
13. Deng, Y., Hennessy, M.: On the semantics of Markov automata. Inf. Comput. **222**, 139–168 (2013)
14. Deng, Y., Hennessy, M., van Glabbeek, R.J., Morgan, C.: Characterising testing preorders for finite probabilistic processes. CoRR (2008)
15. Duflot, M., Kwiatkowska, M., Norman, G., Parker, D.: A formal analysis of bluetooth device discovery. Int. J. Softw. Tools Technol. Transf. **8**(6), 621–632 (2006)
16. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: IEEE 25th Annual Symposium on LICS, pp. 342–351 (2010)
17. Gerhold, M., Stoelinga, M.: Model-based testing of probabilistic systems. In: Stevens, P., Wąsowski, A. (eds.) FASE 2016. LNCS, vol. 9633, pp. 251–268. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49665-7_15

18. Gerhold, M., Stoelinga, M.: Model-based testing of stochastic systems with IOCO theory. In: A-TEST 2016, Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation, pp. 45–51. ACM (2016)
19. Guiotto, A., Acquaroli, B., Martelli, A.: MaTeLo: automated testing suite for software validation. In: DASIA, vol. 532 (2003)
20. Hartmanns, A., Hermanns, H.: The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 593–598. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54862-8_51
21. Hermanns, H., Chains, I.M.: Interactive Markov Chains: and the Quest for Quantified Quality. Springer, Heidelberg (2002)
22. Hessel, A., Larsen, K.G., Mikucionis, M., Nielsen, B., Pettersson, P., Skou, A.: Testing real-time systems using UPPAAL. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) Formal Methods and Testing. LNCS, vol. 4949, pp. 77–117. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78917-8_3
23. Hierons, R.M., Merayo, M.G.: Mutation testing from probabilistic and stochastic finite state machines. J. Syst. Softw. **82**(11), 1804–1818 (2009)
24. Hierons, R.M., Núñez, M.: Testing probabilistic distributed systems. In: Hatcliff, J., Zucca, E. (eds.) FMOODS/FORTE -2010. LNCS, vol. 6117, pp. 63–77. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13464-7_6
25. Hierons, R.M., Núñez, M.: Implementation relations and probabilistic schedulers in the distributed test architecture. J. Syst. Softw. (2017)
26. Hwang, I., Cavalli, A.R.: Testing a probabilistic FSM using interval estimation. Comput. Netw. **54**(7), 1108–1125 (2010)
27. Jegourel, C., Legay, A., Sedwards, S.: A Platform for High Performance Statistical Model Checking – PLASMA. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 498–503. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28756-5_37
28. Krichen, M., Tripakis, S.: Conformance testing for real-time systems. Form. Methods Syst. Des. **34**(3), 238–304 (2009)
29. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: probabilistic symbolic model checker. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) TOOLS 2002. LNCS, vol. 2324, pp. 200–204. Springer, Heidelberg (2002). doi:10.1007/3-540-46029-2_13
30. Larsen, K.G., Mikucionis, M., Nielsen, B.: Online testing of real-time systems using Uppaal. In: Grabowski, J., Nielsen, B. (eds.) FATES 2004. LNCS, vol. 3395, pp. 79–94. Springer, Heidelberg (2005). doi:10.1007/978-3-540-31848-4_6
31. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing, pp. 344–352. ACM Press (1989)
32. Lohrey, M., D'Argenio, P.R., Hermanns, H.: Axiomatising Divergence. In: Widmayer, P., Eidenbenz, S., Triguero, F., Morales, R., Conejo, R., Hennessy, M. (eds.) ICALP 2002. LNCS, vol. 2380. Springer, Heidelberg (2002)
33. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Wiley, Hoboken (1994)
34. Milner, R.: A Calculus of Communicating Systems. Springer, Heidelberg (1980)
35. Mostowski, W., Poll, E., Schmaltz, J., Tretmans, J., Wichers Schreur, R.: Model-based testing of electronic passports. In: Alpuente, M., Cook, B., Joubert, C. (eds.) FMICS 2009. LNCS, vol. 5825, pp. 207–209. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04570-7_19
36. Nie, J., Demmel, J., Gu, M.: Global minimization of rational functions and the nearest GCDs. J. Glob. Optim. **40**(4), 697–718 (2008)

37. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, Hoboken (2014)
38. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 220–270. Springer, Heidelberg (2007). doi:10.1007/978-3-540-72522-0_6
39. Russell, N., Moore, R.: Explicit modelling of state occupancy in hidden markov models for automatic speech recognition. In: IEEE International Conference on Acoustics, Speech, and Signal Processing ICASSP, vol. 10, pp. 5–8 (1985)
40. Segala, R.: Modeling and verification of randomized distributed real-time systems. Ph.D. thesis, Cambridge, MA, USA (1995)
41. B. SIG. Bluetooth Specification, version 1.2 (2003). www.bluetooth.com
42. Song, L., Zhang, L., Godskesen, J.C., Hermanns, H., Eisentraut, C.: Late weak bisimulation for Markov automata. CoRR, abs/1202.4116 (2012)
43. Stoelinga, M.: Alea jacta est: verification of probabilistic, real-time and parametric systems. Ph.D. thesis, Radboud University of Nijmegen (2002)
44. Stoelinga, M., Vaandrager, F.: Root contention in IEEE 1394. In: Katoen, J.-P. (ed.) ARTS 1999. LNCS, vol. 1601, pp. 53–74. Springer, Heidelberg (1999). doi:10.1007/3-540-48778-6_4
45. Stokkink, W.G.J., Timmer, M., Stoelinga, M.I.A.: Divergent quiescent transition systems. In: Veanes, M., Viganò, L. (eds.) TAP 2013. LNCS, vol. 7942, pp. 214–231. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38916-0_13
46. Thrun, S.: Probabilistic robotics. Commun. ACM **45**(3), 52–57 (2002)
47. Timmer, M., Brinksma, H., Stoelinga, M., Testing, M.-B., Software, I., Safety, S.: Specification and verification, Volume 30 of NATO Science for Peace and Security, pp. 1–32. IOS Press (2011)
48. Timmer, M., Katoen, J.-P., Pol, J., Stoelinga, M.I.A.: Efficient modelling and generation of Markov automata. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 364–379. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32940-1_26
49. Timmer, M., van de Pol, J., Stoelinga, M.I.A.: Confluence reduction for Markov automata. In: Braberman, V., Fribourg, L. (eds.) FORMATS 2013. LNCS, vol. 8053, pp. 243–257. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40229-6_17
50. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Softw. - Concepts Tools **17**(3), 103–120 (1996)
51. Veanes, M., Campbell, C., Grieskamp, W., Schulte, W., Tillmann, N., Nachmanson, L.: Model-based testing of object-oriented reactive systems with spec explorer. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) Formal Methods and Testing. LNCS, vol. 4949, pp. 39–76. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78917-8_2
52. Whittaker, J.A., Rekab, K., Thomason, M.G.: A Markov chain model for predicting the reliability of multi-build software. Inf. Softw. Technol. **42**(12), 889–894 (2000)