Less is More: Estimating Probabilistic Rewards over Partial System Explorations

ESTEBAN PAVESE, Departamento de Computación, Universidad de Buenos Aires
VÍCTOR BRABERMAN, Departamento de Computación, Universidad de Buenos Aires and CONICET
SEBASTIAN UCHITEL, Departamento de Computación, Universidad de Buenos Aires and Imperial College London and CONICET

Model-based reliability estimation of systems can provide useful insights early in the development process. However, computational complexity of estimating metrics such as mean time to first failure (MTTF), turnaround time (TAT), or other domain-based quantitative measures can be prohibitive both in time, space and precision. In this paper we present an alternative to exhaustive model exploration–as in probabilistic model checking–and partial random exploration–as in statistical model checking. Our hypothesis is that a (carefully crafted) partial systematic exploration of a system model can provide better bounds for these quantitative model metrics at lower computation cost. We present a novel automated technique for metric estimation that combines simulation, invariant inference and probabilistic model checking. Simulation produces a probabilistically relevant set of traces from which a state invariant is inferred. The invariant characterises a partial model which is then exhaustively explored using probabilistic model checking. We report on experiments that suggest that metric estimation using this technique (for both fully probabilistic models and those exhibiting non-determinism) can be more effective than (full model) probabilistic and statistical model checking especially for system models where the events of interest are rare.

CCS Concepts: •**Software and its engineering** → **Software performance; Software reliability;**

Additional Key Words and Phrases: Quantitative modelling, probability, model checking, partial verification, estimation

ACM Reference Format:
Esteban Pavese, Víctor Braberman and Sebastian Uchitel, 2013. Automated Reliability Estimation over Partial Systematic Explorations. *ACM Trans. Softw. Eng. Methodol.* V, N, Article 1 (January 2016), 49 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Model-based automated verification for assessing quantitative metrics of systems, such as system reliability, aims to provide insights early in the development process that can significantly reduce not only development costs, but also costs associated with deploying faulty systems. However, traditional metrics such as mean time to first failure (MTTF) or availability times require models that support describing probabilistic, non-deterministic and timed behaviour. Unfortunately, models that succeed at effectively conveying all of this information are complex enough that estimating such metrics can be prohibitive in time, space and/or precision.

Model checking is emerging as an effective system verification method. In particular, quantitative guarantees such as those employed for reliability assurance, can be com-

This work was partially supported by grants ANPCyT PICT 2011-1774, ANPCyT PICT 2012-0724, ANPCyT PICT 2013-2341, ANPCyT 2014-1656, CONICET PIP 11220110100596CO, CONICET PIP 11220130100688CO, UBACYT 036, UBACYT 0384 and MEALS 295261.

DOI: http://dx.doi.org/10.1145/0000000.0000000

puted for complex models using the techniques developed in the area known as probabilistic model checking [Vardi 1985; Bianco and De Alfaro 1995]. These techniques take probabilistic models as inputs (such as Markov Chains or Segala Automata) and can assess quantitative properties through exhaustive exploration of the model state space and subsequent numerical analysis.

Applicability of probabilistic model checking for reliability assessment of complex models is threatened by the size of the model. Although state space reduction techniques exist [Larsen et al. 1997; Clarke et al. 1999], they may still fail to prevent state explosion to a manageable extent on complex enough models. Further, even in the event that the entire state space can be explored in its totality, its size typically impedes exact numerical calculation (e.g. Gaussian elimination) of reliability metrics. To overcome this limitation, iterative methods (such as Jacobi or Gauss-Seidel) that approximate metrics need to be used. However, these methods do not always have convergence guarantees, and when they do converge they may do so slowly; as much as to become intractable. The latter problem is heightened in the case of metrics related to rare events (e.g. reliability estimation for models where the probability of failure in a fixed period lies below 10^{-5}). In this case, since the execution budget time for the iterative methods is not infinite, exhausting this budget can lead to iterations being cut short far from the actual value of the metric being estimated. In other words, this early termination causes probabilistic model checking techniques to report on results that may not be exact, but rather only an approximation of the actual result. Good news for reliability assessment is that these approximations are a *lower bound* on the actual value. However, the distance between this actual value and the lower bound obtained is in general unknown.

In summary, although probabilistic model checking may seem to promise exact calculation of quantitative reliability properties, state space explosion and application of numerical methods can be computationally prohibitive or result in poor approximations. Despite these limitations, probabilistic model checking can provide bounds with 100% confidence for reliability metrics even though the distance of these bounds to the real value cannot be known in general.

Numerical analysis and, to some extent, state explosion can be avoided using statistical techniques. These techniques are based on applying statistical inference over a finite set of sample executions extracted from the model. Variations of these approaches are usually referred to with the umbrella term of Monte Carlo estimations. When using these techniques to estimate quantitative metrics, the actual population mean X is approximated through an estimator such as the sample mean $\overline{X}$ [Lyu 1996]. Of course, such estimation is subject to statistical error and thus it is crucial to understand how far and with what likelihood the estimator deviates from the actual mean. This contrasts with probabilistic model checking, which does not suffer from such statistical imprecision.

The deviations from the actual value that result from the specific samples used while performing Monte Carlo based estimations is usually conveyed in terms of statistical errors and confidence intervals. Bounds for statistical error and confidence intervals can be computed, based partly on the number of samples being analysed. Although significant progress for fast generation of random walks over models has been made [Nimal 2010; Rabih and Pekergin 2009], sample generation can be very costly time-wise even for analyses with modest guarantee requirements, simply due to the sheer number of samples required [Sawilowsky 2003].

The number of samples required is not the only limiting factor for these approaches: sample-based reliability estimations must also take into account the length of samples. Sample length can be particularly problematic, since sampled executions must reach a state satisfying a—usually unlikely—property (e.g. a failure) in order to allow computing an estimator. This may turn sample generation for high-reliability systems intractable.

In summary, statistical techniques can provide approximations with measurable confidence intervals and error bounds. However, in the presence of models with rare events, the required number and length of samples may make such techniques intractable, and attempts to reduce either sample size or length might result in weakened (or downright lost) statistical guarantees over results.

In this paper we present an alternative to exhaustive model exploration—as in probabilistic model checking—and partial random exploration—as in statistical model checking—which may counter some of the limitations of existing model-based reliability verification techniques. Our hypothesis, inspired on the Pareto principle, is that a (carefully crafted) partial systematic exploration of system models can be effectively analysed to provide good bounds on quantitative metrics with lower computation cost. More specifically, probabilistic model checking of a submodel of the system can bound the value of these metrics for the complete model, and do so in a cost-effective manner. Furthermore, it can produce better approximations, given equal time and memory budgets, than those that both probabilistic and statistical model checking can achieve.

We hypothesise that there is a potential gain in identifying a small, but probabilistically significant, portion of the state space, considering all other states as failures and performing probabilistic model checking on the resulting submodel. The intuition is that, in contrast to full-model probabilistic model checking, performing a probabilistic check on only a portion of the full model allows for faster iterations of the numerical analysis methods. Consequently, more iterations can be performed within the same time budget and, for slowly converging models, a better approximation may be achieved.

More specifically, in this paper we present a novel automated technique for quantitative metric estimation that combines simulation, invariant inference and probabilistic model checking. We use model simulation to produce a set of traces that represent likely behaviour of the full model. These traces are used to infer an invariant that describes the state space explored during the simulation. A submodel, which restricts the states by not allowing those that do not satisfy this invariant, is constructed and the value of the desired metric is computed over this partial model using a probabilistic model checker.

The technique we propose obtains lower bounds to the actual values of the desired metrics with 100% confidence (as full-model probabilistic model checking and in contrast to statistical model checking). In a more technical note, our technique provides a lower bound on the expectation of a random variable. This random variable is modelled as a reward structure over suitable probabilistic models.

Preliminary evidence shows that the lower bounds achieved (for a fixed budget of time and memory) are higher than those obtained by full model probabilistic and stochastic model checking, especially for models where the probability of reaching the interesting property is low given a fixed time. High bounds are of special interest in reliability, as they allow to argue a reliability case even in the absence of the exact values. Furthermore, automated invariant generation seems to perform reasonably well against domain-expert provided invariants, and have the added advantage of being useful when such expert-provided invariants are unavailable.

The remainder of this paper, which builds and expands our previous work [Pavese et al. 2013], is organized as follows. In section 2 we provide background on probabilistic systems modelling, verification and metrics such as reliability. In Section 3 we describe our approach to metric estimation. Section 4 provides case studies illustrating the approach and comparing results to existing techniques. In Section 5 we present a discussion of our results and of related work. Finally we offer our conclusions and discuss future work in Section 6.

2. BACKGROUND

Quantitative estimation of system metrics has largely focused on reliability estimation, employing measures such as mean time to first failure, mean time between failures, turnaround time, time to reset and many others [Lyu 1996; Musa et al. 1987], some of which will be explored in Section 4. Nevertheless, as we shall discuss further in this paper, the techniques we propose can be applied in a straightforward manner to several of these metrics. Recall however that the approximations we obtain are lower bounds on the actual expected value. Therefore, care must be taken in the interpretation of this bound depending on the metric being evaluated. For example, having a lower bound on mean time to failure is useful for arguing reliability; if the lower bound surpasses the expected lifetime of the system, reliability can be argued successfully. But for metrics such as cost where the aim is in minimisation, assuring a lower bound on its expected value may not be as useful.

In order to calculate these desired measures, practitioners base their efforts on some model that captures the quantitative behaviour of the system. Often, these behaviours are not unique and the choice between them is probabilistic in nature. This stochastic failure behaviour is usually modelled through formalisms based on Markov chains.

2.1. Probabilistic modelling

There exist several formalisms derived from Markov chains for modelling the probabilistic behaviour of systems and their components. *Segala's Simple Automata* (SSA) [Segala 1995] are one such formalism that—besides allowing for probabilistic behaviour—also allows for the modelling of non-deterministic behaviour. The introduction of non-deterministic transitions permits the modelling of underspecified choice, that is, the option of selecting between several possible actions, but for which the actual probability may not be actually known. Alternatively, it may be the case that the component is known to make these choices with several different probabilities, which may be known to be selected from a set of possible probabilistic distributions. However, the mechanism by which a distribution is chosen from this set may be left underspecified. In this case, the distribution choice is made non-deterministically.

Definition 2.1 (*Segala Simple Automaton [Segala 1995]*). A *Segala Simple Automaton* (SSA) is a tuple $\langle S, s_0, A, R\rangle$ where $S \subseteq V \to C$ is a finite set of states, defined by mapping a finite set of variables V to values on a finite subset of $\mathbb{Z}$, C.

Let $\mathcal{D}(S)$ stand for the set of all possible discrete distributions on sample set S. $s_0 \in S$ is the *initial* state. A is a finite set of action labels. The finite relation $R \subseteq S \times (A \cup \{\tau\}) \times \mathcal{D}(S)$ is the transition relation where the transition target is defined by a distribution on target states. In particular, R must be such that for every $s \in S$ there exists at least one $a \in A \cup \{\tau\}$ and $\mu \in \mathcal{D}(S)$ such that $(s, a, \mu) \in R$.

In the previous definition, τ stands for a distinguished action, internal to the component being modelled. In a particular case, the relation R of an SSA M may be such that, for every state $s \in S$, there exists only one tuple of the form $(s, a, \mu) \in R$. In such a case, there are actually no non-deterministic choices; for every state there is exactly one possible transition distribution. Segala Simple Automata that show this absence of non-determinism are known by the name of *Discrete Time Markov Chains* (DTMC) [Segala 1995]. A Segala Simple Automata that allows non-determinism, but restricts to only one transition per state per action label is also known as a Markov Decision Process (MDP). As DTMC. MDP and SSA definitions are therefore quite similar, our work will be based on SSA. However, we shall make some observations that may apply only to one formalism or the other. In the case that it is necessary to distinguish between both, we will make such distinction clear and explicit.

As with other automata-based formalisms, complex SSA can be built compositionally using parallel composition on model components that run asynchronously but synchronise on shared actions [Segala 1995]. Note that SSA are deadlock-free, however a parallel composition could introduce deadlocks. In such cases, these deadlocks are hidden by the addition of a new self-transition labelled by a fresh internal action label, which does not modify the observable behaviour.

Definition 2.2 (*Execution traces*). Given a Segala Simple Automaton $M = \langle S, s_0, A, R\rangle$, an *execution trace* on M is a non-empty and possibly infinite sequence $\pi = s_0 \xrightarrow{\mu_0, p_0} s_1 \xrightarrow{\mu_1, p_1} s_2 \ldots$, such that for all i, $s_i \in S$ and there exists some $(s_i, a, \mu_i) \in R$ such that $\mu_i(s_{i+1}) = p_i > 0$.

As additional notation, we will note the existence of a finite execution trace π from s_0 to s_n by $s_0 \xrightarrow{\pi} s_n$. We will denote the infinite set of all possible traces through M as $\Pi(M)$, and the set of *finite* traces as $\Pi^*(M)$. Finally, given an execution trace π and $i \leq length(\pi)$, we will note π_i^s to refer to the i-th visited state, and π_i^a to refer to the i-th chosen action.

Because of the presence of non-determinism in Segala Simple Automata, the behaviour of an SSA is not uniquely governed by the probabilistic distributions on the transitions. There is a need for an additional mechanism that can resolve these non-deterministic choices. To characterise these choices, we introduce the notion of a *scheduler* or *adversary*.

Definition 2.3 (*Scheduler [Segala 1995]*). A *scheduler* for a Segala Simple Automaton $M = \langle S, s_0, A, R\rangle$ (also called an *adversary*) is a total function $S_M : \Pi^*(M) \rightarrow R$, such that if $S(\pi) = (a, \mu)$ it must be that $(last(\pi), a, \mu) \in R$. The notation $Sched(M)$ refers to the set of all possible schedulers for the SSA M; while $S(\pi)_a$ and $S(\pi)_\mu$ refer to the scheduled action and distribution, respectively, by the scheduler function evaluated over the partial trace π.

In other words, a scheduler is a function that decides, at every step during the execution of a SSA, and depending on the current execution history, the next distribution to be employed for determining the next state. It is worth noting that, as a scheduler resolves non-determinism, the coupling of a SSA with a scheduler effectively results in a fully probabilistic process, that is, a DTMC. In this paper we will focus on schedulers that are deterministic (they are functions returning a single transition) and non-Markovian (they depend on execution history). Other types of schedulers exist [Segala 1995], and we will discuss their relationship with our work further in the paper.

Definition 2.4 (*Trace probability*). Given a Simple Segala Automaton $M = \langle S, s_0, A, R\rangle$ and a finite execution trace π on M, the probability induced by π under governance of a scheduler σ is given by $Pr(\pi, \sigma, M) = \prod_{0 \leq i \leq length(\pi)} scheduled(\sigma, s_0 \ldots s_i, p_i, s_{i+1}) * p_i$, where $length(\pi)$ is the number of transitions in π and $scheduled$ denotes the function

$$scheduled(\sigma, \alpha, p, s') = \begin{cases} 1 & \text{if } \sigma(\alpha) = \mu \text{ and } \mu(s') = p \\ 0 & otherwise \end{cases}$$

2.2. Expressing properties over SSA

In order to express and analyse properties over Simple Segala Automata and other probabilistic models, these automata are coupled with modal logics whose formulae express said properties. In the case of SSA, the temporal logic pCTL [Aziz et al. 1995], which is an extension of the well known temporal logic CTL, can be used to express properties. Essentially, pCTL replaces path quantifiers present in CTL with *probabilistic quantification bounds* on the related path formulae.

2.2.1. pCTL Syntax and Semantics. pCTL formulae are built from state and path formulae, just as is the case with CTL. Let AP be a finite set of atomic propositions. If ϕ stands for a state formula, and ψ for a path formula, then pCTL formulae are built as follows

$$\begin{gathered} \phi \rightarrow true \,|\, a \in AP \,|\, \neg\phi \,|\, \phi \wedge \phi \,|\, P_{\sim p}\psi \\ \psi \rightarrow X\phi \,|\, \phi U \phi \,|\, \phi U^{\leq k}\phi \end{gathered}$$

In the above, $\sim \in \{<, \leq, =, \geq, >\}$ and $p \in \mathbb{R}, p \in [0,1]$ Given a SSA M and a mapping of states to atomic propositions $V : S \rightarrow 2^{AP}$ defining the subset of atomic propositions that are valid for each state, we can define the satisfiability of pCTL formulae for a state $s \in S$ and an execution trace $\alpha \in \Pi(M)$. Note that since M is a Simple Segala Automaton, a scheduler $\sigma \in Sched(M)$ is needed to resolve non-determinism. σ is not necessary in case we are dealing with DTMCs; or else it can be defined trivially as choosing the only distribution available to it.

$$\begin{array}{rcl} s, \sigma \models true & \Leftrightarrow & \textit{true} \\ s, \sigma \models a & \Leftrightarrow & a \in V(s) \\ s, \sigma \models \neg\phi & \Leftrightarrow & \neg(s, \sigma \models \phi) \\ s, \sigma \models \phi_1 \wedge \phi_2 & \Leftrightarrow & (s, \sigma \models \phi_1) \wedge (s, \sigma \models \phi_2) \\ s, \sigma \models P_{\sim p}\psi & \Leftrightarrow & \sum_{\alpha \in \psi_{sat}} \mu(C_\alpha, \sigma, P) \sim p \text{ where } \alpha \in \psi_{sat} \text{ iff } \alpha, \sigma \models \psi \\ & & \text{and for every other } \alpha' \in \psi_{sat} \text{ neither } \alpha \leq \alpha' \text{ nor } \alpha' \leq \alpha. \\ \alpha, \sigma \models X\phi & \Leftrightarrow & \alpha^s_1, \sigma \models \phi \\ \alpha, \sigma \models \phi_1 U^{\leq k} \phi_2 & \Leftrightarrow & \exists 0 \leq i \leq k \cdot \alpha^s_i, \sigma \models \phi_2 \wedge \\ & & \forall 0 \leq j < i \alpha^s_j, \sigma \models \phi_1 \\ \alpha \models \phi_1 U \phi_2 & \Leftrightarrow & \exists 0 \leq k \cdot \alpha, \sigma \models \phi_1 U^{\leq k} \phi_2 \end{array}$$

Informally, given a path formula ϕ, a typical pCTL state formula takes the form of a classic CTL state formula, but where path quantifiers have been replaced by the probabilistic operator $P_{\sim p}$. Thus, a state formula $P_{\leq p}\phi$ (resp. $P_{\geq p}\phi$), is true at a given state of the system if its possible evolutions from that state, driven by the σ scheduler, satisfy the formula ϕ with probability at most (resp. at least) p.

In the case of probabilistic verification of pCTL properties of Simple Segala Automata, it is usually more interesting to learn about how the property stands when the SSA is subject to the *best* and *worst* possible schedulers, that is, those that convey the *extrema* probabilities, i.e. the minimum and maximum possible probabilities. Algorithms exist that can answer these questions effectively [Bianco and de Alfaro 1995] for the class of non-Markovian, deterministic schedulers that we deal with in this paper.

In addition to pCTL property specification, *reward structures* are used to convey some sense of value to traces from probabilistic models such as SSA, that can then be weighed by their corresponding probability. For example, a transition reward structure that assigns a value of 1 to each transition is a standard way of defining overall time steps cost for the traces of a model. This provides a good way to model discrete time, and reliability measures such as mean time to failure can be easily interpreted over this notion of time.

The value of a reward is a random variable itself, as the accumulation of rewards over traces will depend on the probability of the transitions taken. By weighing the values of this modelling of time over the (possibly infinite) set of traces and their probabilities, we can obtain the expectation –or bounds to this expectation– of running time for an arbitrary execution.

Definition 2.5 (*Reward Structures [Qureshi and Sanders 1996]*). Given a SSA $M = \langle S, s_0, A, R\rangle$, a *transition reward structure* is a function $\rho : S \times A \times S \to \mathbb{R}_{\geq 0}$.

Given a trace π of a SSA M, and a reward structure ρ over M, the *path-reward* of π is the sum of the reward of each of its transitions. We will extend this notation and note $\rho(\pi)$ to refer to the path-reward of π based on reward structure ρ. It is important to note that a reward structure assigns a non-negative reward value to transitions. Therefore, if we were to take any prefix π_{prefix} of a trace π, the path-reward of π_{prefix} will necessarily be *at most* that of π.

We will note $\Pi_{S_{end}}(M)$ (where S_{end} is a set of states) to refer to the possibly infinite set of all execution traces of M, but where they have been pruned so that the last state of each trace is one of those in S_{end}, and no other state in S_{end} exists in the trace before the end. Note that $\Pi_{S_{end}}(M)$ may contain traces of infinite length (i.e., those that never reach a state in S_{end} and therefore have not been pruned). This definition will allow us to define the value of a reward structure for *reachability* properties.

Definition 2.6 (*Reachability reward values [Qureshi and Sanders 1996]*). Let $M = \langle S, s_0, A, R\rangle$ be a SSA, $S_{reach} \subseteq S$ be a set of states from M, σ a scheduler for M and ρ a reward structure over M. The reachability reward value for S_{reach} under the conditions above is a random variable $X_{reach}(S_{reach}, M, \sigma)$ on $\mathbb{R}_{\geq 0} \cup \{+\infty\}$ such that the probability p of $X_{reach} = k$ is defined as $Pr(\sigma, X_{reach} = k) = \sum_{\pi \in \Pi_{S_{reach}}(M), \rho(\pi)=k} Pr(\pi, \sigma, M)$

In the definition above, X_{reach} is a random variable denoting the reward value for a random execution trace until it reaches a state in S_{reach}. As such, it may be of interest to know its *expected* or *mean* value, that is, the expected value taking into account every possible execution trace. We will note this expected value as $\overline{X_{reach}}$. Note that S_{reach} may contain states for which there is a non-zero probability that they won't be reached at all. In such a case, it will happen that $\Pi_{S_{reach}}$ will contain some infinite paths. More so, these infinite paths may themselves accumulate infinite reward. In such cases, the mean $\overline{X_{reach}}$ is defined to be ∞.

Even though we restrict ourselves to reachability rewards, this more than suffices for our intended verification setting. For example, consider the mean time to failure metric. In order to be able to calculate this metric, we first need to be able to describe what a failure means in our system. In other words, we need to identify which system states model a failure, or an irrecoverable situation. In the setting of this work, these states would comprise the interesting S_{reach} set. Calculating the mean reachability reward value to this S_{reach} set effectively calculates the mean time to failure of the system.

3. APPROACH

This section formally defines an approach to computing bounds to reward values of probabilistic system models. The approach is based on calculating the reward values for only a partial systematic exploration of the model's state space. We first define what is meant by a partial exploration and show that the mean reward computed over these partial explorations is indeed a lower bound to the mean reward computed over the entire system model. We then show how some partial explorations can be specified declaratively through invariant properties that drive the exploration, discussing at length the details of the procedure. Finally, we show how these invariant-driven partial explorations can be obtained automatically from any given model, without need for human intervention. In the next section we will show, via some case studies, that given a fixed budget of time and memory, analyses performed over automatically inferred invariant-driven partial explorations perform at least as well as, and sometimes outperforms, partial explorations driven by manual specification.

3.1. Partial Explorations

We refer to a partial exploration of a system model as a submodel. Intuitively, a submodel of a probabilistic process M is a model that retains a subset of the states and transitions of M and in which all other states in M have been abstracted away into a new λ trap state. Moreover, the retained states include the initial state, and all other retained states are reachable from this initial state. Formally, a submodel of a probabilistic model is defined as follows, where $supp(\mu)$ denotes the *support set* of the distribution μ, that is, the set of values x_i for which $\mu(x_i) > 0$:

Definition 3.1 (*Submodel*). Given a SSA $M = \langle S, s_0, A, R \rangle$, a *submodel* of M is another SSA $M' = \langle S' \cup \{\lambda\}, s_0, A, R' \rangle$ such that $S' \subseteq S$, $s_0 \in S'$, and $R' \subseteq (S' \cup \{\lambda\}) \times (A \cup \{\tau\}) \times \mathcal{D}(S' \cup \{\lambda\})$ is such that for all $a \in A$

(1) for each $(\lambda, a, \mu_{R'}) \in R'$, it must be the case that $supp(\mu_{R'}) = \{\lambda\}$ and $a = \tau$;
(2) for all $s \in S'$ and $a \in A \cup \{\tau\}$
   a) for all $\mu_{R'}$ such that $(s, a, \mu_{R'}) \in R'$, there exists μ_R such that *i)* $(s, a, \mu_R) \in R$, *ii)* for all $s' \in S'$ $\mu_{R'}(s') = \mu_R(s')$, and *iii)* $\mu_{R'}(\lambda) = 1 - \sum_{s' \in S'} \mu_R(s')$.
   b) for all μ_R such that $(s, a, \mu_R) \in R$, there exists $\mu_{R'}$ such that *i)* $(s, a, \mu_{R'}) \in R'$, *ii)* for all $s' \in S'$ $\mu_{R'}(s') = \mu_R(s')$, and *iii)* $\mu_{R'}(\lambda) = 1 - \sum_{s' \in S'} \mu_R(s')$.

Clause 1 states that transitions originating on the λ state all lead back to the same λ state, and that they do so through the model's internal action τ. Clause 2 states that action transitions on the submodel are drawn from the original model ones, that is, if an action transition is possible at a given state in the submodel, that action must have been possible from the same state in the whole model. Further, it also states that the probabilities on those transitions are also preserved from the original model, except for the case of those that were rerouted to the λ state, which accumulates the probabilities of those rerouted transitions. Finally, Clause 2 states that every transition on the original model is preserved on the submodel for each of the states present in the submodel, while the λ states accumulates the remaining probability.

There is a close relationship between the schedulers that can be defined for a given model M and those that can be defined on its submodels M'. Intuitively, any scheduler σ for M is still a valid scheduler for M', although with some changes. In particular, transitions that over the original model traverse to states that do not exist in the submodel are instead rerouted to the λ state. The following definition captures these changes.

Definition 3.2 (*Restricted scheduler*). Let $M = \langle S, s_0, A, R \rangle$ be a SSA, and $M' = \langle S', s_0, A, R' \rangle$ one of its submodels. Let σ be a scheduler for M. Also, let $\alpha \in \Pi^*(M')$ which implies that either $\alpha \in \Pi^*(M)$ or $last(\alpha) = \lambda$. The *restriction* of scheduler σ to M' is another scheduler σ' for M' such that

— if $last(\alpha) = \lambda$ then $\sigma'(\alpha) = (\tau, \mu)$ where μ is such that $supp(\mu) = \{\lambda\}$.
— if $last(\alpha) \neq \lambda$ and $\sigma(\alpha) = (a, \mu)$ and $(a, \mu) \in R'(last(\alpha))$, then $\sigma'(\alpha) = (a, \mu)$.
— if $last(\alpha) \neq \lambda$ and $\sigma(\alpha) = (a, \mu)$ and $(a, \mu) \notin R'(last(\alpha))$ then it must be the case that, because of Definition 3.1, there must exist $(a, \mu') \in R'(last(\alpha))$ such that
   — $(supp(\mu') \setminus \{\lambda\}) \subseteq supp(\mu)$;
   — for each s' in $supp(\mu) \cap supp(\mu')$ it holds that $\mu(s') = \mu'(s')$;
   — $\lambda \in supp(\mu')$ and is such that $\mu'(\lambda)$ captures the remaining probability.
   In such cases, $\sigma'(\alpha) = (a, \mu')$.

We also say that σ' is the scheduler σ *restricted to* M'.

It is also easy to see that any scheduler for a submodel can be extended to a scheduler that is valid for the complete model—in fact, it can be extended to possibly many schedulers. In other words, every valid scheduler for a submodel is a restriction of one or more schedulers of the complete model.

Submodels are key to our approach since they conservatively approximate the value of reward structures for reachability properties. That is, given a reward structure ρ for a model M and a scheduler σ, the mean reward value of ρ under σ for M until reaching some state in a distinguished set $S_{reach} \subseteq S$ is always greater or equal to the mean reward value of any of its submodels M', under the same scheduler restricted to M', until reaching a state in the set $S'_{reach} = (S_{reach} \cap S') \cup \{\lambda\}$.

THEOREM 3.3. *(Submodels bound reward values). Let $M = \langle S, s_0, A, R\rangle$ and $M' = \langle S', s_0, A, R'\rangle$ be two SSA with state spaces S and S' and such that M' is a submodel of M. Let $S_{reach} \subseteq S$ be a set of states representing the interesting events and σ a scheduler for M. Also, let σ' be the restriction of σ to M'. Then $\overline{X_{reach}(S'_{reach}, M', \sigma')} \leq \overline{X_{reach}(S_{reach}, M, \sigma)}$.*

PROOF. Note that, for every trace in the complete model, it either exists completely in the submodel, or the submodel contains only a prefix that is extended by the λ state. Since reward structures are based on transitions, every trace in the full model accumulates *at least* as much reward to each of the interesting states (possibly ∞) as the corresponding trace (or prefix) in the submodel. Hence these prefixes contribute to $\overline{X_{reach}(S'_{reach}, M', \sigma')}$ at most what their extensions in M contribute to $\overline{X_{reach}(S_{reach}, M, \sigma)}$.

Alternatively, if the submodel allows a trace that never reaches either λ or one of the target states in $S' \cap S_{reach}$, then this trace also exists in the complete model. In such a case, both $\overline{X_{reach}(S_{reach}, M, \sigma)} = \overline{X_{reach}(S'_{reach}, M', \sigma')} = \infty$. □

The above result entails that if computing the value of a reward structure for a system model is intractable, it can be conservatively approximated on any of its submodels. In the case of Segala Simple Automata, because of the presence of non-determinism, it is interesting to examine the case for the extrema schedulers. The following corollary captures the bounding relation for these extreme values.

COROLLARY 3.4. *Let the SSA M as defined in the previous theorem, and its submodel M', be SSAs. Let σ_{min} and σ_{max} be two schedulers for M such that, for any other scheduler σ for M*

— $\overline{X_{reach}(S_{reach}, M, \sigma_{min})} \leq \overline{X_{reach}(S_{reach}, M, \sigma)}$*; and*
— $\overline{X_{reach}(S_{reach}, M, \sigma_{max})} \geq \overline{X_{reach}(S_{reach}, M, \sigma)}$.

In turn, let σ'_{min} and σ'_{max} be schedulers for M' such that for other schedulers σ' for M' it holds that

— $\overline{X_{reach}(S'_{reach}, M', \sigma'_{min})} \leq \overline{X_{reach}(S'_{reach}, M', \sigma')}$*; and*
— $\overline{X_{reach}(S'_{reach}, M', \sigma'_{max})} \geq \overline{X_{reach}(S'_{reach}, M', \sigma)}$.

Under these conditions, it holds that $\overline{X_{reach}(S'_{reach}, M', \sigma'_{min})} \leq \overline{X_{reach}(S_{reach}, M, \sigma_{min})}$ and also that $\overline{X_{reach}(S'_{reach}, M', \sigma'_{max})} \leq \overline{X_{reach}(S_{reach}, M, \sigma_{max})}$.

In a similar manner as Theorem 3.3, this result indicates that estimations for the minimum and maximum rewards over a submodel yield lower bounds for the actual minimum and maximum rewards, respectively, for the whole model.

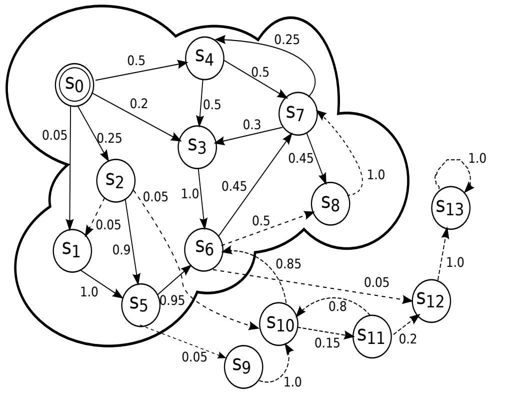


Fig. 1: Example partial exploration of a state space

Key questions are which submodels are cost-effective (i.e. provide good approximations at reasonable computation cost) and how to find them. Another important question to address is whether effective submodels provide reasonable approximations in general. In the next subsection we discuss one particular way of driving the generation of submodels that results in cost-effective reward computation. Later, in Section 4 we will argue that the submodels obtained through our approach are effective at estimating bounds for reward values.

3.2. Automatic Submodel Generation

Although any submodel will provide a lower bound for the value of a given reward structure, the key to a tractable reward estimation technique is to identify a submodel for which its reward value can be computed within a reasonable time budget, and for which the resulting bound is a useful approximation to the actual reward value of the full model. In particular, we have already established in previous work [Pavese et al. 2010] that not every submodel is good for estimating the average value of these reward structures. For example, in that work we have already shown that submodels obtained as the result of a depth-first search exploration are generally very bad at providing good reward estimates. Conversely, we have shown that submodels obtained through breadth-first search explorations outperform DFS ones in general. Nevertheless, they still do not provide good estimates in general either. In other words, not all submodels are created equal; two submodels similar in size can obtain wildly different estimates.

Regrettably, and independently of the fact that the reward value for the full model is unknown, the problem of computing an exact solution (i.e. obtaining the "best" submodel for reward computation) is intractable [Jamieson and Dean 2007]. In this section we discuss a heuristic for automatically constructing submodels that can provide better bounds for reliability at lower computation cost than both full model checking and Monte Carlo approaches.

Our approach adopts a heuristic based on the reasoning that the submodel construction strategy should aim to identify a portion of the model that is probabilistically dense, that is, a small submodel for which the probability of reaching the λ trap state in a given fixed time is low. Such models will contain probabilistically likely loops that delay the traces from reaching the submodel boundary, hence contributing to a higher bound for the reward being estimated.

The problem of finding the most probabilistically dense submodel is NP-hard [Jamieson and Dean 2007]. Our approach attempts to approximate such a submodel through bounded simulation. Hence, the basis of our approach involves the simulation of several traces over the full model. The resulting set of finite traces, if sufficiently large and consisting of sufficiently long traces, is likely to cover a good part of a probabilistically dense submodel. These traces form the basis for building our submodels. The smallest submodel that includes the set of states and transitions covered by the simulated traces can be constructed easily by simply adding any non-visited transitions between any two visited states, abstracting all non-visited states into the λ trap state, and adding transitions to the λ state for whichever state has transitions that were neither explored nor added in the first step. Figure 1 shows such a construction, where solid lines represent transitions that were covered by the simulated traces, while dotted lines are transitions in the model that were not covered. States outside the boundary have not been covered, and would be abstracted away into the λ state of the submodel.

However, submodels built through such a procedure are likely to have relatively short traces that escape the submodel (see path $s_0, s_2, s_{10}, \ldots$ in the figure). These short traces contribute a relatively high probability of escaping the submodel (in general, the shorter the prefix, the larger the probability of the set of traces that extend from it), reducing the bound estimated by the submodel. Note that, in our example, s_{10} falls back within the boundary to s_6 with high probability. If we were to include this state into our submodel, and according to the submodel completion procedure outlined before, the result would be that the bound estimated by the submodel would be raised. This is consistent with our experimentation in [Pavese et al. 2010]. In that work, we observed that submodels generated with a breadth-first search strategy tend to approximate reliability measures better, as they delay the chance of escaping traces until the lowermost levels of the breadth-first exploration.

In the approach that we detail in this present work, rather than adopting a syntactic notion of breadth first traversal for extending the submodel determined by a simulation of the full model, we take a more semantic approach based on the attributes of states visited during the simulation. We compute state invariants based on the states visited during the simulation and then add to the submodel any states that satisfy the invariant, as well as the transitions between them. In this way, we expect to add behaviour that, although not exactly equivalent to what was simulated, represents variations in terms of symmetries, race conditions, and independent events [Baier and Katoen 2008], and contributes significantly to the probabilistic weight of the submodel.

We now formally define our submodel construction method. We start with the notion of invariant of a set of traces.

Definition 3.5 (*Invariant*). Given a probabilistic process $M = \langle S, s_0, A, R\rangle$, and a set of finite execution traces T obtained from said model, an *invariant* of M through T is a state predicate ψ on the variables of M such that for every execution trace $t = s_0 \xrightarrow{p_0} s_1 \xrightarrow{p_1} s_2 \ldots\ s_n \in T$, it holds that $\forall 0 \leq i \leq n, s_i \models \psi$.

An invariant then induces a unique submodel as follows:

Definition 3.6 (*Invariant-driven submodel*). Let $M = \langle S, s_0, A, R\rangle$ be a SSA and ψ a state invariant; an *invariant-driven submodel* induced by ψ is a submodel $M' = \langle S' \cup \{\lambda\}, s_0, A', R'\rangle$ of M such that

a) each state $s' \in S'$ is such that $s' \models \psi$;
b) for each $s'_1 \in S', s'_1 \neq s_0, s_0 \xrightarrow{\pi} s'_1$; and finally
c) for all states $s'_2 \in S \setminus S'$ such that there exist $s'_1 \in S, (s'_1, a, \mu_R) \in R$ with $\mu_R(s'_2) > 0$, it is the case that $M, s'_2 \not\models \psi$.

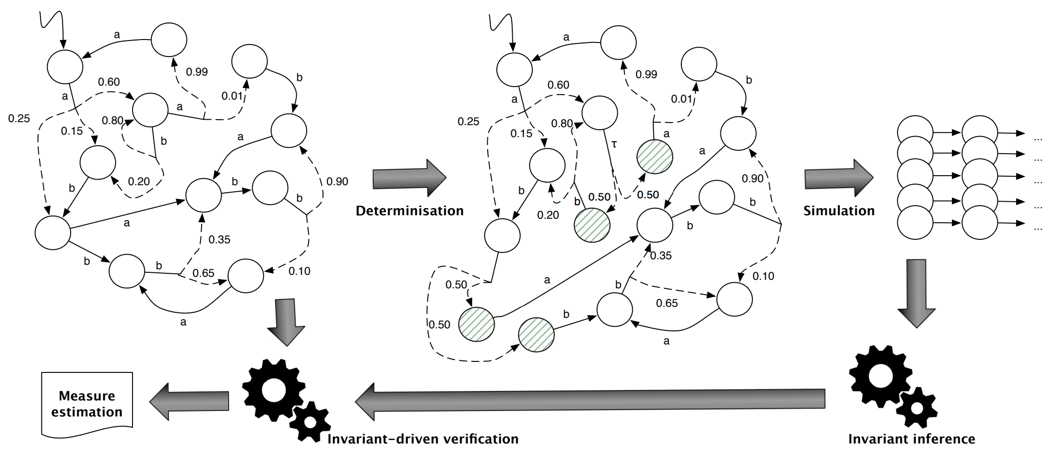


Fig. 2: Workflow for partial exploration analysis

In other words, if a state s'_2 not in the submodel is directly reachable from a state s'_1 in the submodel, it must be the case that s'_2 violates ψ. The submodel is thus maximally connected from the initial state through the invariant ψ.

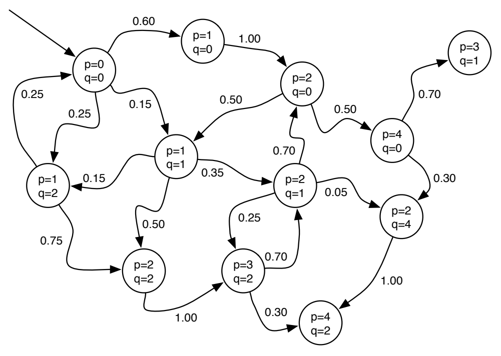
Our approach places a focus on maximising the automation of the estimation process. Therefore, we aim at automatically obtaining invariants. To this end, we produce probabilistically driven walks over the full system model, bounded in length, while we record the states (i.e. variable valuations) traversed. We use the tool Daikon [Ernst et al. 2007], an invariant inference engine, to obtain predicates that hold over all traversed states. These invariant predicates, in turn, are used to synthesise an observer automaton that can drive the generation of a submodel via its parallel composition with the system model.

It is important to note that for working with Segala Simple Automata it is necessary to resolve non-deterministic transitions during the probabilistically driven walk generation. In this paper, we have chosen to replace non-deterministic transitions with an equiprobable distribution that chooses between the possible target distributions. The correctness of our approach is not hampered by this choice, as in fact any method of resolving non-determinism would serve our needs – any non-determinism resolution approach yields a valid submodel. In turn, reward calculations over these submodels yield correct bounds. However, it is left to be studied if this is the best way to resolve non-determinism. That is, whether a different determinisation scheme exists that produces a DTMC that, when analysed for determining reliability bounds, obtains better bounds or does so with less computational effort. We discuss on this decision and possible alternatives in Section 5.

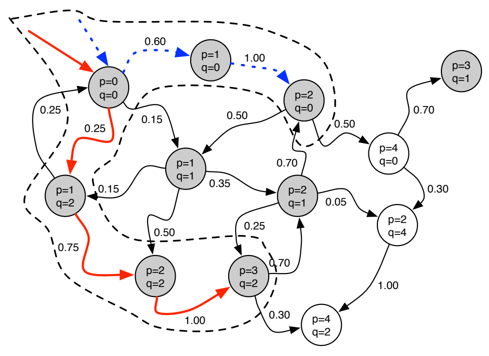
The first step of our approach is then to perform simulation over an *equiprobably determinised* version of the original SSA.

Definition 3.7 (*Equiprobably Determinised Segala Simple Automaton*). Let $M = \langle S, s_0, A, R\rangle$ be a Segala Simple Automaton. The *equiprobably determinised* Segala Simple Automaton of M is a DTMC $M_{det} = \langle S_{det}, s_0, A, R_{det}\rangle$ constructed in such a way that $S \subseteq S_{det}$, and for every $(s, a, \mu) \in R$:

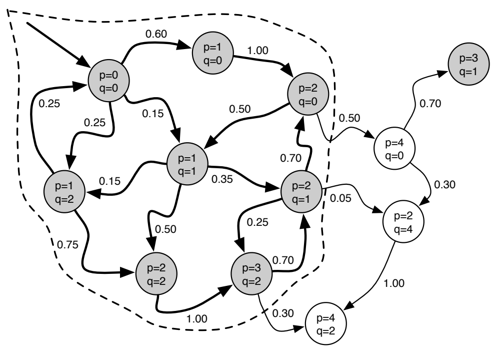
— If (s, a, μ) is the only transition for s in M, add the transition to R_{det};



(a) Model to be explored



(b) Simulation traversal



(c) Invariant-driven submodel

Fig. 3: Stages to obtain an invariant-driven submodel

— otherwise, take all (s, a_i, μ_i). Add i states $t^s_1, \ldots, t^s_i$ to S_{det}. Add a transition (s, τ, μ) to R_{det} where $\mu(t^s_j) = 1/i$ for each of those added states, and 0 everywhere else. Finally, add transitions (t^s_i, a_i, μ_i) to R_{det} for each of the added states.

Once the invariant is inferred through the simulations, it is used to generate the partial submodel of the *original* SSA. Figure 2 depicts the workflow of this approach while in Figure 3 we show the intermediate stages of the submodel generation.

The first step in Figure 2 is model determinisation. Applying determinisation to the original model yields another one where some intermediate states have been added to account for the replacement of non-deterministic transitions for probabilistic ones (these added states appear shaded in the second model of Figure 2). The second part of the workflow consists of simulation, invariant inference and obtaining a submodel.

The stages through which we accomplish this goal are detailed in Figure 3. We start out with the simple model shown in Figure 3a where the states are identified by two integer variables p and q. We have omitted action labels in this case to aid readability of the example. The first step of the approach is to perform a set of length-bounded simulations of the model. For the sake of this small example, we have obtained two simulation traces, the first shown in a blue dotted line, while the second one is shown with solid red transitions.

The result of this simulation phase is that we have witnessed those states and transitions contained in the C-shaped area delimited by the dotted line. From these states, we infer the invariant that describes them all, which turns out to be $p \leq 3 \wedge q \leq 2$. Note that there other states that satisfy this invariant (shown greyed out in Figure 3b) but that were not traversed by the simulations. However, our notion of invariant-based submodel includes some of these states. The submodel resulting from the invariant $p \leq 3 \wedge q \leq 2$ is shown in Figure 3c. Note that the state $(p = 3, q = 1)$ does satisfy the invariant, yet it is not included in the submodel. The reason for the exclusion of this state is that the only way to reach it from the initial state is through states that do not satisfy the invariant and therefore cannot be part of the submodel.

Note that the addition of states $(p = 1, q = 1)$ and $(p = 2, q = 1)$ to the submodel has resulted in several loops being incorporated into the submodel. These loops were partially explored by the simulation, but not fully captured by them. There is a big gain in including loops such as these within our submodels, since the addition of these few states results in an increase of the probability of the behaviours that lie within this submodel. For example, if we analyse the submodel consisting only of the states explored by the simulations, we find that for this very simple example, the average number of transitions necessary for a trace to escape the submodel is 3.0133. However, if we consider the final submodel, the expected number of transitions necessary to escape quickly climbs to 6.9500, that is, more than double the previous time.

Finally, once the submodel has been obtained, we perform a full probabilistic verification over this submodel. As we explained earlier, this yields correct bounds to the rewards of interest.

4. VALIDATION

In this section we set out to answer three questions in order to validate our approach.

Q1: can our approach, when compared to model checking over full explorations, produce better bounds, in less time, for the reward values of system models? Here we also answer related questions: first, whether submodels obtained through our approach perform better than similarly-sized submodels obtained through other approaches such as predetermined exploration criteria (e.g., BFS or DFS); and second, whether the bounds we obtain are good, especially in the cases where we can actually obtain the real reward value, and therefore we can contrast our estimated bounds to the actual value.

Q2: can our approach, when compared to Monte Carlo approaches, produce better bounds, in less time, for the reward values of system models? Can the Monte Carlo approach benefit from our partial exploration techniques, that is, do Monte Carlo approaches perform better over partial explorations?

Q3: how do the reward value estimations for submodels compare when these submodels are generated from automatically inferred invariants as in our approach against manually generated ones?

Q1 and *Q2* aim at comparing our proposal with established approaches to estimation of reward values, to evaluate if our approach can complement existing techniques. The cases where the interesting states to be reached are rare events are of special interest, and we will discuss these at length. *Q3* aims at assessing the added value of automatic techniques for obtaining submodels, against the cost of gaining a deep understanding of the model to be verified and developing a good submodel manually.

4.1. Methodology

We analysed three different systems from the literature, and properties that can be expressed in terms of reward values. These systems are especially amenable to be specified in either DTMC or SSA form, depending on their reliance on non-determinism. In the following sections we provide a description of each of these systems.

For each case study, we built probabilistic system models accordingly, in order to describe behaviour. We modelled the properties of interest as state formulae, and defined appropriate reachability reward structures. We used the same input for all reward estimation techniques.

We put our approach to the test for all case studies for several automatically generated invariants varying the number and length of traces used for invariant inference. We used Daikon v4.6.4 [Ernst et al. 2007] configured to produce invariants that are conjunctions of terms of the form $x \sim y$, where x and y are either variables in the model, or integer constants, and $\sim \in \{<, \leq, =, \geq >\}$. The invariants we obtained were used to automatically build an *observer* automaton O, that monitors the validity of the invariant. This observer, when composed with the system model M, synchronises with all actions and forces transitioning into the λ trap state whenever the destination state of the intended transition would result in an invariant violation. Because of this manner of construction, the resulting subsystem is guaranteed to be a submodel of the original system model.

For *Q1* we used a modified version of PRISM v4.0.3 [Hinton et al. 2006] to perform probabilistic model checking to estimate the reward values for both the full state space and for its invariant-driven submodels. Modifications allow for batch trace generation on a format understandable by Daikon (used for invariant inference) and time and memory-use tracking (used for generating intermediate reward results and for timing out when time budget is up). Intermediate reward results were generated for visualising convergence rates. PRISM was deployed on an 8x Core Intel Xeon CPU @1.60 GHz with 8 GB RAM.

As was noted before, probabilistic verification and calculation of rewards entails solving a linear equation system as well as the model exploration. PRISM provides different numerical methods for this equation solving phase. We compared reward value computation of the full and partial explorations for the Jacobi, Gauss-Seidel and Power iterative methods, as well as several optimisations over the Jacobi and Gauss-Seidel methods. Due to space limitations, we only report on results obtained with the backwards variation of the Gauss-Seidel method (BGS), which proved to be the most effective method for full model probabilistic model checking in terms of bounds obtained for time budget, making it the best competitor against our technique. This is consistent with the fact that BGS and other relaxation based iterative methods consistently outperform other methods such as Jacobi or Power iterations [Woźnicki 2001; Kalambi 2008].

PRISM runs were considered complete when any of the following criteria held: either *a)* the *absolute* difference between results of successive iterations of the numerical method was less than 0.01 (relative differences are not an adequate stopping criteria because of slow convergence, which causes iterative methods to cut too early). Alternatively, *b)* running time reached 24 hours; or *c)* available memory, which was limited to 1 GB for each run as they were deployed concurrently, was exhausted. Note that the time measured includes only the execution of the numerical methods. This allows for convergence analysis and favours full-model exploration as the time spent on construction of the model state space is not considered. In the case of the bigger models, this construction takes as much as 6 hours of execution time. We comment on execution time for submodel generation later in the Experimental Results subsection.

For *Q2* Monte Carlo simulations were generated using the same version of PRISM and the same hardware as $Q1$. However, note that while our approach produces lower bounds to actual reward values with 100% confidence but for which precision (percentual difference between the estimation and the actual value) is unbounded, Monte Carlo produces estimations with varying degrees of confidence but for which precision can be bounded. Consequently, we aimed at performing Monte Carlo-based estimations for a range of confidence and precision values.

A critical precondition for applying Monte Carlo approaches is that all randomly generated traces must eventually reach the target states, and enough traces must be generated in order to guarantee estimations with a fixed precision and confidence. Setting a trace length horizon for the simulator to ensure all traces reach their target is typically done based on a rough estimation of the actual reward value, or an estimate of the underlying probability distribution [Sen et al. 2005a]. This seemingly circular procedure can, however, work in practice. In our particular setting, we used the estimations obtained in $Q1$ as the basis for setting this horizon for each case study. The reason for choosing such an estimate are twofold: first, the actual rewards are guaranteed to be at least as much; and second, we will already have a measure of how much effort is needed to arrive at such an estimation. We will see that even under this setting, Monte Carlo approaches require excessive effort to arrive to similar results.

In addition to comparing probabilistic model checking of submodels against Monte Carlo simulations of the complete model, we compared probabilistic model checking against Monte Carlo simulations over the same submodels. In other words, starting from the hypothesis that submodel generation does provide an added value, we wanted to further establish which approach was best for the second phase of the analysis; that is, whether probabilistic model checking or Monte Carlo evaluations should be employed.

Finally, *Q3* uses the same setup and reward estimation approach based on inferred invariants as in $Q1$. The key difference is in the method for submodel generation. Manually produced invariants for submodel generation were put forth before any of the experiments were performed. Therefore, the manually proposed invariants were not tainted by knowledge gained from the automatic approach. The main heuristic for coming up with the invariants was analysing the model and identifying necessary (and more likely) conditions for reaching the target states.

The cost of manually generating an invariant is not simple to estimate. However, coming up with invariants that are useful for a partial exploration does demand from the user a deep understanding of the model under analysis. This is in general not trivial. In the context of this work, the cost of manually generating invariants, although non-trivial, was mitigated by the fact that the authors are familiar with the models under analysis. Eliminating this author bias would require further validation, possibly involving a well-designed user study. Such a study falls outside the scope of this paper and remains future work.

4.2. Case Studies

Tandem Queueing Network. The first case study is a tandem queueing network, based on [Hermanns et al. 1999]. Queueing systems have been extensively studied in queueing theory, and analytical solutions for some variants exist. However, due to the complexity of this particular model and its different queueing modes, general analytical queueing models are not easily applicable. Generating an ad-hoc analytical formulation would require extensive expertise and time, and it would not be easily adaptable to modifications in the design of the queueing system; even if these modifications are smaller ones.

The system consists of two process queues C and M of given (and in this particular case equal) capacities. Clients queue processes for execution in the first queue while it is not full. This first queue may either route a process to the second queue after a probabilistically chosen time elapses, or it might choose to deal with the request itself. The behaviour of this first queue is governed by two different *phases*. The difference between the phases is given by the probability with which it will choose to route its requests to the second queue or deal with them directly. The second queue has no other queue on which to unload its processes. Therefore, all it can do is service its requests, and it does so after a probabilistically chosen time elapses. A failure is observed when both queues are full, as at this time, clients cannot do anything but wait until some requests have been serviced and there is room in the first queue for another process.

In our specific scenario devised for experimentation in this paper, the capacity of the queues is fixed at 1200 each. Clients are less inclined (i.e., they take more time in average) to enqueue processes as the free capacity of the queues decreases.

The reliability metric that we wish to estimate is the the *mean time to failure* (MTTF) of the system. Mean time to first failure is a widely accepted metric for reliability. This metric represents for how much time a client can expect to operate a system until it experiences its first failure. In this case, the failure is represented by the moment where a client cannot push any more tasks in the queues, and the first queue cannot offload any more work to the second. That is, a failure is met when both queues are full.

Consequently, the reward structure ρ we choose to model assigns the value 1 to every timing transition. It is generally accepted to employ *execution time* rather than *calendar time* for MTTF estimations [Lyu 1996]. While calendar time measures real time in terms of hours, weeks, etc., execution time is the time actually spent in system execution. This distinction is important for reactive systems which may have long idle times.

In our model, the state predicate that captures failure is $cliC = 1200 \wedge cliM = 1200$, and computing the mean time to failure amounts to calculating the expectation of the accumulated reward before reaching a state satisfying this predicate.

Bounded Retransmission Protocol. The second case study [D'Argenio et al. 2001] models a robust communication protocol that attempts to ensure delivery of data, the bounded retransmission protocol (BRP) [Helmink et al. 1994].

BRP is a variant of the alternating bit protocol, which allows for a bounded number of retransmissions of a given chunk (i.e., a part of a file). The protocol consists of a sender, a receiver, and two lossy channels, used for data and acknowledgements respectively. The sender transmits a file composed of a number of chunks, by way of *frames*. Each frame contains the chunk itself and three bits. The first bit indicates whether the chunk is the first one; the second one if it is the last chunk; and the third bit is the alternating one, used for avoiding data duplication.

The sender waits for acknowledgement of each frame sent. The sender may timeout if either the frame or the corresponding acknowledgement are dropped which could be caused, for example, by either the frame or the corresponding acknowledgement being dropped. When this happens, the sender resends the frame and does so repeatedly up to a specified retry limit. If the limit is reached and the transmission is terminated, the sender may be able to establish that the file was not sent (if some chunks were left unsent) or it may not know the outcome (if the last frame was sent but no acknowledgement was received). In any case, the sender may send a new file, resetting the retry count. A maximum of 256 retransmissions are attempted per file before the sender gives up and aborts transmission of the file, regardless of the size of the file being sent, Once a file is sent successfully or its transmission fails, the system waits for another file to be sent.

Protocol clients send files one at a time. Each of these files is of a different size (in number of chunks). This size may be different for each file, varying between just a few and 1500 chunks. We developed two SSA models for this problem, and analysed them separately. First, we assumed complete knowledge about the distribution of the sizes of the file being sent. Therefore, the choice of file size was modelled probabilistically, yielding a DTMC as system model, where exceedingly large or small files are modelled to be less likely to be sent than those of average size. In the second model we developed, we introduced uncertainty regarding this knowledge, and kept the size choice non-deterministic, representing this absence of information. Under this modelling choice, the second case yielded an SSA rather than a DTMC.

In this case, we also wish to estimate the mean time to the first failure, where failure is defined as the sender failing to send a complete file (*incomplete*) or not being able to establish if a file was sent successfully (*unknown*). Consequently, the state predicate describing failures is $incomplete \vee unknown$. The definition of time for this case study aims at establishing how many data packets can be expected to be sent successfully before failure. For the DTMC model we obtained the mean number of packets being sent before experiencing failure, while for the SSA model we obtained both the minimum and maximum mean number of packets, which represent the worst case and best case scenarios respectively.

IEEE 802.11 Wireless LAN.. The third case study depicts the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) mechanism of the IEEE 802.11 protocol [Institute of Electrical and Electronic Engineers 1997]. The protocol uses a randomised exponential backoff rule to minimise the likelihood of transmission collision. That is, whenever a collision was averted by a component sensing the busy carrier when trying to send data over busy media, the component is backed off (it needs to wait until trying to resend) for a time. This time is chosen randomly from a specified range of possible delays, and successive failures cause this range to increase exponentially. The goal of the protocol is to divide, as equally as possible, the access to the channel between all participants that may collide.

The model used depicts a two-way handshake mechanism of the IEEE 802.11 medium access control scheme, operating in a fixed network topology. The SSA model itself was extracted verbatim from [Henriques et al. 2012]. This model exhibits both stochastic behaviour (for example, in the randomised backoff procedure, that allows up to seven exponential backoff levels) and non-deterministic behaviour (for example, in modelling the interleaving of actions between the two independent emitter stations). Therefore, the model is an SSA.

In this case, the protocol is probabilistically guaranteed to never fail, that is, both stations will eventually be able to send their packets. However, it is interesting to know for how long they will have to wait, in average, to achieve this objective. *Turnaround time* is a measure for both reliability of systems, as it may include time necessary for error correction or recovery, as well as a measure for performance. In general, the turnaround time for a process refers to the time that elapses between it starting its task until it finishes or provides some result. The starting and finishing times may be arbitrarily defined (for example, start time may be either the moment the process takes control of execution, or rather the moment it is sent a request). In general, we may refer to turnaround as the time it takes a process to produce the required results after it is started.

In this case, we are interested in estimating the turnaround time for two stations to be able to successfully send their packets and advance to their *done* state, while avoiding potential collisions. As such, the state predicate that describes this final state is $station1 = done \wedge station2 = done$. Note that, unlike the previous case study, both stations managing to send their messages is not a rare event at all if the protocol works correctly. However, the sheer size of the model does hamper direct estimation.

Network virus infection. In this case, we analyse the behaviour of a virus infection on a computer network. This case study is based on [Kwiatkowska et al. 2009; De Nicola et al. 2006] but is heavily expanded as we will detail further on.

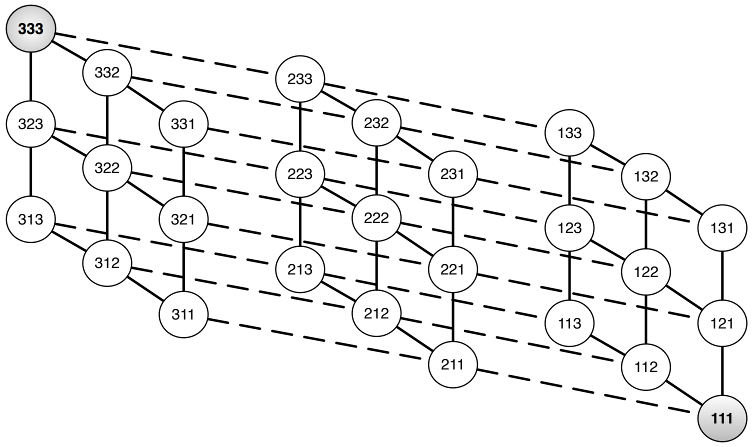


Fig. 4: A $3 \times 3 \times 3$ network cube. On the lower right the infected node 111, the target node is 333 in the upper left.

The network is a cubic grid of nodes, as opposed to the original case study in [Kwiatkowska et al. 2009] which was based on a plane grid; a cubic grid allows more virus paths as well as customising the model to sizes that quickly grow to be intractable. The size of the network is given by N, the number of nodes in any given edge of the cube. Each node is connected to the nodes at its left, right, up and down, as well as to those behind and in front of it. Nodes in the outer faces may have less connections. Figure 4 depicts a $3 \times 3 \times 3$ cubic grid.

We model the behaviour of a virus infection on a firewalled, self-healing network. In this setting, once a node is infected, it tries to propagate to its neighbouring nodes. In order to succeed, it needs to first defeat the node's firewall, and then attempt infection once the firewall is down. The network is self-healing, as healthy nodes will try to repair its infected neighbours.

The scheduling between these actions is completely non-deterministic. On the other hand, we built a SSA model of the environment that describes the probabilities of success when trying to break a firewall, infect a vulnerable node, or repair an infected node.

In each case we start with a healthy network, save for one of the corner nodes, which starts infected. The properties of interest we analyse in this case are the expected minimum time to total infection of the network, and the expected minimum time to infection of the node at the opposite corner of the initially infected one.

4.3. Experimental Results

We now present the experimental results obtained for the three research questions presented above. Table I provides a quick overview and comparison of all results given all the different techniques employed, as follows

- **Full** is a model checking effort over the full model. We always report on bounds on rewards obtained, if any. Bounds on probabilities are only reported if convergence was attained.
- **Partial** denotes our approach, ignoring simulation times. Since we also perform a model checking step, we omit probabilities that did not converge in time.
- **Monte Carlo** denotes the statistical estimation based on trace simulation.

Table I: Summary of (best) results for each technique and case study. TO denotes timeout at 24 hours. N/A denotes results that could not be obtained before timeout or were erroneous due to technique shortcomings.

Tandem Queue (mean time to failure)								
Actual value	Full		Partial		Monte Carlo		Manual	
	Result	Time	Result	Time	Result	Time	Result	Time
Unknown	4.2×10^5	TO	$\mathbf{7 \times 10^7}$	**TO**	N/A	TO	5.5×10^7	TO
Fully probabilistic BRP (mean time to failure)								
Actual value	Full		Partial		Monte Carlo		Manual	
	Result	Time	Result	Time	Result	Time	Result	Time
Unknown	OOM	TO	$\mathbf{2.5 \times 10^7}$	**TO**	N/A	TO	1.69×10^7	TO
Non-deterministic BRP (*minimum* mean time to failure)								
Actual value	Full		Partial		Monte Carlo		Manual	
	Result	Time	Result	Time	Result	Time	Result	Time
Unknown	OOM	TO	$\mathbf{5.6 \times 10^6}$	**TO**	N/A	TO	9999	126.25 s
Non-deterministic BRP (*maximum* mean time to failure)								
Actual value	Full		Partial		Monte Carlo		Manual	
	Result	Time	Result	Time	Result	Time	Result	Time
Unknown	OOM	TO	$\mathbf{9.8 \times 10^6}$	**TO**	N/A	TO	9965.87	46.26 s
WLAN (*minimum* mean turnaround time)								
Actual value	Full		Partial		Monte Carlo		Manual	
	Result	Time	Result	Time	Result	Time	Result	Time
1725.00	1725.00	628.00 s	**1725.00**	**0.98 s**	N/A	N/A	1665.63	490.05 s
WLAN (*maximum* mean turnaround time)								
Actual value	Full		Partial		Monte Carlo		Manual	
	Result	Time	Result	Time	Result	Time	Result	Time
4301.65	4301.65	54149 s	**4300.67***	**2 s***	N/A	N/A	3846.17	1085.87 s
Constrained Virus (*minimum* mean time to total infection)								
Actual value	Full		Partial		Monte Carlo		Manual	
	Result	Time	Result	Time	Result	Time	Result	Time
5200.00	OOM	TO	500.54	2771 s	N/A	N/A	**999.32**	**414 s**
Constrained Virus (*minimum* mean time to corner infection)								
Actual value	Full		Partial		Monte Carlo		Manual	
	Result	Time	Result	Time	Result	Time	Result	Time
1200.00	OOM	TO	599.54	1452 s	N/A	N/A	**999.32**	**1242 s**

— **Manual** describes the best result obtained by any of the manual invariants posed for the case study.

When possible, we also report on the actual reward values. This was either obtained analytically or through a full model check that converged, as described in each case study section.

For each experimental case, we highlight the best performer, taking into account both the quality of the result obtained as the time taken to arrive to this result. In the case of ties, or very close results with very disparaging running times, we opted to report the fastest performer as the best result. We mark these cases with an asterisk.

Question 1. When comparing probabilistic model checking of both full and partial models we are interested in considering the relationship between the inferred invariant, the size of the resulting submodel, and the value of the reward estimation obtained from it. We are also interested in gaining insight on combinations of trace length and number of traces that are likely to yield the best overall result.

Tandem Queue analyses. For the Tandem Queue case study the estimated mean time to failure, calculated using probabilistic model checking, in 24 hours over the full model was 4.20×10^5. This full model comprises $\sim 1.5 \times 10^7$ states. Regarding computations over submodels, we report on MTTF estimation (Figure 5), submodel sizes (Figure 6) and invariants obtained (Table II) for various settings of sample size and individual trace length. We report here on a subset of the values obtained, however non-reported data is in-line with the trends shown. Note that our best MTTF estimation is about

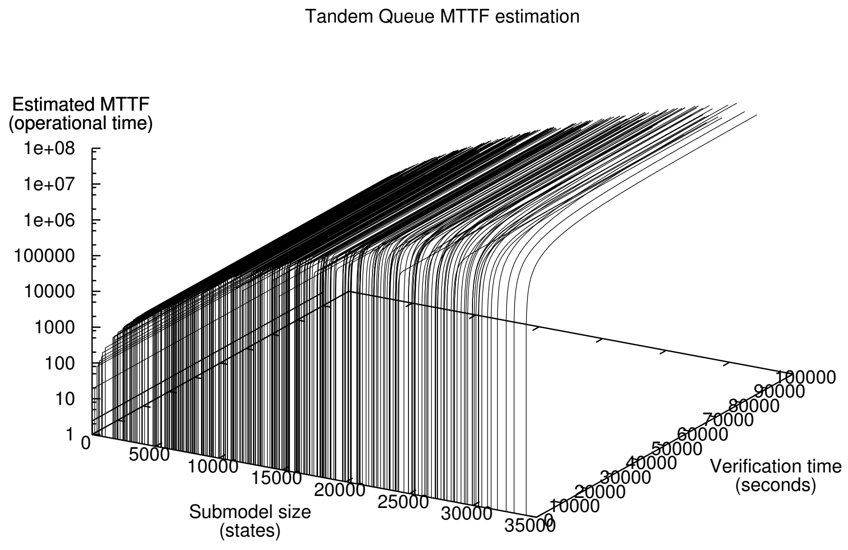


Fig. 5: Results of analysis of Tandem Queue for different sized submodels, Backwards Gauss-Seidel method.

7×10^7, a full two orders of magnitude larger than what could be estimated through full model checking. Even if this is not the actual MTTF, this jump in estimation quality could make a difference in establishing a case for reliability assurance of the system.

The first figure shows, for different automatically generated sized submodels, the estimated MTTF (shown over a logarithmic scale for convenience) along with how much time it took for the calculation to finish. Executions that finished before the 24 hour timeout are flattened on the MTTF axis at the time the result was reached. It is noteworthy that none of the automatically obtained submodels is larger than 35000 states, comprising roughly 0.25% of the states of the complete model. Despite having explored only such a small percentage of the full model, the obtained lower bound for MTTF is quite large in some cases, possibly sufficient to argue for high system reliability – MTTF is *at least* in the order of 1.0×10^7. Although very small submodels do not provide good bounds, larger submodel MTTF estimations increase dramatically, quickly rising to the 7×10^7 maximum MTTF witnessed, which is a full two orders of magnitude beyond the estimation for the full model.

An important question is whether good submodels can be obtained in a consistent fashion by parameterising trace quantity and length parameters of the simulation phase. As we have seen above, the size of the submodels is an initial indicator that the estimation procedure will result in good bounds. Figure 6 shows that such submodels can be obtained automatically in a consistent way for this example. Focusing on the upper-right corner of the figure, it can be seen that choosing values for trace length and sample size in that region consistently results in appropriate submodels.

The charts in Figure 7 expand on this. Figure 7a shows submodel sizes for different parameters, in a way similar to Figure 6. Figure 7b shows the estimated bounds for said parameters. It can easily be seen that submodel size and estimations correlate.

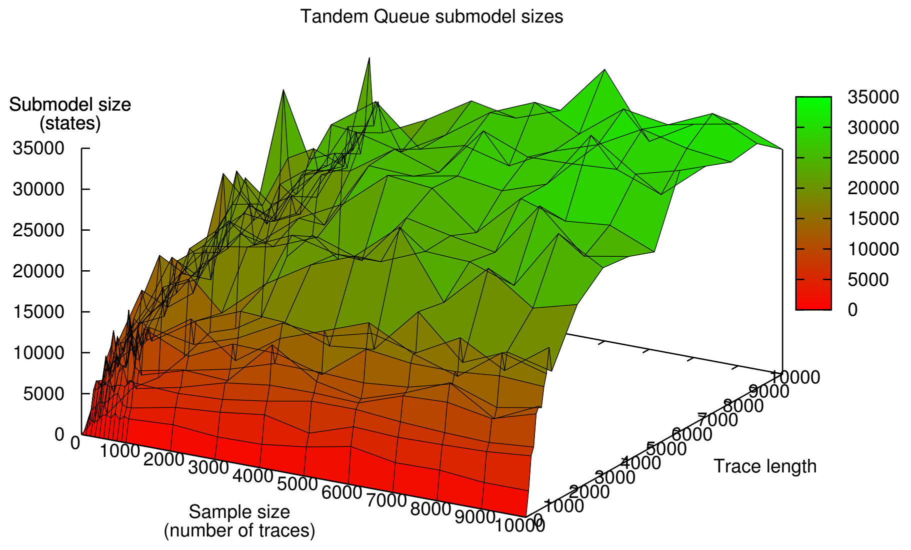
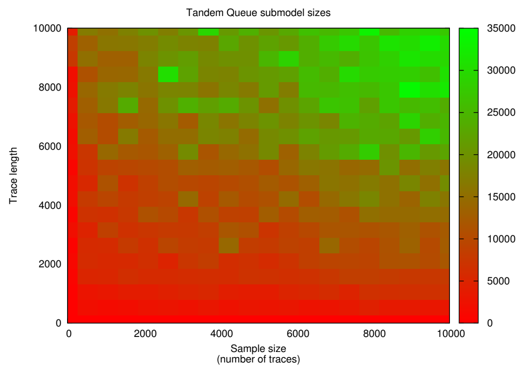


Fig. 6: Tandem Queue submodels sizes for different sample size and trace length parameters.

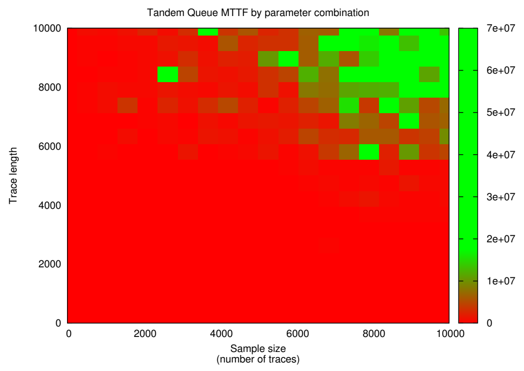
Table II: Selection of Tandem Queue submodel sizes and invariants for different parameter configurations.

Traces	Length	States	Invariant
10000	1	14	$cliC = cliM \wedge cliC \leq 0 \wedge state \leq 2 \wedge cliC \leq state$
10000	501	12690	$cliC \leq 73 \wedge cliM \leq 15 \wedge state \leq 9$
5000	1000	14134	$cliC \leq 69 \wedge cliM \leq 18 \wedge state \leq 9$
10000	1000	16086	$cliC \leq 83 \wedge cliM \leq 17 \wedge state \leq 9$
5000	2000	23388	$cliC \leq 100 \wedge cliM \leq 21 \wedge state \leq 9$
10000	2000	22486	$cliC \leq 92 \wedge cliM \leq 22 \wedge state \leq 9$
5000	3000	20932	$cliC \leq 98 \wedge cliM \leq 19 \wedge state \leq 9$
10000	3000	25228	$cliC \leq 108 \wedge cliM \leq 21 \wedge state \leq 9$
5000	4000	24538	$cliC \leq 105 \wedge cliM \leq 21 \wedge state \leq 9$
10000	4000	24882	$cliC \leq 94 \wedge cliM \leq 24 \wedge state \leq 9$
5000	5000	26424	$cliC \leq 104 \wedge cliM \leq 23 \wedge state \leq 9$
10000	5000	23686	$cliC \leq 97 \wedge cliM \leq 22 \wedge state \leq 9$
5000	6000	26182	$cliC \leq 99 \wedge cliM \leq 24 \wedge state \leq 9$
10000	6000	31902	$cliC \leq 121 \wedge cliM \leq 24 \wedge state \leq 9$
5000	7000	29926	$cliC \leq 123 \wedge cliM \leq 22 \wedge state \leq 9$
10000	7000	30674	$cliC \leq 121 \wedge cliM \leq 23 \wedge state \leq 9$
5000	8000	23910	$cliC \leq 107 \wedge cliM \leq 20 \wedge state \leq 9$
10000	8000	29424	$cliC \leq 116 \wedge cliM \leq 23 \wedge state \leq 9$
5000	9000	29924	$cliC \leq 118 \wedge cliM \leq 23 \wedge state \leq 9$
10000	9000	29926	$cliC \leq 123 \wedge cliM \leq 22 \wedge state \leq 9$
5000	10000	27174	$cliC \leq 107 \wedge cliM \leq 23 \wedge state \leq 9$
10000	10000	27460	$cliC \leq 100 \wedge cliM \leq 25 \wedge state \leq 9$

It can be observed that experiments with trace length below 3000 do not consistently produce rich enough models that yield good MTTF estimates. Unsurprisingly, small



(a) Submodel sizes by parameters



(b) Estimations by parameters

Fig. 7: Tandem queue submodel sizes and estimations for different parameters

sample sets are also inconsistent in their results. However, once the sample set size parameter is set to at least 6000 samples, the submodels produced consistently yield large MTTF estimates. In summary, for this case study a minimum of 6000 samples of traces at least 4000 steps long are necessary for consistent results. Furthermore, increasing these parameters does not yield clear advantage in terms of the final MTTF estimation. Both figures also show that results become more stable as these parameters are increased.

State space size alone is not the only important factor when evaluating the effectiveness of the approach. For a given size expressed in number of states, many submodels of that size exist, and not all of them may be effective. Preliminary work [Pavese et al. 2010] has shown that submodels obtained through depth first search (DFS) explorations yield very poor results, as they allow short traces to escape the submodel to the λ state. Although breadth first search (BFS) obtains higher MTTF lower bounds than DFS when used as a submodel generator, it performs poorly against our approach, as the state space that it explores is not as relevant. For example, our approach using 10000 traces 10000 states long (one of the best performers) obtains a 27460 state sized submodel, which is characterised by the invariant $cliC \leq 100 \wedge cliM \leq 25 \wedge state \leq 9$. Consider a similarly sized BFS generated submodel of 28000 states. The Tandem Queue model allows four different actions (push, fwd, svc$_1$,svc$_2$). Conservatively assuming at most two actions enabled at each state, an equal sized BFS submodel would explore at most $\lceil \log_2(27460) \rceil = 15$ levels deep. Such a submodel would only allow for very limited behaviour. If each transition level generated a new state, queues of no more than 15 elements could be generated by such a submodel. Of course, it is not always the case that a new state is generated. In fact, a BFS exploration that allows for 50 elements per queue results in a 32000 state submodel. The MTTF obtained through such a submodel is ~ 70000, very far from the results we obtain.

Regarding potential overhead of trace generation and invariant inference, memory consumption is negligible with respect to representing the state space of the full model, as only one relatively short trace needs to be kept in memory at a time. Time-wise, analysis of 10000 traces of length 10000 took less than an hour. Accounting for this hour in the verification time budget, the submodel that yielded the highest MTTF lower bound would have achieved a result of $\sim 6 \times 10^7$ in 23 hours, still a large increase against the estimation obtained via full model verification.

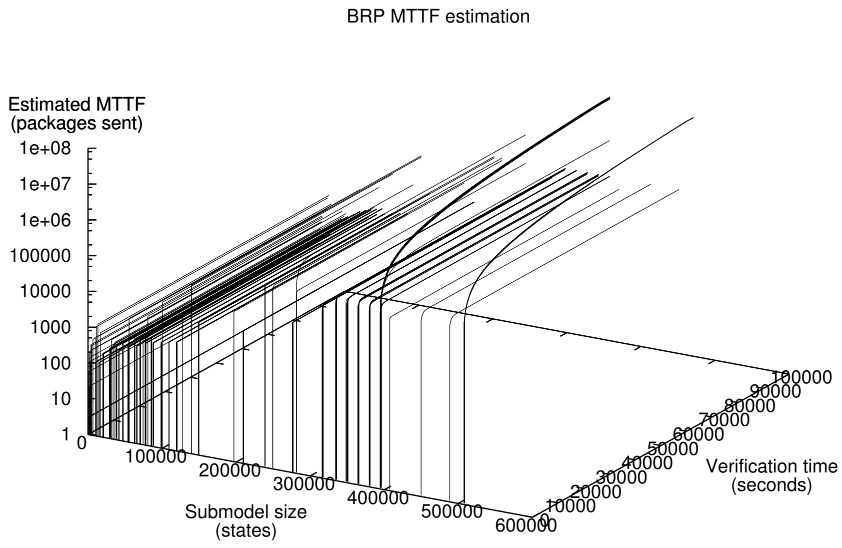


Fig. 8: Results of analysis of BRP (probabilistic file size choice) for different sized submodels, Backwards Gauss-Seidel method.

Table III: Selection of BRP submodel (probabilistic file size choice) sizes and invariants for different parameter configurations.

Traces	Length	States	Invariant
10000	1	35	$srep = nrtr \wedge srep = fileSize \wedge srep = r \wedge srep = rrep \wedge srep = k \wedge srep = l \wedge bs = s\_ab \wedge bs = fs \wedge bs = ls \wedge bs = fr \wedge bs = lr \wedge bs = br \wedge bs = r\_ab \wedge bs = recv \wedge s \leq 7 \wedge srep \leq 0 \wedge i \leq 1 \wedge s \geq srep \wedge s \geq i \wedge srep \leq i$
10000	501	66282	$s \leq 7 \wedge srep \leq 3 \wedge nrtr \leq 2 \wedge fileSize \leq 1500 \wedge i \leq 84 \wedge r \leq 4 \wedge rrep \leq 3 \wedge k \leq 2 \wedge l \leq 2 \wedge s \geq k \wedge s \geq l \wedge srep \leq fileSize \wedge srep \leq i \wedge srep \leq r \wedge srep \leq rrep \wedge nrtr \leq fileSize \wedge nrtr \leq i \wedge fileSize \geq r \wedge fileSize \geq rrep \wedge fileSize \geq k \wedge fileSize \geq l \wedge r \geq l$
10000	5000	333099	$s \leq 7 \wedge srep \leq 3 \wedge nrtr \leq 2 \wedge fileSize \leq 1500 \wedge i \leq 833 \wedge r \leq 4 \wedge rrep \leq 3 \wedge k \leq 2 \wedge l \leq 2 \wedge s \geq k \wedge s \geq l \wedge srep \leq fileSize \wedge srep \leq i \wedge srep \leq r \wedge srep \leq rrep \wedge nrtr \leq fileSize \wedge fileSize \geq r \wedge fileSize \geq rrep \wedge fileSize \geq k \wedge fileSize \geq l \wedge r \geq l$
10000	10000	392786	$s \leq 7 \wedge srep \leq 3 \wedge nrtr \leq 2 \wedge fileSize \leq 1500 \wedge i \leq 1500 \wedge r \leq 4 \wedge rrep \leq 3 \wedge k \leq 2 \wedge l \leq 2 \wedge s \geq k \wedge s \geq l \wedge srep \leq fileSize \wedge srep \leq i \wedge srep \leq r \wedge srep \leq rrep \wedge nrtr \leq fileSize \wedge nrtr \leq i \wedge fileSize \geq r \wedge fileSize \geq rrep \wedge fileSize \geq k \wedge fileSize \geq l \wedge r \geq l$

Although not intended to be shown to developers, we report on some of the automatically inferred invariants in Table II. The discovered invariants deal with bounding the size of both queues, while the variable $state$ encodes whether the queues are full or not, and the phase the system is in at the time. It is noteworthy that although it is intuitive that an invariant should bound the queue sizes, it is unlikely that a human would come up with the particular bounding values used.

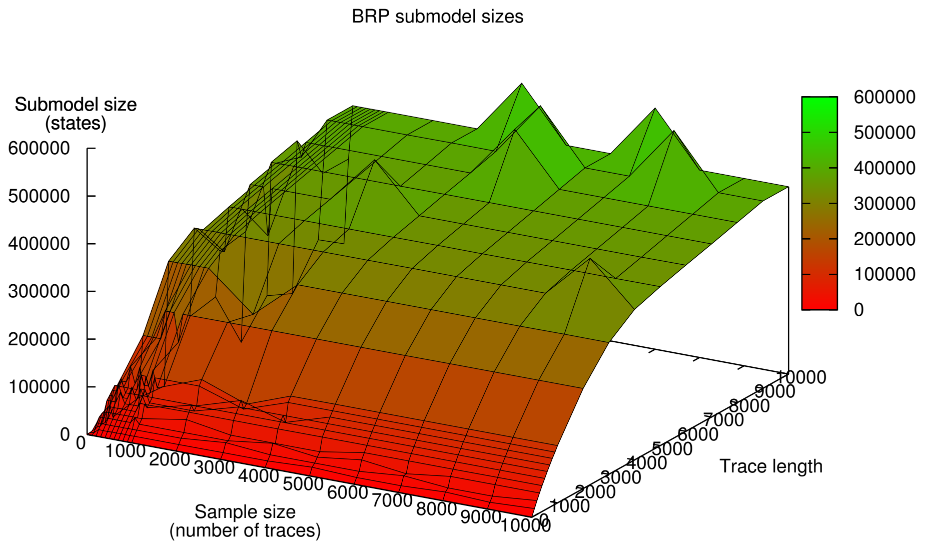
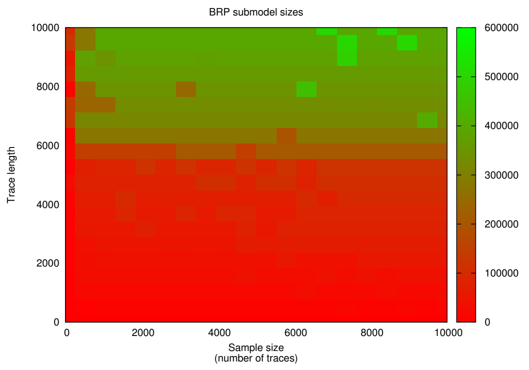


Fig. 9: BRP submodels (probabilistic file size choice) sizes for different sample size and trace length parameters.

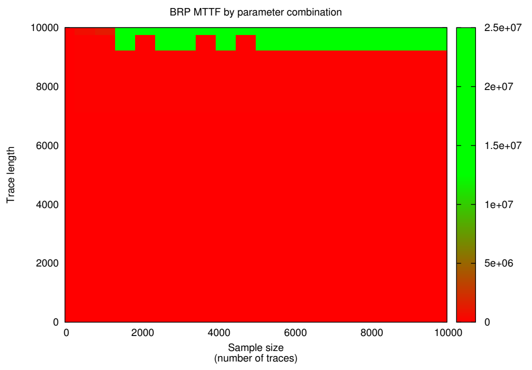
Bounded Retransmission Protocol - fully probabilistic model version. For the BRP case study in its fully probabilistic variation, similar results were obtained and are shown in Figures 8 and 9, and Table III.

In contrast to the prior case study, we were unable to obtain the MTTF for the full model due to state explosion that exhausted available memory. However, observations prior to running out of memory showed that the full model contains at least 30 million states, which means that the submodels we analysed represent at most 2% of the size of the full model, still a very low percentage. Furthermore, the highest MTTF bounds were obtained for submodels with a size starting from 400000 states (less than 1.33% of the full model), which turned out to yield an MTTF in the order of 2.5×10^7. This result is most significant, because of the impossibility of estimating MTTF for the full model.

Note that for submodels whose size is around the 400000 and 500000 states mark, there are both estimations that provide very good bounds and those that yield not so useful ones. Interestingly enough, those that do not perform well arise from submodels obtained through invariants inferred from sample sets where generated traces were shorter than 7000 states long, while sets of longer traces perform very well. This shows that appropriate trace length, as well as sample size, is critical to the final MTTF estimation. As before, a similarly sized submodel obtained through BFS does not provide such higher MTTF lower bounds. One of our best performers, at 10000 traces 10000 states long, produces a submodel 392786 states in size which (with eight BRP actions and conservatively assuming three enabled at any time) results in a BFS submodel of depth $\lceil \log_3(392786) \rceil = 12$, which models very few frames being sent. In fact, a BFS-like submodel that allows only for 5 frames to be sent per file comprises ~ 400000 states and yields an MTTF of only 40.



(a) Submodel sizes by parameters



(b) Estimations by parameters

Fig. 10: BRP submodel sizes and estimations for different parameters

The charts in Figure 7 expand on this. Figure 7a shows submodel sizes for different parameters, in a way similar to Figure 6. Figure 7b shows the estimated bounds for said parameters. It can easily be seen that submodel size and estimations correlate.

Figures 9 and 10 depict information related to the possibility of obtaining useful submodels, in a manner similar as we did for the Tandem Queue case study. It can be seen that it is quite easy to obtain such submodels, without many restrictions on experiment configuration. Again, there is a correlation between submodel size and estimation quality. However, model size is not as good an indicator in this case, since submodels obtained through simulations of lengths 7000 and 9000 are similar in size, but the resulting estimations are much better in the latter, even with as little as 2000 samples. Further increases of these parameters yield larger and slightly better-performing models, and this increase is much smoother (hence predictable) than is the case for the Tandem Queue submodels.

As in the other case study, trace generation and invariant inference incurs an overhead. In this case, since the model is more complex, this analysis can take up to 2 additional hours. Reducing the verification time by these 2 hours, the estimated MTTF would have been still large, about 2×10^7. Recall that this overhead was not included in measured time to allow graphs to show convergence speed of numerical analysis.

As before, we show for reference some of the inferred invariants. The variables $fileSize$, i and $nrtr$ describe the size of the file being sent, how many frames have been sent for that file, and the number of retries attempted, respectively. Other variables such as s_{ab}, r_{ab}, bs and fs encode the bit alternation in the protocol. The invariants obtained establish relationships between variables that seem unrelated, making them quite unintuitive even for a domain expert.

Bounded Retransmission Protocol - non-deterministic model version. As we explained before, we also developed a version of the BRP model that leaves the file size choice to a non-deterministic process. Recall that introducing non-determinism into a model requires a scheduler function to solve this non-determinism, and that we focus on those that yield the minimum and maximum probabilities or reward values. Therefore we turned our attention to finding out the minimum and maximum possible mean times to failure. We performed the same verifications as for the DTMC model, but effectively twice, as we require both extreme values. As was the case for the DTMC model, we were unable to obtain an estimation for the MTTF for the full model via probabilistic model checking, because of memory being exhausted due to state explo-

Table IV: Selection of BRP submodel (non-deterministic file size choice) MTTF evaluation results for different simulation parameter configurations.

Simulation for invariant inference and submodel generation			Model checking			
Traces	Length	States	Min. MTTF	Time	Max. MTTF	Time
7000	7000	362818	7012.95	416.03s	5189082.31	TO
7000	8000	486334	8009.98	310.63s	4131821.02	TO
7000	9000	392786	3562408.74	TO	6057239.73	TO
7000	10000	392786	3520812.45	TO	3836886.45	TO
8000	7000	467078	7012.98	311.74s	2696445.38	TO
8000	8000	377758	8009.93	283.59s	3436473.71	TO
8000	9000	393127	3228676.48	TO	4189020.09	TO
8000	10000	392786	3957488.92	TO	3985846.34	TO
9000	7000	363159	7012.98	166.43s	6672088.97	TO
9000	8000	377758	8009.93	284.88s	3988798.52	TO
9000	9000	392786	3099149.50	TO	3081547.39	TO
9000	10000	392786	3173533.40	TO	3414618.29	TO
10000	3000	276133	3007.00	63.38s	5779075.77	TO
10000	7000	362818	7012.95	192.54s	4006598.92	TO
10000	8000	486334	8009.98	344.00s	2910151.56	TO
10000	9000	393127	3979433.22	TO	4069637.98	TO
10000	10000	506672	3199985.39	TO	3149986.63	TO

sion. After the 24 hours of allotted time elapsed for each extreme value estimation, the results yielded a model comprising nearly 29 million states, while the reward estimation set a minimum MTTF value of 60297 and, surprisingly, a maximum MTTF of 50819. This discrepancy of the maximum estimation being actually less than the minimum one can be explained as an unintended consequence of the numerical verification procedure. The verification algorithm for extreme probabilities involves solving an optimisation problem for each extreme value. In the case of the minimum time to failure, the optimisation converges much faster. Indeed, the minimisation procedure actually performed about 20% more iterations than its maximisation counterpart, a factor that can explain this discrepancy.

After failing to obtain an exact value for the MTTF extreme values, we turned our attention to the estimation over partial explorations. We report on these experiments in Table IV, which summarises the results obtained while estimating the minimum and maximum MTTF.

It is interesting to note several things about these results. First, the submodels analysed represent, similarly to the fully probabilistic case, at most 2% of the size of the full model, a very low percentage. It also quickly becomes evident that there is a strange phenomenon taking place with the estimation of the minimum rewards. Almost all results are polarised either on the 3.25×10^6 - 4.0×10^6 range; or either in the 7000 - 8000 range. Further, the length of traces simulated is critical, particularly in the case of estimating the minimum MTTF. Note that simulating traces less than 9000 actions long, results in the smaller estimations for minimum MTTF. This seems to have its correlation with the invariants that were inferred in each case, depicted in Table V. Note that, in the invariants obtained with traces less than 9000 steps long, the variable i is restricted to no more than 1333. Recall that i indicates the number of packets of the file that have already been set. These invariants show that, for the traces analysed, some times the maximum file size (1500) was chosen, but never completely sent. For our approach, such situations would lie in the *unknown* set of the state space, and thus conservatively evaluated as failing states. However, invariants obtained for longer traces do allow i to reach its maximum of 1500, which explains the dramatic in-

Table V: Selection of BRP submodel (non-deterministic file size choice) sizes and invariants for different parameter configurations.

Traces	Length	Invariant
8000	8000	$s \leq 7 \wedge srep \leq 3 \wedge nrtr \leq 2 \wedge fileSize \leq 1500 \wedge i \leq 1333 \wedge r \leq 4 \wedge rrep \leq 3 \wedge k \leq 2 \wedge l \leq 2 \wedge s \geq k \wedge s \geq l \wedge srep \leq fileSize \wedge srep \leq i \wedge srep \leq r \wedge srep \leq rrep \wedge nrtr \leq fileSize \wedge nrtr \leq i \wedge fileSize \geq r \wedge fileSize \geq rrep \wedge fileSize \geq k \wedge fileSize \geq l \wedge r \geq l$
8000	10000	$s \leq 7 \wedge srep \leq 3 \wedge nrtr \leq 2 \wedge fileSize \leq 1500 \wedge i \leq 1500 \wedge r \leq 4 \wedge rrep \leq 3 \wedge k \leq 2 \wedge l \leq 2 \wedge s \geq k \wedge s \geq l \wedge srep \leq fileSize \wedge srep \leq i \wedge srep \leq r \wedge srep \leq rrep \wedge nrtr \leq fileSize \wedge nrtr \leq i \wedge fileSize \geq r \wedge fileSize \geq rrep \wedge fileSize \geq k \wedge fileSize \geq l \wedge r \geq l$
9000	8000	$s \leq 7 \wedge srep \leq 3 \wedge nrtr \leq 2 \wedge fileSize \leq 1500 \wedge i \leq 1333 \wedge r \leq 4 \wedge rrep \leq 3 \wedge k \leq 2 \wedge l \leq 2 \wedge s \geq k \wedge s \geq l \wedge srep \leq fileSize \wedge srep \leq i \wedge srep \leq r \wedge srep \leq rrep \wedge nrtr \leq fileSize \wedge nrtr \leq i \wedge fileSize \geq r \wedge fileSize \geq rrep \wedge fileSize \geq k \wedge fileSize \geq l \wedge r \geq l$
9000	10000	$s \leq 7 \wedge srep \leq 3 \wedge nrtr \leq 2 \wedge fileSize \leq 1500 \wedge i \leq 1500 \wedge r \leq 4 \wedge rrep \leq 3 \wedge k \leq 2 \wedge l \leq 2 \wedge s \geq k \wedge s \geq l \wedge srep \leq fileSize \wedge srep \leq i \wedge srep \leq r \wedge srep \leq rrep \wedge nrtr \leq fileSize \wedge nrtr \leq i \wedge fileSize \geq r \wedge fileSize \geq rrep \wedge fileSize \geq k \wedge fileSize \geq l \wedge r \geq l$
10000	8000	$s \leq 7 \wedge srep \leq 3 \wedge nrtr \leq 3 \wedge fileSize \leq 1500 \wedge i \leq 1333 \wedge r \leq 4 \wedge rrep \leq 3 \wedge k \leq 2 \wedge l \leq 2 \wedge s \geq k \wedge s \geq l \wedge srep \leq fileSize \wedge srep \leq i \wedge srep \leq r \wedge srep \leq rrep \wedge nrtr \leq fileSize \wedge nrtr \leq i \wedge fileSize \geq r \wedge fileSize \geq rrep \wedge fileSize \geq k \wedge fileSize \geq l \wedge r \geq l$
10000	10000	$s \leq 7 \wedge srep \leq 3 \wedge nrtr \leq 3 \wedge fileSize \leq 1500 \wedge i \leq 1500 \wedge r \leq 4 \wedge rrep \leq 3 \wedge k \leq 2 \wedge l \leq 2 \wedge s \geq k \wedge s \geq l \wedge srep \leq fileSize \wedge srep \leq i \wedge srep \leq r \wedge srep \leq rrep \wedge nrtr \leq fileSize \wedge fileSize \geq r \wedge fileSize \geq rrep \wedge fileSize \geq k \wedge fileSize \geq l \wedge r \geq l$

crease of the estimations. Even more, increasing the simulation length to 10000 actions does pay off in some cases, although the increase is not nearly as dramatic.

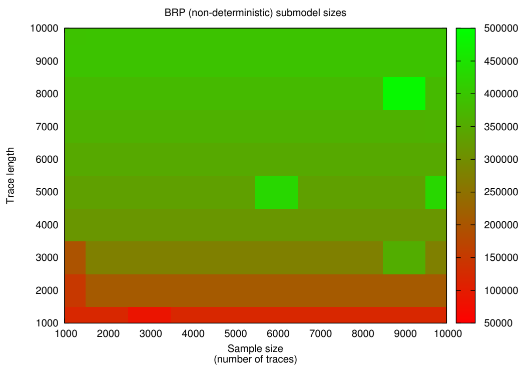
In the case of the maximum MTTF estimation, all submodels behave more or less uniformly. Although there are of course differences, even the worst results are still much better than those obtained by full model evaluation. In fact, when compared with the result obtained for full model estimation, it can clearly be seen that estimation over submodels pays off – the maximum MTTF estimated for submodels is, in all cases, at least 50 times as much than those obtained for the full model.

Figure 11 illustrates these results in a manner similar to the previous case studies. Note that while there are two different charts for minimum and maximum MTTF estimations, the submodels remain the same in both cases.

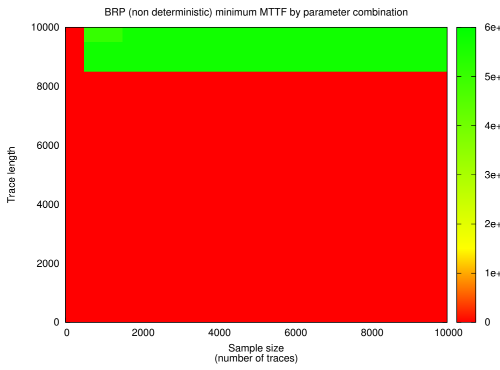
There is a final point that needs to be noted. As we discussed earlier, the submodels obtained by analysing shorter simulations are not very good for minimum MTTF estimation. However, they are the best performers for estimating *maximum* MTTF. This is a consequence of the state space being smaller, as this allows for more numerical iterations in the same time budget. Another important factor is that choosing a smaller file size allows for a larger Mean Time to Failure. This is because when transmitting a smaller file, the chance that the protocol will deplete its allowed retries is smaller than with a bigger file, simply because it has less chances to fail. This contrasts with the minimum MTTF calculation, which becomes larger just as bigger files are allowed in the model.

WLAN collision avoidance protocol. Finally, we turn our attention to the analysis of the WLAN collision protocol model. In this case study, we are interested in estimating the turnaround time (TAT) for both emitting stations to complete sending their intended data. That is, we wish to know the mean time from the moment the first station intends to send data until both of them have successfully sent their data, including all necessary backoff time.

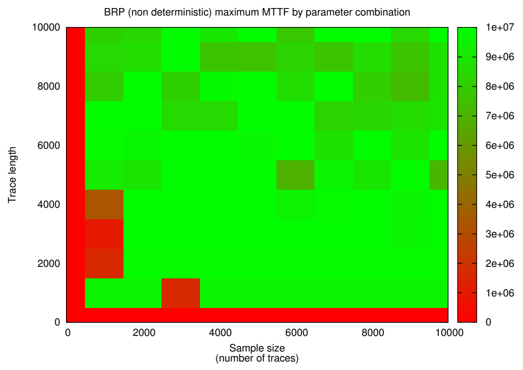
For this case study we also attempted to produce an estimate for the full model. Contrasting with the previous case studies, the event under analysis is not a rare event at all. On the contrary, it is desirable that in every instance both stations are able to send their data in a reasonable time. During this analysis, we obtained a full model comprising about 75 million states. The minimum TAT was estimated at 1725 after



(a) Submodel sizes by parameters



(b) Minimum estimations by parameters



(c) Maximum estimations by parameters

Fig. 11: BRP (non deterministic) submodel sizes and estimations for different parameters

executing for just 10 minutes, while the maximum one was calculated to be 4301.65, after 15 hours into the verification process execution. Turnaround time is measured in microseconds (μs).

Again, we compared this performance with our approach, with the results obtained depicted in Tables VI and VII.

In this case, the results are much easier to interpret. First, note the TAT estimations themselves from Table VI. The minimum turnaround time estimated is the same for all submodels evaluated and coincides with that obtained through the full model evaluation. In the case of the maximum turnaround estimation, they are not all the same, but they are all around the same value, and differ in no more than 2% from the actual value estimated through full model evaluation. However, the verification times that were necessary for estimating these results are what are most significant. For every submodel, both for the minimum TAT estimation as for the maximum, all reward estimations finished in less than 10 seconds, with most of those estimations taking much less time. This marks a stark contrast with the time needed for the full model verification: minimum TAT estimation required no more than 1 second in all cases (except one where it required 4.5 seconds) while full model estimation required 10 minutes. In the case of maximum reward estimation, doing so for the full model required 15 hours, while partial model estimations were mostly completed within 1.5 seconds, except in

Table VI: Selection of WLAN submodel TAT evaluation results for different simulation parameter configurations.

Simulation for invariant inference and submodel generation			Model checking			
Traces	Length	States	Min. TAT	Time	Max. TAT	Time
500	100	117976	1725.00	0.72s	4213.90	1.15s
500	150	117976	1725.00	0.74s	4213.90	1.28s
500	200	118252	1725.00	0.76s	4278.24	1.16s
500	250	118252	1725.00	0.80s	4278.24	1.24s
1000	100	117976	1725.00	0.80s	4213.90	1.33s
1000	150	118252	1725.00	0.73s	4278.24	1.12s
1000	200	118232	1725.00	0.74s	4278.24	1.12s
1000	250	118252	1725.00	0.71s	4278.24	1.21s
1500	100	118104	1725.00	0.69s	4246.08	1.07s
1500	150	118252	1725.00	0.75s	4278.23	1.14s
1500	200	118252	1725.00	0.78s	4278.23	1.19s
1500	250	472809	1725.00	4.47s	4286.21	7.87s
2000	100	118240	1725.00	0.70s	4247.08	1.18s
2000	150	118252	1725.00	0.71s	4278.23	1.11s
2000	200	118252	1725.00	0.70s	4278.23	1.11s
2000	250	118232	1725.00	0.94s	4278.24	1.34s

one case where it required a time short of 8 seconds. Further, the estimated maximum turnaround times are very close to the actual value, deviating at most a 0.02% from the actual value, and most estimations staying within 0.005% of this actual result.

The size of the submodels evaluated is also striking. In all cases, this size is about 0.15% to 0.50% of the size of the whole model. This seems to suggest that the full model has a very large portion of behaviour that is largely irrelevant in regards to their actual contribution to the system's TAT. In fact, it is easy to see from Table VII that although the waiting slots ($slot1$ and $slot2$) can be increased to as much as 128 different slots, the simulations only observed waiting times up to 4 of these slots. Since the slot is chosen equiprobably within the same backoff level, this seems to suggest that only the first two backoff levels were taken on all of the simulated executions. In other words, it was never necessary to increase the backoff to more than this second level.

As in the previous case study, the choice of parameters for the number of traces to simulate and the length of the simulated paths also plays a role. However, this is not as clear-cut as in the previous case. Note that the size of the submodels evaluated seems to lie either near the 120000 state mark except for one that lies near the 460000 state mark, yielding a partial state space that is roughly 4 times as large as the others. This also explains the discrepancy on the estimation times. When the larger submodel was analysed, the estimation took nearly 7 times as much time as the other estimations. The estimated values, however, were not much better than the ones estimated over smaller partial state spaces. They all yielded an estimate equal to the actual value in the case of the minimum turnaround time. In the case of the maximum turnaround time, estimations over the larger partial state space did produce a value that is closer to the actual value than the other estimations, but this difference is only marginal.

We may, however, find an explanation for such a disparity in the invariants inferred– see Table VII. In the cases where a bigger submodel was generated, it turns out that the second sender station was allowed to take the slot number 3 in some of the executions, while in the smaller ones it never did. Since the choice of slot is uniform, and whenever the slot 2 is available the slot 3 also is, we can only conclude that these differences are only a coincidental artefact of the stochasticity of the sampling procedure.

Virus Infection. Finally, we study the network virus infection scenario. As we described earlier, this network has a cubic grid topology. For these experiments we chose

Table VII: Selected WLAN inferred invariants for different parameter configurations.

Traces	Length	Invariant
1000	100	$col \leq 2 \wedge c1 \leq 2 \wedge c2 \leq 2 \wedge x1 \leq 10 \wedge s1 \leq 12 \wedge s1 \geq 1 \wedge slot1 \leq 1 \wedge backoff1 \leq 31 \wedge bc1 \leq 2 \wedge x2 \leq 10 \wedge s2 \leq 12 \wedge s2 \geq 1 \wedge slot2 \leq 1 \wedge backoff2 \leq 31 \wedge bc2 \leq 2 \wedge col \leq s1 \wedge col \geq slot1 \wedge col \leq s2 \wedge col \geq slot2 \wedge c1 < s1 \wedge c1 \leq s2 \wedge c2 \leq s1 \wedge c2 < s2 \wedge s1 > slot1 \wedge s1 > bc1 \wedge s1 > slot2 \wedge s1 > bc2 \wedge slot1 \leq bc1 \wedge slot1 < s2 \wedge slot1 \leq bc2 \wedge bc1 < s2 \wedge bc1 \geq slot2 \wedge s2 > slot2 \wedge s2 > bc2 \wedge slot2 \leq bc2$
1000	150	$col \leq 2 \wedge c1 \leq 2 \wedge c2 \leq 2 \wedge x1 \leq 10 \wedge s1 \leq 12 \wedge s1 \geq 1 \wedge slot1 \leq 1 \wedge backoff1 \leq 31 \wedge bc1 \leq 2 \wedge x2 \leq 10 \wedge s2 \leq 12 \wedge s2 \geq 1 \wedge slot2 \leq 1 \wedge backoff2 \leq 31 \wedge bc2 \leq 2 \wedge col \leq s1 \wedge col \geq slot1 \wedge col \leq s2 \wedge col \geq slot2 \wedge c1 < s1 \wedge c1 \leq s2 \wedge c2 \leq s1 \wedge c2 < s2 \wedge s1 > slot1 \wedge s1 > bc1 \wedge s1 > slot2 \wedge s1 > bc2 \wedge slot1 \leq bc1 \wedge slot1 < s2 \wedge bc1 \leq s2 \wedge s2 > slot2 \wedge s2 > bc2 \wedge slot2 \leq bc2$
1000	200	$col \leq 2 \wedge c1 \leq 2 \wedge c2 \leq 2 \wedge x1 \leq 10 \wedge s1 \leq 12 \wedge s1 \geq 1 \wedge slot1 \leq 1 \wedge backoff1 \leq 31 \wedge bc1 \leq 2 \wedge x2 \leq 10 \wedge s2 \leq 12 \wedge s2 \geq 1 \wedge slot2 \leq 1 \wedge backoff2 \leq 31 \wedge bc2 \leq 2 \wedge col \leq s1 \wedge col \geq slot1 \wedge col \leq s2 \wedge col \geq slot2 \wedge c1 < s1 \wedge c1 \leq s2 \wedge c2 \leq s1 \wedge c2 < s2 \wedge s1 > slot1 \wedge s1 > bc1 \wedge s1 > slot2 \wedge s1 > bc2 \wedge slot1 \leq bc1 \wedge slot1 < s2 \wedge bc1 < s2 \wedge s2 > slot2 \wedge s2 > bc2 \wedge slot2 \leq bc2$
1000	250	$col \leq 2 \wedge c1 \leq 2 \wedge c2 \leq 2 \wedge x1 \leq 10 \wedge s1 \leq 12 \wedge s1 \geq 1 \wedge slot1 \leq 1 \wedge backoff1 \leq 31 \wedge bc1 \leq 2 \wedge x2 \leq 10 \wedge s2 \leq 12 \wedge s2 \geq 1 \wedge slot2 \leq 1 \wedge backoff2 \leq 31 \wedge bc2 \leq 2 \wedge col \leq s1 \wedge col \geq slot1 \wedge col \leq s2 \wedge col \geq slot2 \wedge c1 < s1 \wedge c1 \leq s2 \wedge c2 \leq s1 \wedge c2 < s2 \wedge s1 > slot1 \wedge s1 > bc1 \wedge s1 > slot2 \wedge s1 \geq bc2 \wedge slot1 \leq bc1 \wedge slot1 < s2 \wedge bc1 < s2 \wedge s2 > slot2 \wedge s2 > bc2 \wedge slot2 \leq bc2$
1500	100	$col \leq 2 \wedge c1 \leq 2 \wedge c2 \leq 2 \wedge x1 \leq 10 \wedge s1 \leq 12 \wedge s1 \geq 1 \wedge slot1 \leq 1 \wedge backoff1 \leq 31 \wedge bc1 \leq 2 \wedge x2 \leq 10 \wedge s2 \leq 12 \wedge s2 \geq 1 \wedge slot2 \leq 1 \wedge backoff2 \leq 31 \wedge bc2 \leq 2 \wedge col \leq s1 \wedge col \geq slot1 \wedge col \leq s2 \wedge col \geq slot2 \wedge c1 < s1 \wedge c1 \leq s2 \wedge c2 \leq s1 \wedge c2 < s2 \wedge s1 > slot1 \wedge s1 > bc1 \wedge s1 > slot2 \wedge s1 > bc2 \wedge slot1 \leq bc1 \wedge slot1 < s2 \wedge bc1 < s2 \wedge bc1 \geq slot2 \wedge s2 > slot2 \wedge s2 > bc2 \wedge slot2 \leq bc2$
1500	150	$col \leq 2 \wedge c1 \leq 2 \wedge c2 \leq 2 \wedge x1 \leq 10 \wedge s1 \leq 12 \wedge s1 \geq 1 \wedge slot1 \leq 1 \wedge backoff1 \leq 31 \wedge bc1 \leq 2 \wedge x2 \leq 10 \wedge s2 \leq 12 \wedge s2 \geq 1 \wedge slot2 \leq 1 \wedge backoff2 \leq 31 \wedge bc2 \leq 2 \wedge col \leq s1 \wedge col \geq slot1 \wedge col \leq s2 \wedge col \geq slot2 \wedge c1 < s1 \wedge c1 \leq s2 \wedge c2 \leq s1 \wedge c2 < s2 \wedge s1 > slot1 \wedge s1 > bc1 \wedge s1 > slot2 \wedge s1 > bc2 \wedge slot1 \leq bc1 \wedge slot1 < s2 \wedge bc1 \leq s2 \wedge s2 > slot2 \wedge s2 > bc2 \wedge slot2 \leq bc2$
1500	200	$col \leq 2 \wedge c1 \leq 2 \wedge c2 \leq 2 \wedge x1 \leq 10 \wedge s1 \leq 12 \wedge s1 \geq 1 \wedge slot1 \leq 1 \wedge backoff1 \leq 31 \wedge bc1 \leq 2 \wedge x2 \leq 10 \wedge s2 \leq 12 \wedge s2 \geq 1 \wedge slot2 \leq 1 \wedge backoff2 \leq 31 \wedge bc2 \leq 2 \wedge col \leq s1 \wedge col \geq slot1 \wedge col \leq s2 \wedge col \geq slot2 \wedge c1 < s1 \wedge c1 \leq s2 \wedge c2 \leq s1 \wedge c2 < s2 \wedge s1 > slot1 \wedge s1 > bc1 \wedge s1 > slot2 \wedge s1 \geq bc2 \wedge slot1 \leq bc1 \wedge slot1 < s2 \wedge bc1 < s2 \wedge s2 > slot2 \wedge s2 > bc2 \wedge slot2 \leq bc2$
1500	250	$col \leq 2 \wedge c1 \leq 2 \wedge c2 \leq 2 \wedge x1 \leq 10 \wedge s1 \leq 12 \wedge s1 \geq 1 \wedge slot1 \leq 3 \wedge backoff1 \leq 31 \wedge bc1 \leq 3 \wedge x2 \leq 10 \wedge s2 \leq 12 \wedge s2 \geq 1 \wedge slot2 \leq 1 \wedge backoff2 \leq 31 \wedge bc2 \leq 3 \wedge col \leq s1 \wedge col \leq s2 \wedge col \geq slot2 \wedge c1 < s1 \wedge c1 \leq s2 \wedge c2 \leq s1 \wedge c2 < s2 \wedge s1 > slot1 \wedge s1 \geq bc1 \wedge s1 > slot2 \wedge s1 \geq bc2 \wedge slot1 \leq bc1 \wedge slot1 < s2 \wedge s2 > slot2 \wedge s2 \geq bc2 \wedge slot2 \leq bc2$
2000	100	$col \leq 2 \wedge c1 \leq 2 \wedge c2 \leq 2 \wedge x1 \leq 10 \wedge s1 \leq 12 \wedge s1 \geq 1 \wedge slot1 \leq 1 \wedge backoff1 \leq 31 \wedge bc1 \leq 2 \wedge x2 \leq 10 \wedge s2 \leq 12 \wedge s2 \geq 1 \wedge slot2 \leq 1 \wedge backoff2 \leq 31 \wedge bc2 \leq 2 \wedge col \leq s1 \wedge col \geq slot1 \wedge col \leq s2 \wedge col \geq slot2 \wedge c1 < s1 \wedge c1 \leq s2 \wedge c2 \leq s1 \wedge c2 < s2 \wedge s1 > slot1 \wedge s1 > bc1 \wedge s1 > slot2 \wedge s1 \geq bc2 \wedge slot1 \leq bc1 \wedge slot1 < s2 \wedge slot1 \leq bc2 \wedge bc1 \leq s2 \wedge s2 > slot2 \wedge s2 > bc2 \wedge slot2 \leq bc2$
2000	150	$col \leq 2 \wedge c1 \leq 2 \wedge c2 \leq 2 \wedge x1 \leq 10 \wedge s1 \leq 12 \wedge s1 \geq 1 \wedge slot1 \leq 1 \wedge backoff1 \leq 31 \wedge bc1 \leq 2 \wedge x2 \leq 10 \wedge s2 \leq 12 \wedge s2 \geq 1 \wedge slot2 \leq 1 \wedge backoff2 \leq 31 \wedge bc2 \leq 2 \wedge col \leq s1 \wedge col \geq slot1 \wedge col \leq s2 \wedge col \geq slot2 \wedge c1 < s1 \wedge c1 \leq s2 \wedge c2 \leq s1 \wedge c2 < s2 \wedge s1 > slot1 \wedge s1 > bc1 \wedge s1 > slot2 \wedge s1 > bc2 \wedge slot1 \leq bc1 \wedge slot1 < s2 \wedge bc1 \leq s2 \wedge s2 > slot2 \wedge s2 > bc2 \wedge slot2 \leq bc2$
2000	200	$col \leq 2 \wedge c1 \leq 2 \wedge c2 \leq 2 \wedge x1 \leq 10 \wedge s1 \leq 12 \wedge s1 \geq 1 \wedge slot1 \leq 1 \wedge backoff1 \leq 31 \wedge bc1 \leq 2 \wedge x2 \leq 10 \wedge s2 \leq 12 \wedge s2 \geq 1 \wedge slot2 \leq 1 \wedge backoff2 \leq 31 \wedge bc2 \leq 2 \wedge col \leq s1 \wedge col \geq slot1 \wedge col \leq s2 \wedge col \geq slot2 \wedge c1 < s1 \wedge c1 \leq s2 \wedge c2 \leq s1 \wedge c2 < s2 \wedge s1 > slot1 \wedge s1 > bc1 \wedge s1 > slot2 \wedge s1 > bc2 \wedge slot1 \leq bc1 \wedge slot1 < s2 \wedge bc1 \leq s2 \wedge s2 > slot2 \wedge s2 > bc2 \wedge slot2 \leq bc2$
2000	250	$col \leq 2 \wedge c1 \leq 2 \wedge c2 \leq 2 \wedge x1 \leq 10 \wedge s1 \leq 12 \wedge s1 \geq 1 \wedge slot1 \leq 1 \wedge backoff1 \leq 31 \wedge bc1 \leq 2 \wedge x2 \leq 10 \wedge s2 \leq 12 \wedge s2 \geq 1 \wedge slot2 \leq 1 \wedge backoff2 \leq 31 \wedge bc2 \leq 2 \wedge col \leq s1 \wedge col \geq slot1 \wedge col \leq s2 \wedge col \geq slot2 \wedge c1 < s1 \wedge c1 \leq s2 \wedge c2 \leq s1 \wedge c2 < s2 \wedge s1 > slot1 \wedge s1 > bc1 \wedge s1 > slot2 \wedge s1 > bc2 \wedge slot1 \leq bc1 \wedge slot1 < s2 \wedge bc1 < s2 \wedge s2 > slot2 \wedge s2 > bc2 \wedge slot2 \leq bc2$

to set the number N of nodes per edge to be 3; that is, the network is comprised of a total of 27 nodes. This is more than enough to quickly deplete all available memory before reaching a full state space. The total potential state space is $3^{27} \sim 7 \times 10^{12}$ states. The actual reachable states are less. For example, a state where every node has its firewall down is unreachable (there should be at least one infected node responsible for having broken the firewall of the last node). However, the reachable states are still enough to make a complete analysis infeasible.

This is a similar situation to that of the BRP case study, so we focus on partial explorations only. We will show, however, that in this case we have a way of computing the values of interest in an analytical manner.

We start out with a non-deterministic model of the network, since we do not know which distribution (if any) governs the races between the different nodes. At any given point any of the nodes can choose to perform its action. However, the internal behaviour of each node is modelled probabilistically.

According to the behaviour we modelled, the nodes are quite resistant to attack. An infected node has a 0.01 chance to break a neighbour's firewall. Once this firewall is down, it has a further 0.01 chance to infect it. A healthy node is much more efficient and has a 0.98 chance of repair success. However, all nodes are agnostic respect the status of their neighbours. This means that an infected node may attempt to reinfect an already infected node, and a healthy node may attempt to repair a non-infected one.

The first property of interest is the minimum expected time to total infection of the network. The maximum expected time is uninteresting, as it is infinite: a scheduler may choose to alternatively infect a node, and once it is infected, have a neighbour repair it, and do so indefinitely. Therefore, there exist valid schedulers that avoid attaining total infection. Following the same reasoning, we also wish to calculate the minimum expected time to propagating infection from one corner of the cubic grid to the opposite corner, without requiring full infection

Even though we cannot perform a complete model check over the whole system, we can calculate the values of the interesting properties in an analytical manner.

For the first property, the fastest way to achieve total infection is to infect each of the remaining 26 nodes, without allowing for any recovery from the healthy nodes. Recall that infection of a node implies first lowering its firewall. Since the probability of breaking the firewall and infecting a vulnerable node is the same (0.01), the previous analysis amounts to studying a Negative Binomial distribution with parameter 0.99. In order to witness total infection, we need to see 52 (26 firewall breaks + 26 infections) failure events. Therefore the expected time to total infection is $52/0.01 = 5200$.

The case for corner infection is similar. We can calculate the mean time to corner infection, since the worst scheduler is the one that takes the fastest vector of infection from one corner to another. This involves infecting just 6 nodes to reach the opposite corner. The expected time to corner infection follows the same distribution as before. Following this known distribution, it turns out that the expected time to infection of the opposite corner is 1200.

Partial exploration approach results. As we did with the other case studies, we put our approach to the test. Although we managed to obtain correct results, in this case the values obtained turned out to lie far from the actual values. Using our standard simulation parameters of simulating 1000-10000 traces of 1000-10000 steps each, we always obtained submodels for which the bound to mean time of both total infection and corner infection was ~ 200.

These results are a consequence of the simulated traces not capturing enough of the system's behaviour. This is caused, in turn, by the strongly non-deterministic nature of the model. It happens that, at any given point in simulation, there exist several possible actions to take. Namely, since each node is unaware of its neighbours status, each node can try to break or infect its neighbours (if itself is infected), or repair it (if it is not infected). At each point, there are in excess of 27 choices possible, each with a simulation probability of $1/27 = 0.03737$. This makes it extremely unlikely that a simulation will even infect 2 nodes. In fact, the probability of a simulation immediately infecting two nodes is $(0.03737 \times 0.01)^4 = 1.95 \times 10^{-14}$. Even taking into account that a simulation can take up to 10000 steps, the probability still remains extremely small.

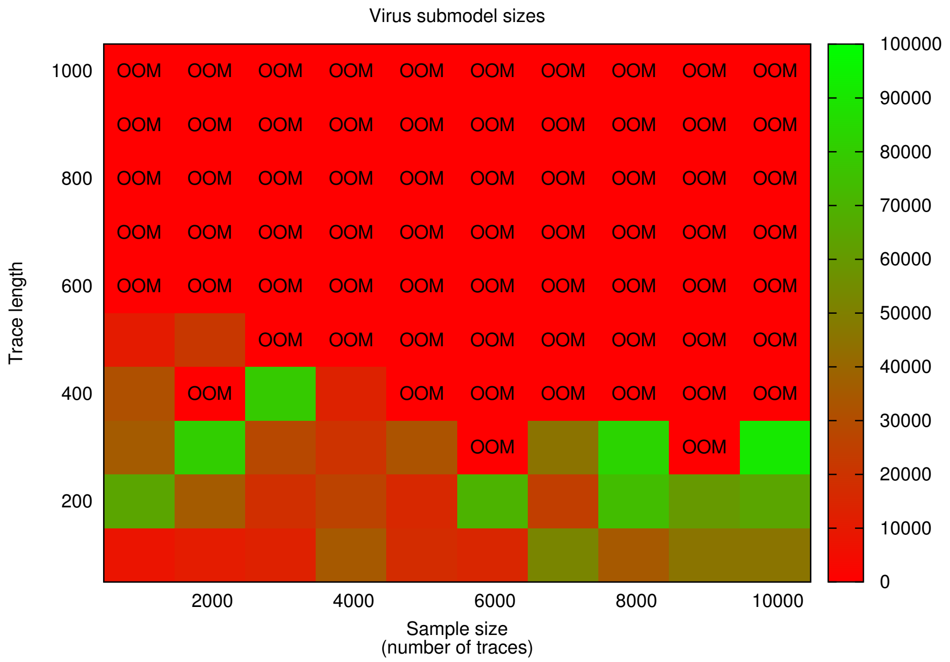


Fig. 12: Sizes of submodels of the Virus infection model for different simulation parameters. OOM denotes submodels that exceeded available memory.

This results in submodels that describe very little behaviour. However, the results are still correct, although arguably not as useful as in the other cases. In any case, recall that obtaining results based on a full exploration is infeasible and, as we will see later, Monte Carlo approaches do not help either.

In order to be able to perform a more meaningful analysis, we modelled a second version of the virus infection where we restricted some behaviour. This second model introduces two changes. First, the nodes do not perform repair operations. Therefore, once a node is infected, it stays infected. Second, nodes are aware of their neighbours status. As a result, infected nodes do not try to break broken neighbours, and do not try to infect infected neighbours. These two changes significantly constrain the model, and reduce both the number of reachable states as well as available transitions. Interestingly enough, the analytical results for the extreme case still hold the same values, as the analysis is still valid under this constrained model.

From initial experimentation it was clear that running simulations as long as those we performed for the previous case studies yielded submodels that were still large enough to be infeasible to analyse. Therefore, we reduced the length of simulations for this case study. The results we present in this section were obtained by performing simulations where the number of traces varied between 1000 and 10000 (stepping size by 1000), and the traces were between 100 and 1000 steps long (stepping size 100). Even with this model simplification and simulation parameters adjustment, we also ran into cases where memory was not enough to hold the submodel. Figure 12 shows these results.

As a result we only report results on those submodels that we could analyse. Figures 13 and 14 show the bounds on minimum expected time to total and corner infection, respectively, along with the time taken to arrive to those results. The results are

Table VIII: Selection of virus infection submodel sizes and invariants for different parameter configurations.

Traces	Length	States	Invariant
1000	100	7728	$s123 = s132 \wedge s123 = s133 \wedge s123 = s213 \wedge s123 = s222 \wedge s123 = s223 \wedge s123 = s231 \wedge s123 = s232 \wedge s123 = s233 \wedge s123 = s312 \wedge s123 = s313 \wedge s123 = s321 \wedge s123 = s322 \wedge s123 = s323 \wedge s123 = s331 \wedge s123 = s332 \wedge s123 = s333 \wedge s111 \leq 2 \wedge s111 \geq 2 \wedge \mathit{true} \wedge s112 \leq 2 \wedge s113 \leq 1 \wedge s121 \leq 2 \wedge s122 \leq 1 \wedge s123 \leq 0 \wedge s131 \leq 1 \wedge s211 \leq 2 \wedge s212 \leq 2 \wedge s221 \leq 1 \wedge s311 \leq 1 \wedge s111 \geq s112 \wedge s111 > s113 \wedge s111 \geq s121 \wedge s111 > s122 \wedge s111 > s123 \wedge \mathit{true} \wedge s111 > s131 \wedge s111 \geq s211 \wedge s111 \geq s212 \wedge s111 > s221 \wedge s111 > s311 \wedge s112 \geq s113 \wedge s112 \geq s122 \wedge s112 \geq s123 \wedge s113 \geq s123 \wedge s121 \geq s123 \wedge s121 \geq s131 \wedge s122 \geq s123 \wedge s123 \leq s131 \wedge s123 \leq s211 \wedge s123 \leq s212 \wedge s123 \leq s221 \wedge s123 \leq s311 \wedge s211 \geq s311$
5000	100	17378	$s123 = s132 \wedge s123 = s133 \wedge s123 = s213 \wedge s123 = s222 \wedge s123 = s223 \wedge s123 = s231 \wedge s123 = s232 \wedge s123 = s233 \wedge s123 = s312 \wedge s123 = s313 \wedge s123 = s321 \wedge s123 = s322 \wedge s123 = s323 \wedge s123 = s331 \wedge s123 = s332 \wedge s123 = s333 \wedge s111 \leq 2 \wedge s111 \geq 2 \wedge \mathit{true} \wedge s112 \leq 2 \wedge s113 \leq 2 \wedge s121 \leq 2 \wedge s122 \leq 1 \wedge s123 \leq 0 \wedge s131 \leq 1 \wedge s211 \leq 2 \wedge s212 \leq 2 \wedge s221 \leq 2 \wedge s311 \leq 1 \wedge s111 \geq s112 \wedge s111 \geq s113 \wedge s111 \geq s121 \wedge s111 > s122 \wedge s111 > s123 \wedge \mathit{true} \wedge s111 > s131 \wedge s111 \geq s211 \wedge s111 \geq s212 \wedge s111 \geq s221 \wedge s111 > s311 \wedge s112 \geq s113 \wedge s112 \geq s123 \wedge s113 \geq s123 \wedge s121 \geq s123 \wedge s121 \geq s131 \wedge s122 \geq s123 \wedge s123 \leq s131 \wedge s123 \leq s211 \wedge s123 \leq s212 \wedge s123 \leq s221 \wedge s123 \leq s311 \wedge s211 \geq s311$
1000	400	3128661	$s133 = s223 \wedge s133 = s233 \wedge s133 = s313 \wedge s133 = s323 \wedge s133 = s331 \wedge s133 = s332 \wedge s133 = s333 \wedge s111 \leq 2 \wedge s111 \geq 2 \wedge \mathit{true} \wedge s112 \leq 2 \wedge s113 \leq 2 \wedge s121 \leq 2 \wedge s122 \leq 2 \wedge s123 \leq 1 \wedge s131 \leq 2 \wedge s132 \leq 2 \wedge s133 \leq 0 \wedge s211 \leq 2 \wedge s212 \leq 2 \wedge s213 \leq 1 \wedge s221 \leq 2 \wedge s222 \leq 2 \wedge s231 \leq 1 \wedge s232 \leq 1 \wedge s311 \leq 2 \wedge s312 \leq 1 \wedge s321 \leq 1 \wedge s322 \leq 1 \wedge s111 \geq s112 \wedge s111 \geq s113 \wedge s111 \geq s121 \wedge s111 \geq s122 \wedge s111 > s123 \wedge s111 \geq s131 \wedge s111 \geq s132 \wedge s111 > s133 \wedge \mathit{true} \wedge s111 \geq s211 \wedge s111 \geq s212 \wedge s111 > s213 \wedge s111 \geq s221 \wedge s111 \geq s222 \wedge s111 > s231 \wedge s111 > s232 \wedge s111 \geq s311 \wedge s111 > s312 \wedge s111 > s321 \wedge s111 > s322 \wedge s112 \geq s113 \wedge s112 \geq s133 \wedge s112 \geq s232 \wedge s112 \geq s322 \wedge s113 \geq s133 \wedge s121 \geq s131 \wedge s121 \geq s133 \wedge s121 \geq s232 \wedge s121 \geq s322 \wedge s122 \geq s133 \wedge s122 \geq s232 \wedge s122 \geq s322 \wedge s123 \geq s133 \wedge s123 \geq s322 \wedge s131 \geq s133 \wedge s131 \geq s322 \wedge s132 \geq s133 \wedge s132 \geq s232 \wedge s133 \leq s211 \wedge s133 \leq s212 \wedge s133 \leq s213 \wedge s133 \leq s221 \wedge s133 \leq s222 \wedge s133 \leq s231 \wedge s133 \leq s232 \wedge s133 \leq s311 \wedge s133 \leq s312 \wedge s133 \leq s321 \wedge s133 \leq s322 \wedge s211 \geq s231 \wedge s211 \geq s311 \wedge s211 \geq s321 \wedge s211 \geq s322 \wedge s212 \geq s213 \wedge s212 \geq s312 \wedge s221 \geq s232 \wedge s222 \geq s232 \wedge s222 \geq s322$
4000	400	13385277	$s223 = s232 \wedge s223 = s233 \wedge s223 = s313 \wedge s223 = s322 \wedge s223 = s323 \wedge s223 = s331 \wedge s223 = s332 \wedge s223 = s333 \wedge s111 \leq 2 \wedge s111 \geq 2 \wedge \mathit{true} \wedge s112 \leq 2 \wedge s113 \leq 2 \wedge s121 \leq 2 \wedge s122 \leq 2 \wedge s123 \leq 2 \wedge s131 \leq 2 \wedge s132 \leq 2 \wedge s133 \leq 1 \wedge s211 \leq 2 \wedge s212 \leq 2 \wedge s213 \leq 2 \wedge s221 \leq 2 \wedge s222 \leq 2 \wedge s223 \leq 0 \wedge s231 \leq 2 \wedge s311 \leq 2 \wedge s312 \leq 2 \wedge s321 \leq 1 \wedge s111 \geq s112 \wedge s111 \geq s113 \wedge s111 \geq s121 \wedge s111 \geq s122 \wedge s111 \geq s123 \wedge s111 \geq s131 \wedge s111 \geq s132 \wedge s111 > s133 \wedge s111 \geq s211 \wedge s111 \geq s212 \wedge s111 \geq s213 \wedge s111 \geq s221 \wedge s111 \geq s222 \wedge s111 > s223 \wedge \mathit{true} \wedge s111 \geq s231 \wedge s111 \geq s311 \wedge s111 \geq s312 \wedge s111 > s321 \wedge s112 \geq s113 \wedge s112 \geq s133 \wedge s112 \geq s223 \wedge s113 \geq s223 \wedge s121 \geq s131 \wedge s121 \geq s133 \wedge s121 \geq s223 \wedge s122 \geq s133 \wedge s122 \geq s223 \wedge s123 \geq s133 \wedge s123 \geq s223 \wedge s131 \geq s133 \wedge s131 \geq s223 \wedge s132 \geq s223 \wedge s133 \leq s211 \wedge s133 \geq s223 \wedge s211 \geq s223 \wedge s211 \geq s311 \wedge s212 \geq s223 \wedge s213 \geq s223 \wedge s221 \geq s223 \wedge s222 \geq s223 \wedge s223 \leq s231 \wedge s223 \leq s311 \wedge s223 \leq s312 \wedge s223 \leq s321$
1000	500	10495696	$s133 = s232 \wedge s133 = s233 \wedge s133 = s313 \wedge s133 = s323 \wedge s133 = s331 \wedge s133 = s332 \wedge s133 = s333 \wedge s111 \leq 2 \wedge s111 \geq 2 \wedge \mathit{true} \wedge s112 \leq 2 \wedge s113 \leq 2 \wedge s121 \leq 2 \wedge s122 \leq 2 \wedge s123 \leq 2 \wedge s131 \leq 2 \wedge s132 \leq 2 \wedge s133 \leq 0 \wedge s211 \leq 2 \wedge s212 \leq 2 \wedge s213 \leq 2 \wedge s221 \leq 2 \wedge s222 \leq 2 \wedge s223 \leq 1 \wedge s231 \leq 1 \wedge s311 \leq 2 \wedge s312 \leq 2 \wedge s321 \leq 1 \wedge s322 \leq 1 \wedge s111 \geq s112 \wedge s111 \geq s113 \wedge s111 \geq s121 \wedge s111 \geq s122 \wedge s111 \geq s123 \wedge s111 \geq s131 \wedge s111 \geq s132 \wedge s111 > s133 \wedge \mathit{true} \wedge s111 \geq s211 \wedge s111 \geq s212 \wedge s111 \geq s213 \wedge s111 \geq s221 \wedge s111 \geq s222 \wedge s111 > s223 \wedge s111 > s231 \wedge s111 \geq s311 \wedge s111 \geq s312 \wedge s111 > s321 \wedge s111 > s322 \wedge s112 \geq s113 \wedge s112 \geq s133 \wedge s112 \geq s213 \wedge s112 \geq s223 \wedge s113 \geq s133 \wedge s113 \geq s223 \wedge s121 \geq s131 \wedge s121 \geq s133 \wedge s121 \geq s223 \wedge s121 \geq s322 \wedge s122 \geq s133 \wedge s122 \geq s223 \wedge s123 \geq s133 \wedge s123 \geq s223 \wedge s131 \geq s133 \wedge s132 \geq s133 \wedge s133 \leq s211 \wedge s133 \leq s212 \wedge s133 \leq s213 \wedge s133 \leq s221 \wedge s133 \leq s222 \wedge s133 \leq s223 \wedge s133 \leq s231 \wedge s133 \leq s311 \wedge s133 \leq s312 \wedge s133 \leq s321 \wedge s133 \leq s322 \wedge s211 \geq s311 \wedge s211 \geq s322 \wedge s212 \geq s223 \wedge s212 \geq s322 \wedge s213 \geq s223 \wedge s312 \geq s322$
2000	500	21603820	$s133 = s232 \wedge s133 = s233 \wedge s133 = s322 \wedge s133 = s323 \wedge s133 = s331 \wedge s133 = s332 \wedge s133 = s333 \wedge s111 \leq 2 \wedge s111 \geq 2 \wedge \mathit{true} \wedge s112 \leq 2 \wedge s113 \leq 2 \wedge s121 \leq 2 \wedge s122 \leq 2 \wedge s123 \leq 2 \wedge s131 \leq 2 \wedge s132 \leq 2 \wedge s133 \leq 0 \wedge s211 \leq 2 \wedge s212 \leq 2 \wedge s213 \leq 2 \wedge s221 \leq 2 \wedge s222 \leq 2 \wedge s223 \leq 1 \wedge s231 \leq 2 \wedge s311 \leq 2 \wedge s312 \leq 2 \wedge s313 \leq 1 \wedge s321 \leq 2 \wedge s111 \geq s112 \wedge s111 \geq s113 \wedge s111 \geq s121 \wedge s111 \geq s122 \wedge s111 \geq s123 \wedge s111 \geq s131 \wedge s111 \geq s132 \wedge s111 > s133 \wedge \mathit{true} \wedge s111 \geq s211 \wedge s111 \geq s212 \wedge s111 \geq s213 \wedge s111 \geq s221 \wedge s111 \geq s222 \wedge s111 > s223 \wedge s111 \geq s231 \wedge s111 \geq s311 \wedge s111 \geq s312 \wedge s111 > s313 \wedge s111 \geq s321 \wedge s112 \geq s113 \wedge s112 \geq s133 \wedge s112 \geq s223 \wedge s112 \geq s313 \wedge s113 \geq s133 \wedge s113 \geq s313 \wedge s121 \geq s131 \wedge s121 \geq s133 \wedge s121 \geq s223 \wedge s121 \geq s313 \wedge s122 \geq s133 \wedge s122 \geq s223 \wedge s123 \geq s133 \wedge s131 \geq s133 \wedge s132 \geq s133 \wedge s133 \leq s211 \wedge s133 \leq s212 \wedge s133 \leq s213 \wedge s133 \leq s221 \wedge s133 \leq s222 \wedge s133 \leq s223 \wedge s133 \leq s231 \wedge s133 \leq s311 \wedge s133 \leq s312 \wedge s133 \leq s313 \wedge s133 \leq s321 \wedge s211 \geq s223 \wedge s211 \geq s311 \wedge s211 \geq s312 \wedge s211 \geq s313 \wedge s213 \geq s313 \wedge s221 \geq s223 \wedge s222 \geq s223$

still not close to the actual values, but are much more informative than in the more relaxed case.

Perhaps a yet more interesting result from these graphs is that the obtained values are exactly the same both for the total infection property as well as the corner infection, although verification times are higher for the total infection case due to the added complexity of the formula that describes this total infection. This suggests that the bound is being calculated to the point of reaching the trap state rather than the actual infection states. This is confirmed by the invariants obtained, that effectively prune the infection states out of the partial state space. Table VIII shows a subset of the invariants for the submodels where partial verification was feasible.

Summary of results. What all case studies and experiments indicate is that, through careful partial exploration of the model, we can obtain bounds for reward estimation

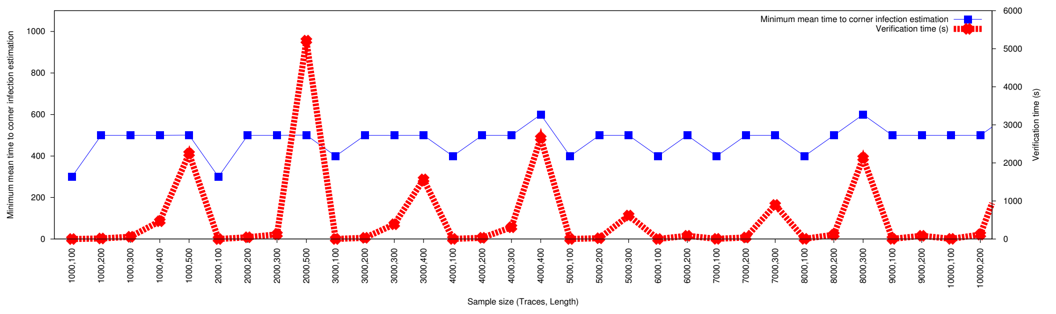


Fig. 13: Minimum mean time to total infection. Bounds calculated on submodels obtained through combinations of traces and trace lengths.

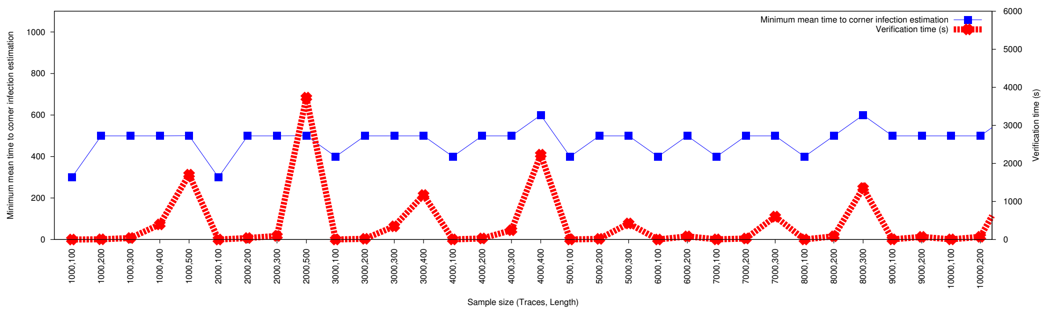


Fig. 14: Minimum mean time to corner infection. Bounds calculated on submodels obtained through combinations of traces and trace lengths.

with very low percentages ($< 1.5\%$) of the actual state space explored, and that these bounds are more useful than those obtained through an analysis of the complete state space. Further, submodels that yield these results also converge very quickly (much before the 24 hour timeout) to good estimation results. While the estimation does constantly improve during the rest of the 24 hours, it does so at a slower pace. This is good news, as even with the trace analysis, good results can still be attained under the same time budget. From these results it follows that, for these case studies, effort into estimating reward values through automatically obtained submodels through model invariants of the full model pays off.

It must be noted that it is possible that the *actual* value of the reward being estimated is much larger than any of those obtained. Of course, we are always limited by the fact that the actual reward value cannot be calculated, neither with partial nor full models. It can be argued, though, that it is often the case that the exact value is not needed as such; rather, satisfying a minimum threshold value is a sufficient guarantee for the reliability measure being analysed. Hence, methods which provide higher lower bounds faster are useful.

It is also interesting to note that the efficiency of our proposed approach does not seem to depend on whether the states tested for reachability are actually reachable in the submodels or not. For example, in both the Tandem Queue and BRP cases, the inferred invariants preclude the failure states from appearing in the submodels. However, in the case of the WLAN protocol the interesting state which describes the protocol success (and for which we aim at calculating its turnaround time) is not cut out from the submodels by the invariant, so states that do satisfy the property exist in the submodels (although it is possible that not all states that satisfy the property are in the submodel).

We also found evidence that the degree of non-deterministic behaviour present in the model plays a role on the quality of the estimations as well. This is a consequence of the simulation phase, as non-determinism is simulated via an equiprobable distribution. The result of this non-determinism resolution is that non-deterministic behaviour will tend to be explored in a *flat* manner, not unlike a BFS exploration. The end result is that complex, deep behaviour is not witnessed as frequently, which yields models that are poor in describing rich behaviour.

Question 2. Contrasting to the previous experimentation that aimed to compare our approach with probabilistic model checking, Q2 aims to establish a comparison with Monte Carlo techniques. Experimentation to answer this question is not straightforward due to the problem of generating sufficient failing simulations to ensure given precision and confidence parameters. We first aimed at performing a straightforward statistical analysis of the model. A first experiment was designed requiring a result precision of 99%. As is standard for statistical analyses, we also required a 95% confidence.

A straightforward calculation of the necessary sample size based on the Chernoff bound [Chernoff 1952] determines that a total of ~ 60000 samples are necessary, which does not seem excessive. However recall that each sample must eventually reach a state where the property can be determined to be true or false. For systems where witnessing this behaviour is rare, this means that samples may be extremely long. Through trial and error, and based on the bounds obtained in *Q1*, we tried to determine the minimum length for samples to consistently reach failure states. For the Tandem Queue full model—for which its MTTF was already estimated to be at least 7×10^7—even samples as long as 4×10^8 do not consistently reach the failure state where the queues are both full. Considering that generating a sample of such length takes 15 minutes, generation of the full 60000 traces required leads to a 2 year period for sample generation. A similar situation is found upon analysis of the BRP model.

Relaxing the precision requirement to 95% reduces the sample generation cost to 1 month. Further relaxation to 90% still requires a week of execution. In fact, if we were to set a 24 hour budget for sample generation, the precision obtained would be of just 70%. That is, the MTTF estimate would be up to $\pm 30\%$ away from the true MTTF value with a 95% guarantee. Note that this is a very conservative estimate as it is unlikely that all traces of length 4×10^8 generated in the 24 hour period will consistency reach failure states, and possibly much lengthier traces will be needed.

To overcome this limitation of standard Monte Carlo verification, we tried carrying out a variation of Wald's sequential testing [Nimal 2010]. This procedure generates samples while at the same time it determines whether more samples are necessary or not. As a result of this online estimation, it might require less samples than those mandated by the Chernoff bound, although it cannot be stated beforehand how many samples will be needed exactly. This optimization does not eliminate the need for samples to reach property-determining states, so sample length remains a problem. We attempted to perform this analysis truncating generated samples at length 4×10^8 and treating them as *failing* samples once they reached this threshold. This is a similar strategy as the one used in our approach (anything beyond the submodel is a failure). However, this procedure yielded no results after 24 hours of execution, indicating that the sequential testing still needed more evidence in order to produce a reasonable estimate.

Furthering this strategy of over-approximation of failures in Monte Carlo verification, we generated samples over the submodels with highest MTTF obtained in *Q1* rather than over the full model. However, the problem of producing samples that consistently fail persisted, failing to provide an estimate for MTTF in the budgeted time. These results suggest that Monte Carlo approaches may be unsuitable to answer reliability questions in systems with high MTTF (i.e., rare failures).

Monte Carlo analysis of non-rare events. The WLAN and Virus infection case studies, as opposed to the previous ones, do not depict rare events. As we have already seen, the expected times to the interesting properties are short enough that they should be attainable by a reasonable simulation. However, both of these models are non-deterministic, and we will see in this section that this characteristic introduces a second problem for Monte Carlo approaches.

As a way to illustrate this problem, we first set out to estimate the minimum and maximum turnaround times for the WLAN collision avoidance protocol. Recall that we already analysed this model completely and found these times to be 1725 and 4301.65 for the minimum and maximum cases respectively. In the previous section, we already established that 60000 samples would be necessary for a robust estimation. Since we know that the maximum expected turnaround time is ~ 4300, we set the trace horizon to 10000 in order to have a reasonable confidence that every trace would hit the success state (i.e., one where both stations have sent their data successfully).

The obtained results are disconcerting, however. In both cases, the estimation procedure was efficient, as it only required 80 seconds of execution in both cases. The reason for this fast sample generation is that not only is the bound low, but the required property is reached on an average of 25 steps as well. This is because, unlike the other case studies, the reward structure for the WLAN case assigns a reward of at least 50 to transitions. Because of these reasons, most samples are very short and are generated very quickly.

The estimations themselves are the problem in this case. For the minimum turnaround estimation, we obtained a time of 2729.45 ± 0.1929 with 95% confidence. Surprisingly enough, the estimation for the *maximum* turnaround is extremely similar: 2731.06 ± 0.1915 with 95% confidence. Not only are both results the same, they are equally incorrect.

The estimation analysis for the Virus infection case does not fare better. We have already noted that we could calculate the minimum time to complete infection and the minimum time to infection of the opposite corner network node in an analytical way. We already calculated these expected times to be 5200 and 1200 respectively. Additionally, we know that the maximum expected time is actually infinite. This makes the setting of a trace horizon as difficult as in the Tandem Queue and BRP cases. In fact, experimentation showed that traces as long as 10^7 steps long do not consistently reach the target state. This situation renders the estimation analysis as infeasible as in the BRP and Tandem Queue cases.

On the other hand, since the minimum bounds are low enough, we set out to use them as bounds for a bounded probability analysis. We performed Monte Carlo estimations of the probability of reaching total infection before the expected 5200 steps, and the probability of infecting the opposite node before the 1200 steps expected in that case. The actual values can be easily obtained by analysing the negative binomial distribution.

The results for these analyses are included in Table IX. Again, it can easily be seen that these results cannot be correct.

These (incorrect) results can be easily explained, however. Unfortunately, Monte Carlo approaches are not very good at dealing with non-determinism [Henriques et al. 2012]. Statistical simulation approaches are based on the fact that each simulated sample can be unequivocally *quantified* with its probability of being witnessed.

Table IX: Monte Carlo estimations for the WLAN collision avoidance protocol and Virus infection systems.

Property	Known value	Time to estimation	Estimation
WLAN minimum turnaround	1725.00	81.67 sec.	2729.45 ± 0.1929
WLAN maximum turnaround	4301.65	80.22 sec.	2731.06 ± 0.1915
Max. prob. of total network infection before 5200 steps	0.51872	20 *hours*	0.00 ± 0.00
Max. prob. of corner infection before 1200 steps	0.53898	4 *hours*	0.00 ± 0.00
Max. prob. of total network infection before 5200 steps (constrained model)	0.51872	693.34 sec.	$0.54200 \pm 9.8 \times 10^{-4}$
Max. prob. of corner infection before 1200 steps (constrained model)	0.53898	166.23 sec.	0.00 ± 0.00

This is not true once non-determinism is introduced into the model, since the non-deterministic choices cannot be quantified. We have attempted to quantify these choices in a way equivalent to that of our partial exploration approach, that is, considering non-deterministic choices as equiprobable. This, however, introduces a bias that is difficult in general to remove at the moment of performing the actual estimation.

This uniform choice explains why both minimum and maximum estimations resulted in the same values. Since the neither the *best* nor the *worst* schedulers are uniform in their choice, these extreme behaviours are not witnessed, and therefore cannot be estimated. The second problem is that turning a non-deterministic choice into a probabilistic one introduces a bias that cannot be estimated itself. As a result, estimation results when non-determinism is present are meaningless.

Surprisingly, this uniformity also explains why the Monte Carlo approach yielded a result close to the actual one in the case of total virus infection, but not in the case of corner infection. In the case of total infection, since every node needs to be infected, every non-deterministic choice needs to be taken. Since the Monte Carlo simulations are more or less uniform in resolving non-determinism, they turn out to actually be selected, and therefore provide a result close to the true one. However, in the case of corner infection, only non-deterministic options that lead to advance towards the corner have to be selected. This is not the case for uniform non-determinism resolution, and therefore the (wrongly) estimated probability is 0.

Question 3. In this section, we compare the results obtained while answering *Q1* with the results a practitioner might obtain by specifying invariants herself, based on her knowledge of the model. Prior to experimenting on automatically generated invariants, we analysed the models and came up with at least one invariant for each one. These invariants were selected based on our understanding that their negation is a necessary condition for reaching failure states.

For the Tandem Queue case study, we established the invariant to be that the total number of enqueued processes globally in both queues is less than c, and ran experiments for different values of c ranging up to the total capacity of the queueing system ($2 \times C$). A failure entails that the invariant does not hold for $c < 2 \times C$, and that for $c = 2 \times C$ the resulting invariant-driven submodel is exactly the whole model. In our experiments we found that there exist multiple c values for which the invariant resulted in a significantly higher MTTF than the MTTF estimated for the full model.

In Table X results are presented for various invariant-driven submodel parameter values together with estimated MTTF and computation time using the BGS method.

Table X: Experimental results for tandem queue (2×1200 processes) and BRP (256 retries) mean times failure times.

c	Size	BGS	
		MTTF	Time
20	2398 st 6560 tr	$0.83{\cdot}10^3$	68.75 s
40	8778 st 24280 tr	$1.12{\cdot}10^4$	82.72 s
60	19158 st 53200 tr	$1.25{\cdot}10^5$	276.69 s
80	33538 st 93320 tr	$1.36{\cdot}10^6$	64.06 m
100	51918 st 144640 tr	$1.49{\cdot}10^7$	17.93 h
120	74298 st 207160 tr	$5.50{\cdot}10^7$	TO
140	100678 st 280880 tr	$4.63{\cdot}10^7$	TO
160	131058 st 365800 tr	$3.17{\cdot}10^7$	TO
180	165438 st 461920 tr	$2.31{\cdot}10^7$	TO
200	203818 st 569240 tr	$1.66{\cdot}10^7$	TO
900	4067118 st 11381440 tr	$8.41{\cdot}10^5$	TO
1600	11219198 st 31407194 tr	$4.20{\cdot}10^5$	TO
2400	14362898 st 40213194 tr	$4.20{\cdot}10^5$	TO

retries	Size	BGS	
		MTTF	Time
1	366915 st 489574 tr	$1.50{\cdot}10^6$	21.06 h
2	480460 st 646758 tr	$1.69{\cdot}10^7$	TO
5	821095 st 1118310 tr	$1.08{\cdot}10^7$	TO
10	1388820 st 1904230 tr	$6.29{\cdot}10^6$	TO
50	5930620 st 8191590 tr	$1.39{\cdot}10^6$	TO
150	17285120 st 23909990 tr	$4.86{\cdot}10^5$	TO
250	28639620 st 39628390 tr	$2.73{\cdot}10^5$	TO
256	N/A st N/A tr	N/A	OOM

From the table it follows that the best MTTF is obtained for the submodel which considers up to 120 processes queued (MTTF $> 5.5 * 10^7$).

In the case of the Bounded Retransmission Protocol case study, a parametric invariant chosen was that the number of retries performed while transmitting a single file was less than $max_{retries}$. We ran experiments for different values of $max_{retries}$ ranging up to the true maximum number of retries (256). A failure entails that the invariant does not hold for $max_{retries} < 256$. For $retries = max_{retries}$ the resulting invariant-driven submodel is the whole model.

Again, we show a selection of submodels, ranging from the very small upwards to almost the complete model. Results for these experiments are depicted in Table X. Estimation results are even more significant than for the previous case study considering that analysis of the full model with 256 retries was not possible within the memory budget. However, the trend indicates that augmenting the number of retries considered does not yield better MTTF and in fact, a very low number of retries gives a much higher MTTF.

Although the WLAN collision avoidance protocol could be verified in its totality, we nevertheless ventured an invariant that we thought would be useful in reducing the state space. It turns out in this case that our proposed invariant is much simpler than those inferred by the automatic approach, as our initial belief was that bounding the time a sending station is forced to backoff, the model would be reduced. This interpretation, however, turned out to be erroneous. In fact, regardless of how many times a sending station found a collision, the backoff time is chosen uniformly over the whole possible range. The results we obtained by applying these invariants are presented in Table XI. Note that even restricting the backoff time to just one value (zero) does not

Table XI: Selection of WLAN submodel TAT evaluation results for different manual invariants.

`backoff1` and `backoff2` bounding		Model checking			
Max. backoff time	States	Min. TAT	Time	Max. TAT	Time
0	59185713	465.97	109.36s	1201.71	176.88s
5	64160812	559.68	206.37s	1273.44	224.98s
10	68239697	686.78	304.47s	1460.94	286.30s
15	71431132	901.65	440.29s	1764.45	364.90s
20	73735117	1157.81	614.94s	2244.19	435.67s
25	75151652	1392.19	641.49s	2922.23	781.37s
30	75680737	1665.63	490.05s	3846.17	1085.87s

Table XII: Experimental results for mean times to total infection with manual invariants.

# infected	Size	Min. time to infection	
		Value	Time
1	74 st 222 tr	199.88	~ 0.00 s
2	1269 st 5233 tr	399.69	0.07 s
3	19181 st 99607 tr	599.55	0.64 s
4	351990 st 2215026 tr	799.43	17.88 s
5	6035220 st 44517828 tr	999.32	414.14
≥ 6	N/A st N/A tr	N/A	OOM

really reduce the size of the model. Although for smaller values of this bound the verification time is reduced drastically, these execution times are still much larger than those that result from the automatically inferred invariants. Further, the turnaround times obtained, both minimum and maximum, are very poor contrasted with those that resulted from the automatic approach.

Finally, we turn our attention to the Virus infection model. The manually stated invariants in this case deal with limiting the number of infected nodes that can coexist at once. We first applied these invariants to the original, unconstrained model. As was the case with the results obtained with our approach, these manually inferred invariants can't restrict the model size enough. Setting the limit to just two infected nodes, we quickly obtained a bound to minimum mean time to failure of ~ 200, the same value obtained with our approach. However, raising this limit to three infected nodes makes analysis infeasible.

Consequently, we applied these same manual invariants to the constrained infection model. The results of these analyses are pictured in Tables XII and XIII.

In this case, it can be seen that these manually posed invariants perform slightly better than the automatically inferred ones. More specifically, increasing the limit of infected nodes by one results in model size increases that do not grow as dramatically as in the case of growing the number of traces and their length in the automatic approach. This allows for better submodels to be obtained and therefore better bounds, up to 5 infected nodes. On the other hand, the obtained bounds on times to failure and probabilities are still far from the actual values.

Summary of manual invariants analyses. In the cases where the manual invariants did succeed, it is interesting to note that for relatively small submodels (e.g. $c = 80$

Table XIII: Experimental results for mean times to corner infection with manual invariants.

# infected	Size	Min. time to infection	
		Value	Time
1	74 st 222 tr	199.88	~ 0.00 s
2	1269 st 5233 tr	399.68	0.07 s
3	19181 st 99607 tr	599.55	2.79 s
4	351990 st 2215026 tr	799.43	51.71 s
5	6035220 st 44517828 tr	999.32	1241.86 s
≥ 6	N/A st N/A tr	N/A	OOM

on the Tandem Queue case study, and $max_{retries} < 2$ for BRP) the estimated MTTF is much higher than the MTTF computed over the complete model. Still, while the manual invariant approach did provide useful bounds, it turns out that the best MTTF values generated by the automatic approach obtains slightly higher bounds for the same time budget. For the Tandem Queue study, the best automatically estimated MTTF is of $\sim 7 \times 10^7$ against $\sim 5.5 \times 10^7$. For the BRP case study the best automatic estimation is $\sim 2.5 \times 10^7$ versus $\sim 1.69 \times 10^7$ when manual intervention is applied. The case of the Virus infection model is atypical, as the manually posed invariants slightly outperformed the automatically inferred ones.

An initial interpretation of the results would suggest that, except for the WLAN case study, automatically inferred invariants do not have an added advantage over manually suggested ones. However, there is an added cost in understanding a protocol model and being able to suggest which factors are the most relevant in increasing a model size or in making numerical computation infeasible. This cost is in general not trivial, and requires a thorough understanding of the modelling formalisms as well as the verification procedures under the hood. These are not, a priori, traits that every engineer can be reasonably expected to have.

5. DISCUSSION AND RELATED WORK

In this paper, we have presented a fully automated technique for reward estimation of system models. Experimental results have shown that this approach may provide more useful estimations than both standard probabilistic model checking and Monte Carlo verification, at a fraction of the cost required by such techniques. The strength of our technique lies on the fact that our simulation approach identifies the states which are more likely to be traversed over real executions. Moreover, the inclusion of all states satisfying the invariant results in the addition of many execution loops to the submodel, which further increase the probability that an arbitrary execution is captured by the inferred submodel. This results in submodels that, with a reduced number of states, still capture a large part of the probabilistic behaviour of the system. This reduced size, in turn, allows the iterative verification of pCTL properties to perform, in the same time budget, an increased number of iterations than are possible for the larger, complete model. This increased iteration results in faster convergence to the actual values that are being sought, and therefore in better approximations. We have also observed that these results are especially notorious when the properties under analysis are probabilistically rare.

However, some parameters exist that need to be set for the approach to work. First, there is the matter of the size of the simulation set and the length of the simulated traces; and second, in the case where non-determinism is present in the model under analysis, a strategy is necessary for solving these non-deterministic choices during the simulation phase.

Regarding the size of the simulation set and its traces, good news is that our experimentation has shown that, at least for the examples studied, very good results can be obtained through a relatively small set of short traces. Although the elaboration of guidelines on how to set the number and length of traces is beyond the scope of this paper, results show that there may be a broad combination of parameter values for which high estimation results are obtained in reasonable time. Further, overshooting these parameters does not have a dramatic impact in the resulting submodel size, so erring in the side of caution and choosing larger parameters does not seem to be a cause for concern.

It is important to note that exploration of an appropriate parameter space can be done concurrently, taking as the final reward estimation the highest of the bounds obtained. Such an approach would leverage on the fact that, as can be seen from Figures 5 and 8, the estimation of results over partial explorations quickly converges to a value, while it refines this value over the rest of the alloted time. Note from these Figures that an initial good estimation can be obtained in less than an hour. Future work is focused on taking advantage of this fact to estimate good trace and trace length parameters. This approach would call for an initial spawning of several concurrent estimation processes, each with a different valuation for trace and trace length generation. Given the initial estimation, we can quickly compare which parameter combinations outperform the others. These parameters could then be further refined and compared, and then settling with the best parameters obtained after a given set-up time has elapsed. This approach could also be enhanced with heuristic searches that look for the best parameter combination. All of this requires additional experimentation and remains future work.

Note that this set-up time can also be set low enough to still be much less than the time required to build the full model. We also recall that full model probabilistic checking cannot exploit concurrent computation in such a way. Monte Carlo verification can be applied concurrently. However, as we have seen in our experimentation in this section, the number of traces and their length are so large that massive parallelisation would be required to diminish its impact. This significant time cost for sample generation would not be outweighed by concurrent execution. Further experimentation is needed to address this point.

A second point that merits additional attention is the invariant inference procedure. In the work presented in this paper, the invariants were produced by letting the Daikon tool analyse linear relationships between any pair of variables in the model and additional fixed bounds. However, Daikon provides other criteria for invariant inference that were not explored in the course of the work presented in this paper, such as checking the variable values for specific set membership or common value sequences. Although the results we present in this work are very positive, it remains further work to perform a deep comparison with other invariant inference configuration. Different configurations may not only yield various invariants and therefore different submodels, but also may take more (or less) time to be computed. This point merits further work and experimentation.

As was previously mentioned, most probabilistic model checkers [Katoen et al. 2011; Hinton et al. 2006; Sen et al. 2005b; Younes 2005; Sun et al. 2009] provide functionality that may either reduce the time required to obtain results, or reduce the memory footprint required for verification, such as symmetry reductions [Kwiatkowska et al.

2006], *lumping* [Dean and Givan 1997] and several numerical methods. All these optimizations are orthogonal to the model checking procedure itself. Our work relies on probabilistic model checking and the experiments were run on PRISM, which implements some of these optimizations.

In those settings where exhaustive probabilistic model checking of models is intractable due to required memory size or verification time, statistical simulation has proven to be an effective technique. As was mentioned in section 4, an important issue with simulation approaches is that they tend to work well mostly in the case that the specified properties are bounded in time, i.e. when these properties can be written in the form $\psi\mathcal{U}^{\leq T}\rho$ for a fixed T. This is so because estimation of the random variable X_ϕ by means of a sample of traces σ_i requires that the question of whether $M, \sigma_i \models \phi$ or not be answered in a definite way for each trace σ_i in the sample set. If the formula ϕ is temporally bounded, then termination is guaranteed when evaluating its truth for the traces, but for temporally unbounded formulae such termination is threatened. In such cases, generating traces within acceptable length bounds that answer the property definitively can be very unlikely. To address this problem *biased sampling* [Sen et al. 2005a; Rabih and Pekergin 2009; Lassaigne and Peyronnet 2006; Basu et al. 2009] has been studied. However, bias to sampling must be done manually resulting in an impact on the analysis results that cannot be quantified in general. The result obtained by our approach is guaranteed to be a true bound to the reward values being sought after.

Recent work by Younes et. al. [Younes et al. 2011] proposes two novel Monte Carlo approaches that do not rely on biased sampling. However, one of them may require an inordinate number of samples to produce results; while the other relies on reachability analysis, which requires the full model to be constructed, relinquishing one of the key advantages of Monte Carlo model checking over probabilistic model checking. The work in [He et al. 2010] also presents a bounded statistical approach for checking unbounded properties that does not need the full model to be constructed. However, the bound on the necessary trace length is excessively large, as traces may be as long as the total number of states in the model. Other works [Kaufman et al. 2002] acknowledge the problem of generating traces exhibiting the failure (or guaranteeing its absence). This approach relies on extreme value theory to produce results. Unfortunately, extreme values techniques still require a good number of actual samples exhibiting the property, as these techniques require the inference of a fitting distribution. Having too few samples to work with usually results in fitting distributions that are actually different than the one being analysed [Coles 2001].

As noted, an additional point for analysis lies in the solving of non-deterministic choices during simulation. Several works have attempted to solve this problem, especially in the context of generating simulations for Monte Carlo estimation. In these cases, it is critical that the simulation of non-deterministic transitions is performed in such a way that there is no bias in the generation (or alternatively, in such a way that this bias can be controlled and quantified), as doing so otherwise would introduce errors in the final estimation. In [Henriques et al. 2012] the authors leverage on the fact that, usually, verification is performed while looking for the worst and best cases. In that sense, only the two schedulers that induce the best and worst results are of interest, and the authors propose a self-adjusting simulation algorithm that converges to these extremes.

In [Bogdoll et al. 2011], rather than focusing on the problem of biasing scheduler selection, the authors aim at detecting whether non-determinism can be ignored safely. As the authors point out, it is often the case that non-deterministic choices are actually behaviour-equivalent. By detecting these situations via partial order methods, it can be used to identify situations where non-determinism can be ignored while keeping only one of the possible choices when performing simulation.

In our present work, we have opted to resolve non-determinism by simply assuming an equiprobable distribution over the possible non-deterministic choices at a given state. However, it must be noted that, in the context of our work, *any* method of resolving non-determinism would have been acceptable, as we always produce a lower bound to the actual reward value, regardless of the procedure used for simulation. This is not to say that any non-determinism resolution method will produce the same outcome, as different choices may lead to different invariants. Although the results presented in this paper are promising, it still remains to be seen if different approaches to the initial simulation might produce even better results. In particular, the choice of simulating via equiprobable distribution of non-deterministic transitions is a double-edged sword. On the one hand, by establishing a balanced choice, it maximises the chance of exploring most of the non-deterministic alternatives so that verification of all of them is carried out at a later step. But, on the other hand, some of this explored behaviour might possibly be irrelevant when calculating the maximum (or minimum) rewards, as the best and/or worst schedulers might never take some of the explored non-deterministic transitions. In this sense, adapting the approach of [Henriques et al. 2012] to the simulation step of our framework might prove to be beneficial. Although that proposed approach is geared towards model checking of probabilistic properties rather than reward calculations, it may be adapted to our needs. It is worth noting, however, that such an approach would need to carry out two simulation steps as opposed to one. This is because the approach in [Henriques et al. 2012] aims at simulating executions that resemble those of the extreme scheduler that is of interest, which may be either the one providing the minimum value, or the maximum, but not both at the same time. In that sense, if we are interested in calculating both extreme values, we would need different simulation sets, one for each extreme.

The analysis of system behaviour that exhibits *rare* yet relevant events (e.g. failures) is the subject of focused study within the simulation community as well. A technique that is usually used in conjunction with stochastic processes that have rare events is that of *importance sampling* [Rubinstein and Kroese 2008]. Roughly speaking, the idea of importance sampling is to replace the original process's distribution for another more likely to generate the (originally) rare event during the sample generation. The distribution replacement is chosen so that results from analyses for the new distribution can be translated back to results valid for the original distribution. Although this is a promising approach, finding suitable replacement distributions is a complex and ad-hoc task for which further research and expertise is necessary, as different system models possibly require different sampling distributions. Further, special care is required when proposing importance sampling distributions. In fact, it is possible to choose a replacement distribution such that it makes the simulation process *more costly* and requiring even more samples than the original one. In practice, choosing optimal replacement distributions is extremely difficult and not suitable for a general, complex process model.

Another promising simulation technique that also focuses on rare events is that of *sample splitting* [Rubinstein and Kroese 2008; Robert and Casella 2005], most notably the RESTART implementation [Villén-Altamirano and Villén-Altamirano 1994] which, roughly, rather than starting each simulation from the initial state, it does so from a state s visited in a previous simulation and from which reaching a rare event is more likely. The likelihood of reaching state s from the initial state is taken into account for producing the final analysis results. Key to the application of these techniques is making appropriate decisions on where to restart simulations. These decisions demand deep understanding of both the model and the underlying splitting technique, as naïve splitting may not help the verification effort. Worse, it could even hamper the effort if the splits are not done in such a way that they are incrementally closer to fulfilling the rare event. Another interesting approach is that of [Reijsbergen et al. 2013], which is geared toward simulating rare events, although restricted to Stochastic Petri Nets.

Finally, common to both the Monte Carlo approach and the simulation techniques discussed is the fact that they are inherently statistical results. As such, there is always a non-zero probability that the results obtained are completely off the mark. Further reducing this error probability may require excessive amount of additional traces to be sampled in order to obtain the guarantee. Our technique, though conservative in the bounds it obtains, is definitive in its answers.

The work we present in this paper is concerned with the verification of systems that are specified through the use of automata-like languages. We believe our approach can be extended in order to analyse source code as well. In this regard, there have been promising advances similar to our work. For example in [Filieri et al. 2013; Borges et al. 2014], symbolic execution is used to analyse the source code, and that information is used to direct a sampling approach towards interesting portions of the source code. The setting for this work is different and complementary, though, as it focuses on non-reactive, non-probabilistic software (by quantifying the usage profile of *program variables*); and the inference of conditions for reaching a given portion of the code. Further, this approach requires the solution space to be built and available for analysis; we argue that this, in our setting, is prohibitive in size.

On a related note, [Luckow et al. 2014] has tackled the problem of synthesising appropriate schedulers for attaining a desired probability, a goal that is closely related to finding the extrema probabilities in the presence of non-determinism. Approaches such as this could benefit our technique by resolving non-determinism in a way that later directs verification to the more extreme (and interesting) values.

As a final note, we point out that the technique presented in this paper is not, and is not intended to be, effective for every possible probabilistic model and probabilistic property. Although in this we work we have pushed the feasibility boundary further by allowing the analysis of models that are too big to tackle with current techniques, there are still models and properties for which our technique does not provide satisfactory bounds. In particular, it might be the case that the inferred invariants are too general, and therefore the submodels too big; or else that the invariants are too restrictive, resulting in small submodels that yield small, not very informative bounds. For these cases, future work is focused on submodel refinement techniques that can identify states that can be cut away from models that are too large, or added to those that are too small.

6. CONCLUSIONS AND FURTHER WORK

In this paper we have proposed an approach to estimating mean reward values for probabilistic system models. The approach is a novel combination of simulation, invariant inference and probabilistic model checking. We report on experiments that suggest that reward estimation using this technique can be more effective than (full model) probabilistic and statistical model checking for system models. This increase in effectiveness is most evident in the case of models where the properties under analysis are rare events, or else are unbounded in time. In addition, our estimation approach also supports non-determinism besides probabilistic behaviour.

We believe the notion of reliability analysis over partial yet systematic explorations offers an alternative to, and hence complements, exhaustive model exploration–as in probabilistic model checking–and partial random exploration–as in statistical model checking.

The experimental results presented in this paper are promising. Our experiments show that, for system models extracted from reliability and probabilistic verification literature, lower bounds can be obtained with little effort compared to full model verification. More specifically, we have shown that we can obtain reliability values that allow for strong dependability arguments, while only performing an exploration of at most 5% of the projected total state space of the system. These savings also translate into verification time as well, and the additional effort required for inferring submodels remains a good trade-off taking into account the quality of the obtained results.

The obtained results are more striking when the behaviours under analysis are rare events, and they have not been witnessed in the (already small) submodel being explored. However, experiments have also shown that our technique is effective even in the case of systems where the behaviour of interest is not rare, and even when some of the states exhibiting this behaviour are present in the obtained submodels. This evidence provides encouragement towards arguing for generalisation of results.

We also believe that further experimentation is required to achieve a better understanding of the influence of parameter choices in the process. In particular, an area that calls for future work is looking for a better understanding of the relationship between the simulated set of traces (both its size as the trace length) and the submodels that result from them, as well as the estimations that can be expected from them. This understanding should lead to heuristics for setting appropriate values to these parameters in order to achieve more cost-effective submodels.

REFERENCES

A. Aziz, V. Singhal, F. Balarin, R.K. Brayton, and A. Sangiovanni-Vincentelli. 1995. It Usually Works: The Temporal Logic of Stochastic Systems. *Lecture Notes in Computer Science* (1995), 155–155.

C. Baier and J.P. Katoen. 2008. *Principles of model checking*. MIT press.

S. Basu, A. Ghosh, and R. He. 2009. Approximate model checking of PCTL involving unbounded path properties. *ICFEM'09* (2009), 326–346.

A. Bianco and L. De Alfaro. 1995. Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science*. Springer, 499–513.

A. Bianco and L. de Alfaro. 1995. Model checking of probabilistic and nondeterministic systems. *Proc. Foundations of Software Technology and Theoretical Computer Science* 1026 (1995), 499–513.

J. Bogdoll, L.M. Ferrer Fioriti, A. Hartmanns, and H. Hermanns. 2011. Partial Order Methods for Statistical Model Checking and Simulation. In *FMOODS/FORTE*. 59–74.

Mateus Borges, Antonio Filieri, Marcelo d'Amorim, Corina S. Păsăreanu, and Willem Visser. 2014. Compositional Solution Space Quantification for Probabilistic Software Analysis. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14)*. ACM, New York, NY, USA, 123–132. DOI:http://dx.doi.org/10.1145/2594291.2594329

H. Chernoff. 1952. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics* 23, 4 (1952), 493–507.

Edmund M Clarke, Orna Grumberg, Marius Minea, and Doron Peled. 1999. State space reduction using partial order techniques. *International Journal on Software Tools for Technology Transfer* 2, 3 (1999), 279–287.

S. Coles. 2001. *An Introduction to Statistical Modelling of Extreme Values*. Springer.

P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. 2001. Reachability analysis of probabilistic systems by successive refinements. In *PAPM/PROBMIV (LNCS)*, Vol. 2165. Springer, 39–56.

Rocco De Nicola, Joost-Pieter Katoen, Diego Latella, and Mieke Massink. 2006. Towards a logic for performance and mobility. *Electronic Notes in Theoretical Computer Science* 153, 2 (2006), 161–175.

T. Dean and R. Givan. 1997. Model minimization in Markov decision processes. In *Proceedings of the National Conference on Artificial Intelligence*. 106–111.

Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. 2007. The Daikon system for dynamic detection of likely invariants. *Sci. Comput. Program.* 69, 1-3 (Dec. 2007), 35–45.

Antonio Filieri, Corina S. Pasareanu, and Willem Visser. 2013. Reliability analysis in symbolic pathfinder. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*. 622–631. http://dl.acm.org/citation.cfm?id=2486870

Ru He, Paul Jennings, Samik Basu, Arka P Ghosh, and Huaiqing Wu. 2010. A bounded statistical approach for model checking of unbounded until properties. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, 225–234.

L. Helmink, M. Sellink, and F. Vaandrager. 1994. Proof-checking a data link protocol. In *Proc. International Workshop on Types for Proofs and Programs (TYPES'93) (LNCS)*, Vol. 806. Springer.

D. Henriques, J. Martins, P. Zuliani, A. Platzer, and E. Clarke. 2012. Statistical Model Checking for Markov Decision Processes. In *QEST*. 84–93.

H. Hermanns, J. Meyer-Kayser, and M. Siegle. 1999. Multi Terminal Binary Decision Diagrams to Represent and Analyse Continuous Time Markov Chains. In *Proc. NSMC'99*. Prensas Universitarias de Zaragoza, 188–207.

A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. 2006. PRISM: A tool for automatic verification of probabilistic systems. In *TACAS'06 Proceedings*, Vol. 3920. Springer, 441–444.

Institute of Electrical and Electronic Engineers. 1997. IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. (1997).

L.H. Jamieson and B.C. Dean. 2007. Weighted alliances in graphs. *Congressus Numerantium* 187 (2007), 76.

IB Kalambi. 2008. A comparison of three iterative methods for the solution of linear equations. *Journal of Applied Sciences and Environmental Management* 12, 4 (2008).

Joost-Pieter Katoen, Ivan S Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N Jansen. 2011. The ins and outs of the probabilistic model checker MRMC. *Performance evaluation* 68, 2 (2011), 90–104.

L.M. Kaufman, B.W. Johnson, and J.B. Dugan. 2002. Coverage estimation using statistics of the extremes for when testing reveals no failures. *IEEE Trans. Comput.* (2002), 3–12.

M. Kwiatkowska, G. Norman, and D. Parker. 2006. Symmetry reduction for probabilistic model checking. In *Computer Aided Verification*. Springer, 234–248.

Marta Kwiatkowska, Gethin Norman, David Parker, and Maria Grazia Vigliotti. 2009. Probabilistic mobile ambients. *Theoretical Computer Science* 410, 12 (2009), 1272–1303.

Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. 1997. Efficient verification of real-time systems: compact data structure and state-space reduction. In *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*. IEEE, 14–24.

R. Lassaigne and S. Peyronnet. 2006. Probabilistic verification and approximation. *ENTCS* 143 (2006), 101–114.

Kasper Luckow, Corina S. Păsăreanu, Matthew B. Dwyer, Antonio Filieri, and Willem Visser. 2014. Exact and Approximate Probabilistic Symbolic Execution for Nondeterministic Programs. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*. ACM, New York, NY, USA, 575–586. DOI:http://dx.doi.org/10.1145/2642937.2643011

Michael R. Lyu. 1996. *Handbook of software reliability engineering*. McGraw-Hill, Inc., Hightstown, NJ, USA.

J.D. Musa, A. Iannino, and K. Okumoto. 1987. *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill.

V. Nimal. 2010. *Statistical Approaches for Probabilistic Model Checking*. M.Sc. Dissertation. Oxford University Computing Laboratory.

Esteban Pavese, Víctor Braberman, and Sebastian Uchitel. 2010. My model checker died!: how well did it do?. In *QUOVADIS/ICSE'10*. ACM, 33–40. DOI:http://dx.doi.org/10.1145/1808877.1808884

Esteban Pavese, Víctor Braberman, and Sebastian Uchitel. 2013. Automated reliability estimation over partial systematic explorations. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 602–611.

Muhammad A Qureshi and William H Sanders. 1996. A new methodology for calculating distributions of reward accumulated during a finite interval. In *Fault Tolerant Computing, 1996., Proceedings of Annual Symposium on*. IEEE, 116–125.

D. Rabih and N. Pekergin. 2009. Statistical Model Checking Using Perfect Simulation. In *Proc. ATVA'09*. Springer-Verlag, 120–134.

Daniël Reijsbergen, Pieter-Tjerk de Boer, Werner Scheinhardt, and Boudewijn Haverkort. 2013. Automated Rare Event Simulation for Stochastic Petri Nets. In *Quantitative Evaluation of Systems*, Kaustubh Joshi, Markus Siegle, Mariëlle Stoelinga, and Pedro R. D'Argenio (Eds.). Lecture Notes in Computer Science, Vol. 8054. Springer Berlin Heidelberg, 372–388. DOI:http://dx.doi.org/10.1007/978-3-642-40196-1_31

C. P. Robert and G. Casella. 2005. *Monte Carlo Statistical Methods*. Springer-Verlag New York.

R.Y. Rubinstein and D.P. Kroese. 2008. *Simulation and the Monte Carlo method (Series in Probability and Statistics)*. Vol. 707. Wiley.

Shlomo S Sawilowsky. 2003. You think you?ve got trivials? *Journal of Modern Applied Statistical Methods* 2, 1 (2003), 21.

R. Segala. 1995. *Modelling and verification of randomized distributed real time systems*. Ph.D. Dissertation. Massachusetts Institute of Technology.

K. Sen, M. Viswanathan, and G. Agha. 2005a. On statistical model checking of stochastic systems. In *Proc. CAV'05*. Springer, 266–280.

K. Sen, M. Viswanathan, and G. Agha. 2005b. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *QEST'05*. IEEE, 251–252.

Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. 2009. PAT: Towards Flexible Verification under Fairness. *Proceedings of the 21th International Conference on Computer Aided Verification (CAV'09)* 5643 (2009), 709–714.

M. Vardi. 1985. Automatic verification of probabilistic concurrent finite state programs. In *SFCS 1985*. IEEE, 327–338.

Manuel Villén-Altamirano and José Villén-Altamirano. 1994. RESTART: a straightforward method for fast simulation of rare events. In *Proc. WSC'94*. San Diego, USA, 282–289.

Zbigniew I. Woźnicki. 2001. On performance of {SOR} method for solving nonsymmetric linear systems. *J. Comput. Appl. Math.* 137, 1 (2001), 145 – 176. DOI:http://dx.doi.org/10.1016/S0377-0427(00)00705-6

H. Younes. 2005. Ymer: A statistical model checker. In *Computer Aided Verification*. Springer, 171–179.

H. Younes, E. Clarke, and P. Zuliani. 2011. Statistical Verification of Probabilistic Properties with Unbounded Until. *Formal Methods: Foundations and Applications* (2011), 144–160.