Steady-state dependability verification for very large systems

Diana El Rabih   Gael Gorgo   Nihal Pekergin   Jean-Marc Vincent

September 2010
TR–LACL–2010–9
Laboratory of Algorithmics, Complexity and Logic (LACL)
University Paris 12 (Paris Est)


D. ELRABIH, G. GORGO, N. PEKERGIN, J.M. VINCENT

Steady-state dependability verification for very large systems

© Diana ELRABIH, Gael GORGO, Nihal PEKERGIN, Jean-Marc VINCENT Septembre 2010.
Steady-state dependability verification for very large systems *

Diana El Rabih 1  Gael Gorgo 2  Nihal Pekergin 1  Jean-Marc Vincent 2

1 LACL, University of Paris-Est (Paris 12),
61 avenue Général de Gaulle 94010, Créteil, France

2 LIG, University of Grenoble (Joseph Fourrier),
51, av. Jean Kuntzmann, 38330 MONTBONNOT, France

delrabih@univ-paris12.fr, gael.gorgo@imag.fr,
nihal.pekergin@univ-paris12.fr, jean-marc.vincent@imag.fr

Abstract

Probabilistic model checking can be done either by numerical analysis or by simulation and statistical methods. In the paper presented in [12], we have compared the efficiency of the numerical model checking implemented in PRISM tool and our statistical model checking approach which combines perfect sampling and statistical hypothesis testing to study the steady-state properties of large Markovian models. In this research report we extend this work by considering also the statistical module of the MRMC model checker. Therefore the comparisons are established between the numerical model checker PRISM, the statistical module of MRMC and our statistical verification engine included to the perfect sampler Ψ 2 . In fact, we compare the efficiency and the scalability of these approaches when they are applied to the verification of steady-state properties of very large models. We show that our statistical approach using perfect sampling is generally more efficient than the two other approaches and it allows us to consider very large models and to verify rare event properties efficiently.

Extension of recent work

1. We have enriched the experimental studies. This is done by considering a larger set of input parameters. We have also considered a tandem queue with 10 queues essentially for the comparison with MRMC statistical model checking module.

2. We have included a section to briefly introduce the new considered tools (MRMC tool) (subsection 4.2) and a section to illustrate the tools differences (subsection 4.4).

3. We have included a subsection (subsection 3.3) for the complexity and the applicability of the proposed approach. The section for the brief introduction of tools have been extended and MRMC is included. Moreover the paper is reorganised for the sake of readability and sections are extended to make it self-contained (subsection 3.3.1).

1 Introduction

Model Checking is a technique for automated verification of software, hardware and network systems. It has been introduced to verify functional properties of systems expressed in a

*This work is supported by a french research project CHECKBOUND, ANR-06-SETI-002
formal logic like Computational Tree Logic (CTL). It is done by accepting as input system models and the properties or specifications that the final system is expected to satisfy and by giving outputs Yes if the model satisfies given specifications and No otherwise. Probabilistic model checking is an extension for the formal verification of systems exhibiting stochastic behavior. The system model is usually specified as a state transition system, with probability values attached to transitions, like for example Markov chains. A wide range of quantitative performance, reliability, and dependability measures can be specified by means of temporal logics: PCTL for Discrete Time Markov Chains (DTMC) and CSL for Continuous Time Markov Chains (CTMC) [3]. Dependability verification of computerised systems is gaining considerable importance in daily life. Ensuring minimum breakdown probabilities of, for instance, components of real-life computer systems, is vital to fulfil safety requirement specifications. The study of the worst case risk to hit by a safety-critical system at equilibrium in the system design process is becoming indispensable. In fact, to ensure that a system design meets its safety requirement specification, one method consists of creating and analysing a formal model of the envisaged design by using model checking techniques. The underlying stochastic models which are usually Markov chains is defined by high-level formalisms such as stochastic Petri nets, stochastic process algebras, and queueing networks. We consider queueing networks as high level formalism for the case studies in this research report.

There are two distinct approaches to perform probabilistic model checking: Numerical techniques based on computation of transient-state or steady-state distributions of the underlying Markov chain [2] and statistical techniques applied on samples obtained by means of discrete event simulation or by measurement. In fact, numerical approach is highly accurate but it suffers from state space explosion problem while statistical approach can overcome the state space explosion problem but it provides verification results with probabilistic guarantees of correctness. Thus statistical model checking techniques can be seen as an alternative to numerical techniques and they can be applied when it is infeasible to use numerical techniques. In the last years, different statistical model checkers have been proposed [16, 20, 7] especially for properties specified by time-bounded until formulas. Moreover, the statistical model checker MRMC [9] has been proposed and extended to support CSL steady-state formula. For this formula the probability is estimated based on steady-state simulation of bottom strongly connected components (BSCCs) and estimates for the probabilities to reach those BSCCs. On the other hand, the PRISM model checker [8] is largely used. It makes use of symbolic data representation in order to reduce memory requirements for numerical techniques and it supports the verification of CSL steady-state formula.

In [6] numerical and statistical techniques have been compared when they are applied to the verification of time-bounded until formulas in the temporal stochastic logic CSL. In [12], we have compared the efficiency of the numerical model checking implemented in PRISM tool [8] and our statistical model checking approach proposed in [15, 14] which combines perfect sampling and statistical hypothesis testing to study the steady-state properties of large Markovian models. In this research report, we extend this comparison by considering also the statistical module of the MRMC model checker. We have also enriched experimental studies. This is done by considering a larger set of input parameters. We have also considered a tandem queue with 10 queues essentially for the comparison with MRMC statistical model checking module. The significant advantage of perfect sampling is that it permits to analyse rare event probabilities efficiently and it provides an unbiased sampling of the steady-state distribution, hence the accuracy of the verification only depends on the statistical testing. In other words, we ensure the correctness of our results considering a specified precision level.

This research report is organized as follows: Section 2 briefly presents the perfect sam-
pling. In section 3 we present our proposed approach based on perfect sampling. We give a brief introduction of the studied tools in section 4. Section 5 is devoted to the case studies. First we present the models and their validation. Next, we compare and analyze the results of our experiments. Finally, in section 6 we summarize the conclusions and provide the future works.

2 Perfect Sampling

Let $\{X_n\}_{n \in \mathbb{N}}$ be an irreducible and aperiodic discrete time Markov chain with a finite state space $\mathcal{X}$ and a transition matrix $P = (p_{i,j})$. Let $\pi$ denote the steady-state distribution of the chain: $\pi = \pi P$. The evolution of the Markov chain can always be described by a stochastic recurrence sequence

$$X_{n+1} = \eta(X_n, e_{n+1})$$

with $\{e_n\}_{n \in \mathbb{N}}$ an independent and identically distributed sequence of events $e_n \in \mathcal{E}$, $n \in \mathbb{N}$. The transition function $\eta : \mathcal{X} \times \mathcal{E} \to \mathcal{X}$ verifies the property that $\mathbb{P}(\eta(i,e) = j) = p_{i,j}$ for every pair of states $(i,j) \in \mathcal{X} \times \mathcal{X}$ and a random event $e \in \mathcal{E}$. An execution of the Markov chain is defined by an initial state $x_0$ and a sequence of events $\{e_n\}_{n \in \mathbb{N}}$. The sequence of states $\{x_n\}_{n \in \mathbb{N}}$ defined by equation (1) is called a trajectory.

<table>
<thead>
<tr>
<th>Algorithm 1 Backward monotone steady-state property-sampling of a Markov chain</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Require:</strong></td>
</tr>
<tr>
<td>$\eta$ a monotone transition function</td>
</tr>
<tr>
<td>${e_n}_{n \leq 0}$ a backward event process</td>
</tr>
<tr>
<td>$\mathcal{M} \subseteq$ the set of extremal elements (Max and Min) of the state space $\mathcal{X}$</td>
</tr>
<tr>
<td>$r_\varphi$ the reward function associated to the checked property $\varphi$</td>
</tr>
<tr>
<td><strong>1:</strong> $n \leftarrow 0$</td>
</tr>
<tr>
<td><strong>2:</strong> repeat</td>
</tr>
<tr>
<td><strong>3:</strong> {Start from the past at time $-2^n$ }</td>
</tr>
<tr>
<td><strong>4:</strong> $\mathcal{Z} \leftarrow \mathcal{M} \subseteq$</td>
</tr>
<tr>
<td><strong>5:</strong> for $i = -2^n + 1$ downto 0 do</td>
</tr>
<tr>
<td><strong>6:</strong> $\mathcal{Z} \leftarrow \eta(\mathcal{Z}, e_i)$ ;</td>
</tr>
<tr>
<td><strong>7:</strong> end for</td>
</tr>
<tr>
<td><strong>8:</strong> ${ \eta^{(2^n)}(\mathcal{X}, e_{-2^n+1 \rightarrow 0}) \subseteq \mathcal{Z}$ the bounding set of all possible states at time 0 knowing the event process starting at time $-2^n }$</td>
</tr>
<tr>
<td><strong>9:</strong> $n = n + 1$</td>
</tr>
<tr>
<td><strong>10:</strong> until $</td>
</tr>
<tr>
<td><strong>11:</strong> return $r_\varphi(\mathcal{Z})$ { $r_\varphi(\mathcal{Z})$ is reduced to one value, 0 or 1 }</td>
</tr>
</tbody>
</table>

Trajectories are generated with the same sequence of events $\{e_n\}_{n \in \mathbb{N}}$. If at time $t$, two trajectories are in the same state, we say that they couple. Propp and Wilson [11] have introduced the perfect/exact sampling method which is based on a backward coupling, also called coupling from the past: by coming from a distant time $-\tau$ sufficiently far in the past, if all trajectories (trajectories that come from all possible initial states in $\mathcal{X}$ at time $-\tau$) are coupled in one state at time 0, then the sampled state is exactly distributed according to the stationary distribution. The backward coupling provides steady-state sample in a controlled finite number of steps, that could not be obtained by a forward coupling scheme unless the model have a strong stationary time which is rare in our examples [10].
The backward coupling called also coupling from the past is especially efficient when the underlying system is monotone. We first give the definition of monotone systems.

**Definition 2.1** Given a partial order $\preceq$ on $X$, an event $e$ is said to be monotone if it preserves the partial ordering $\preceq$ on $X$. That is

$$\forall (x, y) \in X \quad x \preceq y \Rightarrow \eta(x, e) \preceq \eta(y, e)$$

If all events are monotone, the global system is said to be monotone.

According to an order $\preceq$ on $X$, there exists a set $M_{\preceq} \subset X$ of extremal states (maximal and minimal states). When a Markov chain is monotone, all trajectories issued from $X$ are always bounded by trajectories issued from $M_{\preceq}$. Thus, it is sufficient to compute trajectories issued from $M_{\preceq}$ since when they couple, global coupling also occurs. As the size of $M_{\preceq}$ is usually drastically smaller than the size of $X$, monotone perfect sampling [11] significantly improves the sampling time.

Efficiency of simulations is also improved by functional perfect sampling [19]. The algorithm sample a reward value, according to a user defined reward function $r : X \rightarrow \mathbb{R}$: The algorithm then stops when all trajectories are in a set of states at time 0 that belong to the same reward value (going further in the past will inevitably couple in a state that belong to this reward value). To combine monotone and functional perfect sampling, the reward function $r$ must be monotone, that is $x \preceq y \Rightarrow r(x) \preceq r(y)$. As $|\mathcal{R}|$ is smaller than $|X|$, this technique may lead to an important reduction of the coupling time.

Algorithm 1 is the perfect sampling algorithm that we use in this research for steady-state property verification. It combines the monotone and functional techniques explained above. The main loop follows a doubling period scheme to find a time $-\tau$ sufficiently far so that coupling occurs (Propp and Wilson have shown that doubling period scheme provides a better mean complexity). In a property verification context, we focus on reward functions that correspond to properties we want to check, so that $\mathcal{R} = \{0, 1\}$. In Figure 1, we show an illustration of the behaviour of algorithm 1. In this example, like in case studies of section 5, the set of extremal states is composed of one maximum and one minimum; $M = \{\text{Max, Min}\}$.

Note that, as the reward function is monotone, values 0 and 1 cover contiguous zones of the state-space. Then, an interesting phenomenon happens when the property to be checked has a small set of positive states $\{x \in X | r_\varphi(x) = 1\}$ ($\varphi$ corresponds to a rare property / event): coupling frequently occurs in reward value 0 and the coupling time is very short (the time needed by the trajectory issued from Max to decrease until it leaves the "positive zone"). Moreover, if $|\{x \in X | r_\varphi(x) = 1\}|$ does not depend on $|X|$ (case of saturation properties for example), then the performance of algorithm 1 will be as good for very large state-spaces as for small ones. This intuition is validated by results of section 5.

### 3 Statistical Model Checking by Perfect Sampling

Recently, we have proposed to statistically check steady-state properties by means of perfect sampling and hypothesis testing [14, 15]. However the consideration of the unbounded until formula by this approach [15] needs more work. In this section, we give some preliminaries about the temporal logics of Markov chains and about the principle of statistical hypothesis testing. Next in this section, we briefly present our verification procedure by describing how sample paths are generated and tested for the statistical model checking of the steady-state formula and we discuss the applicability and the complexity of our statistical verification procedure.
3.1 Temporal logics for Markov chains

In this subsection we give a brief introduction for the considered temporal logics operators. The stochastic behaviour of the underlying system is described by a labelled Markov chain, $\mathcal{M}$, which may be a discreet-time Markov chain (DTMC) or a continuous-time Markov chain (CTMC). The CSL (resp. PCTL) is a branching-time temporal logic with state and path formulas over CTMCs (resp. DTMCs) [2]. Thus it is useful to specify performance and dependability measures as logical formulas over Markov chains.

We consider indeed a labelled Markov chain $\mathcal{M}$ defined over a finite state space $\mathcal{X}$. Let $AP$ denote the finite set of atomic propositions. $L : \mathcal{X} \rightarrow 2^{AP}$ is the labeling function which assigns to each state $s \in \mathcal{X}$ the list of atomic propositions satisfied in this state. Intuitively speaking, when the system is in state $s$, the properties defined by the set of atomic propositions $L(s)$ assigned to this state are satisfied. The satisfaction operator is denoted by $|=$. For all state $s \in \mathcal{X}$, $s |= true$. Atomic proposition $a$ is satisfied by state $s$ ($s |= a$) iff $a \in L(s)$.

In this research report, we consider essentially the CSL (resp. PCTL) steady-state operator for long run behaviours of the underlying model. The steady-state operator (formula) $S_{\geq \theta}(\varphi)$ (resp. $L_{\geq \theta}(\varphi)$) allows us to analyze the long-run behaviour of the system where $\theta$ is a probability threshold, $\geq \in \{<,,>,\leq,\geq\}$, $\varphi$ is a state formula.

Computing whether $s |= S_{\geq \theta}(\varphi)$ amounts to solving a system of linear equations combined with graph analysis methods, namely a search for all bottom strongly connected components (BSCC), which takes $O(|\mathcal{X}| + |E|)$ time where $|\mathcal{X}|$ denotes the number of states in a CTMC (resp. DTMC) and $|E|$ denotes the number of transitions with non-zero probability. A steady-state analysis is performed for each BSCC, after which the probabilities of reaching the individual BSCCs are computed. If the sum of steady-state probabilities of states satisfying $\varphi$ meets $\theta$, this operator is satisfied. According to [8], steady-state analysis can be performed with a worst case time complexity of $O(|\mathcal{X}|^3)$. In this work, we consider ergodic Markov chains $\mathcal{M}$, hence there is a unique steady-state distribution independent of the initial state. Note that for a CSL (resp. PCTL) steady-state formula, the verification for example of $S_{\geq \theta}(\varphi)$ is the same as $S_{\leq 1-\theta}(\neg \varphi)$ and also is the same as $\neg S_{<\theta}(\varphi)$.
3.2 Statistical verification of steady-state properties

For the statistical verification, we have as input parameters: the model defined by a labelled Markov chain $M$, the property $\varphi$ (to be verified on each sample). Formally, the CSL (resp. PCTL) steady-state property is specified by $\psi = S_{\geq \theta}(\varphi)$. Moreover, the threshold $\theta$, the indifference region $\delta$, the statistical hypothesis testing strength $(\alpha, \beta)$ are the other input parameters. We propose to apply functional perfect sampling (Algorithm 1, Figure 1), so at time 0, we test if the rewards are coupled at reward 0 or 1. In other words, we test if it is a positive or negative sample. Thus we associate the reward $r_\varphi(x)$ to each state $x \in \mathcal{X}$ for the given property $\varphi$:

$$
\begin{align*}
  r_\varphi(x) &= 1, \text{ if } x \models \varphi \\
  r_\varphi(x) &= 0, \text{ otherwise } x \not\models \varphi
\end{align*}
$$

The statistical decision method that we use is inspired from the Single Sampling Plan (SSP) method [22]. In fact, this decision method counts the number of positive samples obtained in the considered sample size $n$, when we test if $\varphi$ is verified (positive sample) or not (negative sample) on each generated sample path by using functional perfect sampling. Then this decision method provides decision either Yes if the obtained number of positive samples is greater or equal to the acceptance threshold $m$ ($\varphi$ is satisfied) or No otherwise ($\varphi$ is not satisfied).

3.2.1 Hypothesis Testing

Let $p$ be the probability that the system satisfies the underlying property $\varphi$, then the verification problem of $\psi = S_{\geq \theta}(\varphi)$ can be formulated as an hypothesis testing problem: $H : p \geq \theta$ against the alternative hypothesis $K : p < \theta$. In fact, the strength of the statistical test was determined by two parameters: $\alpha$ and $\beta$, where $\alpha$ is a bound on the probability of accepting $K$ when $H$ holds (known as a type I error, or false negative) and $\beta$ is a bound on the probability of accepting $H$ when $K$ holds (a type II error, or false positive). In practice, two thresholds, $p_0$ and $p_1$ are defined in terms of the probability threshold, $\theta$, and the half-width $\delta$ of the indifference region: $p_0 = \theta + \delta$ and $p_1 = \theta - \delta$. Then instead of testing $H : p \geq \theta$ against $K : p < \theta$, we test $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$. Suppose that we have generated $n$ samples, and a sample $X_i$ is a positive sample ($X_i = 1$) if it satisfies $\varphi$ and negative ($X_i = 0$) otherwise. $X_i$ is a random variable with Bernoulli distribution with parameter $p$. Thus the probability to obtain a positive sample is $p$. In this research report, we only explain the Single Sampling Plan (SSP) decision method for statistical hypothesis testing, that we apply in the case studies when using $\Psi^2$.

**Single Sampling Plan (SSP):** It is based on the acceptance sampling with fixed sample size and with a given acceptance strength $(\alpha, \beta)$. If $\sum_{i=1}^{n} X_i \geq m$, then $H_0$ is accepted otherwise $H_1$ is accepted, where $m$ is the acceptance threshold. The hypothesis $H_1$ will be accepted with probability $F(m, n, p)$ and the null hypothesis $H_0$ will be accepted with the probability $1 - F(m, n, p)$, where $F(m, n, p)$ is a binomial distribution: $F(m, n, p) = \sum_{i=1}^{m} \binom{n}{i} p^i (1-p)^{n-i}$ with $\binom{n}{i}$ is the combination of $i$ from $n$. It is required that the probability of accepting $H_1$ when $H_0$ holds is at most $\alpha$, and the probability of accepting $H_0$ when $H_1$ holds is at most $\beta$. These constraints can be illustrated as below:

- $\Pr[H_1 \text{ is accepted } | \ H_0 \text{ is true}] \leq \alpha$ which implies $F(m, n, p_0) \leq \alpha$  \hspace{0.5cm} (C1)
Pr[H_0 is accepted | H_1 is true] \leq \beta \text{ which implies } 1 - F(m, n, p_1) \leq \beta \quad (C2)

The number of samples \( n \) and the acceptance threshold \( m \) must be chosen under these constraints and can be determined as explained in the following [22]:

The constraints (C1) and (C2) can be expressed as given in [21, 22] where \( \phi^{-1} \) is the inverse cumulative distribution function for the standard normal distribution:

- \( \frac{m - np_0}{\sqrt{(np_0(1-p_0))}} = \phi^{-1}(\alpha) \quad (E1) \)
- \( - \frac{m - np_1}{\sqrt{(np_1(1-p_1))}} = \phi^{-1}(\beta) \quad (E2) \)

Then by adding (E1) and (E2), the approximation formula of \( n \) to optimise performance can be derived as follows:

\[
n = \frac{(\phi^{-1}(\alpha)\sqrt{(p_0(1-p_0))} + \phi^{-1}(\beta)\sqrt{(p_1(1-p_1))})^2}{(p_0-p_1)^2} \quad (E3)
\]

From [21, 22] the following approximation formula for the inverse normal cumulative distribution function with \( v = \sqrt{(-\log \alpha^2)} \) can be used as follows:

\[
\phi^{-1}(\alpha) \approx -v + \frac{a_0 + a_1v}{1 + b_1v + b_2v^2}
\]

where \( a_0 = 2.30753, b_1 = 0.99229, a_1 = 0.27061, b_2 = 0.04481 \).

After computing the sample size \( n \) using (E3), the acceptance threshold \( m \) can be computed by using either (E1) or (E2).

Note that, the acceptance threshold can be approximated to \( m = \lfloor n.\theta \rfloor \) where \( \theta \) is the threshold of the considered formula.

### 3.3 Applicability and complexity of proposed approach

Perfect simulation can be applied to monotone systems as proposed in [19, 18] and to non monotone systems as proposed in [1]. It can be applied to queueing networks, Markov chains [19, 18] and to stochastic automata networks [4]. Therefore perfect sampling can be applied to monotone and to non monotone systems modeled by Markov chains, queueing networks or stochastic automata networks. Moreover the functional perfect sampling is particularly useful in the model checking since we do not look for the exact values but only for the satisfaction or not of a given property.

The complexity of our approach is related to statistical model checking complexity which depends on the computed sample size, on the perfect simulation effort (or complexity) and on the trajectory length in case of path formulas. The computed sample size depends on applied hypothesis testing decision method. In this work, we use the Single Sampling Plan decision method. Then the sample size will be computed using the approximation formulas given in [22]. It has been shown that if the complexity of the backward simulation algorithm in the number of transition function evaluation is bounded by \( |\mathcal{X}|.(2.\mathbb{E}\tau^*) \) [19], then the mean time complexity \( \mathbb{C} \) can be bounded by \( \mathbb{C} \leq |\mathcal{X}|.(2.\mathbb{E}\tau^*).c_\eta \) where \( \tau^* \) is the coupling time of the backward scheme (coupling from the past scheme), \( |\mathcal{X}| \) is the state space size, \( c_\eta \) is the mean computation cost of \( \eta(x, e) \). The memory complexity (storage of the set of generated events) is bounded by \( 2\mathbb{E}\tau^* \) and could be reduced to \( E\log_2\tau^* + 1 \) [19].

Thus the complexity of the perfect simulation is clearly modest in case of monotone systems because we consider only maximal and minimal states as initial values among all states.
In fact, the simulation time reduction is proportional to the size of the state space in this case [18], which is usually very large. It has been shown in [18] when studying coupling time distribution, that simulation times per sample for the studied queueing networks examples are just a few milli-seconds on a standard PC. In case of non monotone systems, it has been shown in [1] that there is only a need to compute two trajectories: an infimum and supremum envelopes, then the complexity of perfect simulation in this case will be also modest. Moreover, because of the independence of generated samples, the perfect simulation method can be parallelized efficiently [4].

4 Probabilistic Model Checking Tools

We now give a brief presentation of the considered tools.

4.1 PRISM: Probabilistic Symbolic Model Checker

PRISM [8] is a largely used probabilistic model checker developped at the University of Oxford (http://www.prismmodelchecker.org/). It supports three types of probabilistic models: discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs) and Markov decision processes (MDPs). PRISM has been used to analyse systems from a wide range of application domains, including communication and multimedia protocols, randomised distributed algorithms, security protocols, biological systems and many others. Models are described using the PRISM language, a high level language. PRISM supports automated analysis of a wide range of quantitative properties of these models. In fact, the property specification language incorporates the temporal logics PCTL, CSL.

PRISM is a based on numerical techniques solving a system of linear equations combined with graph analysis methods, namely a search for all bottom strongly connected components (BSCC). It also features discrete-event simulation functionality for generating approximate results to quantitative analysis but it does not support steady-state properties.

In addition, PRISM incorporates symbolic data structures and algorithms used for state space representation, based on BDDs (Binary Decision Diagrams) and MTBDDs (Multi-Terminal Binary Decision Diagrams). For numerical computation, PRISM includes three separate engines making varying use of symbolic methods. These engines use different data structures: The first engine generates an MTBDD to represent the transition matrix, the sparse engine permits to convert the transition matrix to a sparse matrix. The hybrid engine is generally faster than MTBDD one, and while handling larger systems is expected to be faster and to consume less memory than sparse matrices, and hence is the one used in this research report. The user interface and parsers of PRISM are written in Java; the core algorithms are mostly implemented in C++.

As input the PRISM engine takes the model file (DTMC, CTMC) and the property specification file (PCTL, CSL). This engine performs numerical computation of the probability \( p \) that we look for by solving a numerical equations system (Jacobi, Gauss siedel methods \( \cdots \)), then it compares the computed probability \( p \) to the property threshold, \( \theta \) and then it generates an output file containing the model checking decision (True, False) and a log file containing the model checking time (VT) in seconds and the memory consumption (VSZ) in Kbytes.
4.2 MRMC: Markov Reward Model Checker

MRMC [9] stands for Markov Reward Model Checker developed at the University of Twente (http://www.mrmc-tool.org/trac/). It has first implemented numerical model-checking techniques for DTMC and CTMC models. It is a command-line tool implemented in C. MRMC represents the state space by sparse matrices. In fact, it uses simple but high-performance data structures, such as: a slightly modified version of the well-known compressed-row, compressed-column representation of probability (rate) matrices, and bit vectors for representing sets of states. Since v1.4.1, it has a full support for the statistical model checking of CSL properties on CTMC models.

For the steady-state formulae the probability is estimated based on steady-state simulation (based on the regeneration method) [9] of bottom strongly connected components (BSCCs) and estimates for the probabilities to reach those BSCCs. In fact, the formula is verified by estimating its probability using a confidence interval of desired width and then by comparing it against the formula probability bound. MRMC does not employ standard sequential confidence interval but rather emulates it by gradually increasing the sample size.

MRMC model and property files can be generated from the transformation of the PRISM model and property files, by using the exports options of the PRISM tool version 3.0. As input the MRMC engine takes the transformed model file (probability or rate matrix of a DTMC, CTMC) and the property specification file (state labeling with atomic propositions). This engine performs statistical estimation of the probability $p$ using confidence intervals, then it compares the computed probability $p$ to the property threshold, and then it generates an output file containing the model checking decision (True, False) and a log file containing the model checking time (VT) in seconds and the memory consumption (VSZ) in Kbytes.

4.3 Ψ²: Perfect Sampler 2

Ψ² [18], a performance evaluation software developed by MESCAL INRIA/LIG team of the University of Grenoble (http://psi.gforge.inria.fr/website/Psi2-Unix-Website/Introduction.html), allows us to estimate steady-state properties of various finite capacity queueing networks. Based on Algorithm 1 and written in C, it builds independent samples of stationary rewards of the underlying CMTC. Queueing systems description is based on an events library which is continually improved by the MESCAL team.

Typical events of queueing networks are client arrivals, end of services, routing from one
Figure 3: The MRMC Tool
queue to another, negative clients, and batch means arrivals. Complex routing strategies had been captured in a common framework, based on index functions [17], so that a large scope of monotone queueing networks can be studied. Moreover, the sandwiching principle of monotone backward coupling had been generalized to non-monotone queueing networks (envelopes techniques [1]) and implemented in Ψ². This tool has been used in various domains, including networks dimensioning, telecommunications systems, resource brokering problems, etc. Ψ² is well suited for probabilistic model checking, and in particular for steady state formula verification, since it provides an unbiased sampling of stationary rewards and guarantees independence of samples. It is specifically suited for rare event probability estimation, as was already done in [17].

As input the Ψ² engine takes a model file which describes a queueing network model in a simple dedicated language and a property specification file which contains the reward function. The model file gives the number of queues and their size and the events that describe the system dynamics (arrivals of clients in a queue, routing from an origin queue to a destination queue, end of services of clients etc...). This way to model a Markovian system has the advantage to stay light in memory when the model scales (the number of events stays the same in function of the capacities of queues and increases linearly in function of the number of queues).

This engine performs a statistical hypothesis testing for the computed probability \( p \) (number of positive samples over the total number of samples). In fact, it compares the computed number of positive samples to the given acceptance threshold and it generates then an output file containing the model checking decision (True, False) and a log file containing the model checking time (VT) in seconds.

We present in Figure ??, a simplified architecture of the Ψ² tool.

![Figure 4: The Ψ² Tool](image)

### 4.4 Tools Differences

A) Implementation: PRISM is developed using a combination of Java and C++. Its user interface and parsers are written in Java. The core algorithms are mostly implemented in C++. On the other hand, MRMC (version 1.4.1) and Ψ² are command-line tools, written in C. Thus, there is not an expected efficiency difference at the implementation level. Moreover, PRISM supports the Windows, Linux, MAC OS platforms whereas MRMC currently supports the Linux platform only and Ψ² supports the Linux, MAC OS platforms.
B) Model and specification: For PRISM, system models are described using the PRISM programming language, which is a high-level state-based description language. In this language a system is described as the parallel composition of a set of modules. A module state is determined by a set of finite-range variables and its behaviour is given using a guarded-command based notation. Communication between modules takes place either via global variables or synchronisation over common action labels. PRISM is able to export models in many different formats, including the ones accepted by MRMC. The PRISM model description is translated (by the tool) into one of the three supported probabilistic models (DTMC, CTMC, MDP). In PRISM, properties can be specified using PCTL (for DTMCs and MDPs) or CSL (for CTMCs). It is possible to either determine if a probability satisfies a given bound or obtain the actual value. There is also support for the specification and analysis of properties based on costs and rewards. For MRMC, input models are described in a text format where each line specifies one transition consisting of an action name, the source state, the target state, the type, and the transition rate. Input model is defined by a .tra file and the state labelling by a .lab file. MRMC supports DTMC, CTMC as input models, and their reward extensions DMRM, CMRM. In MRMC, properties can be specified using PCTL, CSL temporal logics and their reward extensions PRCTL, CSRL. For Ψ², input models are described in a text format (.dsc file) where lines specify number of queues, events type and number, transition consisting of an event execution. In Ψ², DTMC and CTMC are supported as input models and properties can be specified by PCTL and CSL temporal logics. Ψ² can only verify properties in the initial state of the model. Thus, the results correspond to model checking formulae in the initial state. Moreover, it does not provide the probability estimates.

C) Verification algorithms: PRISM is based on symbolic and numerical algorithms representing the state space implicitly using a compact data structure and offering a range of methods for solving a system of linear equations. The advantage of using numerical methods is their high accuracy, but the drawback is that they require a large amount of memory (caused by the state-space explosion problem). MRMC and Ψ² are based on statistical algorithms using simulation and sampling. They will then estimate if the property holds based on the generated samples. Choosing a better confidence level (lower probability of getting wrong answers) results in more samples being taken, which in turn causes the model check time of the statistical method to increase. Then this method will obviously not provide the high level of accuracy as in numerical methods, since it is statistical in nature.

D) State space representation: PRISM (Hybrid engine) use a combination of Multi Terminal Binary Decision Diagram (MTBDD) and sparse matrix for state space representation. MRMC represents the state space by sparse matrices by using simple but high-performance data structures, such as: a slightly modified version of the well-known compressed-row, compressed-column representation of probability (rate) matrices, and bit vectors for representing sets of states. In Ψ², instead of building a complete state-space and verifying properties in each state, the statistical method implemented in Ψ² uses the model description to generate sample execution paths. In fact, Ψ² unlike PRISM and MRMC has on-the-fly model generation. PRISM and MRMC accept pre-generated CTMCs, and thus their memory values should depend on the model size.
E) Statistical techniques: The statistical algorithm of MRMC implements criteria different from the acceptance criteria used in model checking by hypothesis testing used in $\Psi^2$. The difference between these two statistical techniques, is the requirement to use confidence interval of the width $\prec \delta$, whereas under the same conditions in hypothesis testing we would have to use the indifference region of the width less than only $2\delta$. This can cause MRMC model checking algorithms to require more samples than needed for the ones based on hypothesis testing used in $\Psi^2$.

G) Simulation methods: The simulation by regeneration method used in MRMC has some disadvantages over the perfect simulation used in $\Psi^2$. First, the lengths of the regeneration cycle are unpredictable. Then it is not possible to plan the simulation time beforehand. Second, finding the regeneration point is not trivial since it may require a lot of checking after every event. In fact, when the number of queues in a regenerative system increases, the regeneration points become rarer and regeneration cycles become longer. However, not all systems are regenerative. Third, many of variance reduction techniques such as antithetic variables for example cannot be used due to the variable length of the regeneration cycles. Fourth, in the regeneration method, the mean and variance estimators are biased in the sense that their expected values from a random sampling are not equal to the quantity being estimated. Finally, when using the regeneration method for analysing rare events (rare probabilities), the simulation become longer.

5 Experimental Comparison Study

We now evaluate four case studies, taken from $\Psi^2$ and PRISM benchmarks, on which we will base our performance and scalability comparison. In fact, we verify the steady-state formula for these four case studies using the numerical approach implemented in PRISM tool, the statistical approach implemented in MRMC tool, and our statistical approach implemented in $\Psi^2$ tool, by varying the problem size (state space size related to the maximal queue capacity), the precision parameters of the considered verification tools, and the number of queues (number of events). We illustrate the verification time in seconds for these case studies as a function of the maximal queue capacity (state space size) and we determine the memory limit for each case when using the verification tools. Since the considered Markovian models are ergodic (by construction), thus the steady-state probabilities are independent of the initial state. Thus, the considered steady-state formula is satisfied or not whatever the initial states. In fact, these four case studies are modelised by three formal models as described in the following.

5.1 Tandem network

This model is taken from the $\Psi^2$ benchmark. We have implemented and validated [12, 13] this model as a PRISM model. We consider $b$ finite buffers in tandem where each buffer is a M/M/1/$N_{\text{max}}$ queue (Figure 5). This tandem network is defined by an input Poisson process (rate $\lambda$) at the first stage and by an exponential service rates in each stage. Let $\mu_i$ be the service rate in stage $i$. In fact, the end of service in stage $i, 1 \leq i \leq b-1$ constitutes an arrival to stage $i+1$. The packet acceptance mechanism is the rejection: a packet which arrives to a full buffer is lost. Denote by $N_{\text{max}}$ the maximal capacity of each queue. The state space associated with this tandem network is defined by $(N_{\text{max}} + 1)^b$. Three types of events occur.
in this system: **Arrival** from exterior, **end of service** in stage $i$ and arrival to stage $i+1$, and **departure** to exterior after end of service in the last stage. The monotonicity of these events is shown in [17]. Thus this considered model is monotone. Let $N_i, 1 \leq i \leq b$ be the number of packets in buffer $i$. Thus $(N_1, N_2, \cdots, N_b)$ is a CTMC of size $(N_{\text{max}} + 1)^b$. In the sequel, we denote by $s = (n_1, n_2, \cdots, n_b)$ a state of this Markov chain. We are interested in saturation properties in the last stage and we can also conclude about availability properties. Since all earlier stages must be taken into account to compute saturation probabilities in the last buffer $b$, we must consider whole Markov chain of $(N_{\text{max}} + 1)^b$ size. Thus the numerical complexity to solve the underlying model increases rapidly with $N_{\text{max}}$. We define the following atomic proposition related to buffer $b$: *last-full* is valid if the $b$th buffer is full. Based on this atomic proposition, we check the following steady-state formula: $S < \theta$ (last-full) to check whether the probability that buffer $b$ is full in steady-state is less than $\theta$ or not.

5.2 Multistage Delta Network (MDN)

This model is taken from the $\Psi^2$ benchmark. We have implemented and validated [12, 13] this model as a PRISM model. The considered model is a delta network with $y$ stages and $z$ buffers at each stage (Figure 6). Thus the total number of queues (buffers) is $b = y \times z$. With Markovian arrival and service hypothesis, the model can be defined as a CTMC with a state vector $(N_1, N_2, \cdots, N_b)$ where $N_i$ is the number of packets in the $i$th queue. The size of the state space is $(N_{\text{max}} + 1)^b$, if the maximum queue size is $N_{\text{max}}$. We suppose an homogeneous input traffic with arrival rate $\lambda$ to the first stage and service rate is $\mu$ in each queue. The routing policy is rejection (packets are lost if the queue is full) and at the end of a service in stage $i$ the routing service rates to stage $i+1$ are $(\tau_{\text{rout}_1}, \tau_{\text{rout}_2})$ with $1 \leq i \leq y - 1$. There are events ($z$ external arrivals at the 1st stage, $z$ departures at the $y$th stage, $2 \times z$ routing events between stage $i$ and stage $i+1$ with $1 \leq i \leq y - 1$). The monotonicity of these events and thus the monotonicity of this model has been shown in [17]. State labels are defined through atomic propositions depending on the number of packets in queues. For a given $k \in \{0, \cdots, N_{\text{max}}\}$, the atomic proposition $a_i(k)$ is true if $N_i \geq k$ and false otherwise. For example, $a_i(N_{\text{max}})$ is true if the $i$th buffer is full. The underlying CTMC is labelled with these atomic propositions depending on the number of packets in queues. For a given $k \in \{0, \cdots, N_{\text{max}}\}$, we can check the saturation property in the $i$th buffer to see whether the long run saturation probability of the $i$th buffer is less than $\theta$ or not. This allows us also to check the availability property, $S > 1 - \theta$ ($\neg a_i(N_{\text{max}})$). We define the atomic proposition last-stage-full that is valid if at least a queue at the second level is saturated. Thus it is defined as the disjunction of atomic propositions $a_i(N_{\text{max}}), 4 \leq i \leq 7$. Based on this atomic proposition, we consider to verify the following steady-state formula $\psi = S < \theta$ (last-stage-full) that let us to study saturation or availability properties ($S > 1 - \theta$ (last-stage-full)).
5.3 Tandem Queueing Network with coxian phase (TQN)

This model is taken from PRISM benchmark. In [12], we have implemented this non-monotone model as a $\Psi^2$ model by using non monotone techniques (envelope function) such as defined in [1] and we have validated the correctness of our implementation. We refer to [5, 13] for more details about the implementation. The non-monotonicity of this model is shown in [12, 5, 13]. The system consists of an M/$Cox_2$/1 queue sequentially composed with an M/M/1 queue (Figure 7). Let $N_{max}$ be the maximal capacity of each queue then the state space is $O((N_{max} + 1)^2)$. Messages arrive at the first queue with rate $\lambda$, and exit the system from the second queue with rate $\kappa$. If the first queue is not empty and the second queue is not full, then messages are routed from the first to the second queue. The routing time is governed by a two-phase Coxian distribution with parameters $\mu_1$, $\mu_2$, and $a$. Here, $\mu_i$ is the exit rate for the $i^{th}$ phase of the distribution, and $1 - a$ is the probability of skipping the second phase. Let $x_i \in \{0, \cdots, N_{max}\}$, for $i \in \{1, 2\}$, denote the number of messages currently in queue $i$, and $x_{ph} \in \{1, 2\}$ denote the current phase of the Coxian distribution.

We define the atomic proposition that the system is full with the formula $sys\text{-}full = (x_1 = N_{max}) \land (x_2 = N_{max}) \land (x_{ph} = 2)$. Based on this atomic proposition, we check the following Steady-state formula: $S_{\leq \theta} (sys\text{-}full)$ to check whether the probability that the system is full in steady-state is less than $\theta$ or not.

5.4 Experimental setup

**Tools and hardware settings:** The experimental results were generated on a 1.5 GHz Intel Core 2 Duo PC running Linux, and with a 2 GB of RAM. The PRISM tool has mainly two parameters $\epsilon$: the desired convergence error or precision; $maxiternumber$: the maximal number of iterations to obtain the result with certain precision. $\Psi^2$ tool has mainly three parameters $\alpha$: the desired probability of false-positive answer for Hypothesis Testing (H.T.) ; $\beta$: the desired probability of the false-negative answer for H.T. ; $\delta$: the half width of the
indifference region. MRMC tool has mainly two parameters $\xi$: the desired confidence of the result; $\delta'$: the upper bound on the width of the considered confidence interval c.i. In fact, to match parameters of $\Psi^2$ and MRMC we take $1-\xi = \alpha = \beta$ in order to have equal bounds on probabilities of having incorrect answers. Moreover, to have a fair comparison we take $\epsilon = 2\cdot \delta$ [21] and $\delta' = \delta$.

**Timing:** In (probabilistic) model checking, two time factors are of interest: the model construction time, i.e. the time to build the internal representation from the input model, and the model checking time, i.e. the time to verify the property on the internal representation. We mainly focused on the model checking time. In our comparison study, we use the time as reported by the system command *Time* (real value).

**Verification time precision:** All experiments were repeated many times using shell scripts. An experiment consists of verifying one property on one particular model using one of the model checkers. Each experiment was repeated 20 times, except that experiments for which a single run took more than 30 minutes were repeated only three times. Thus, from the collected data (runtime), we calculated the mean and the standard deviation with 95% of confidence level. In fact, the obtained precision on the collected data (running time) will be greater or equal to $10^{-2}$. The latter is determined using Student t distribution, which takes the number of experiments into account.

**Memory consumption:** In the case of the numerical method implemented in PRISM, all experiments were run using the hybrid engine which, although not necessarily the fastest engine, in general allows the analysis of larger problems than the other engines. The limiting factor in the hybrid approach is the space required to store the iteration vector [8], then the memory is proportional to the number of states. In the case of the statistical method implemented in MRMC, the state space representation were performed by using sparse matrices (same techniques as PRISM), then the memory is also proportional to the number of states [9]. On the other hand, the memory requirements for the statistical approach implemented in $\Psi^2$ are very conservative. In principle, the current state is needed to be stored during verification, which only requires memory logarithmic in the size of the state space [18]. Then memory is never exhausted during verification when using the statistical solution method implemented in $\Psi^2$.
5.5 Experimental results

5.5.1 Tandem network verification results

For numerical application, first we consider $b = 4$ and $\lambda = \{0.5, 1.5\}$, all service rates will be state-independent with rate $\mu_i = 1$, $1 \leq i \leq b$. We give in Table 1 for $b = 4$, $\theta = 0.001$ and $\epsilon = \{10^{-3}/2, 10^{-4}\}$, the numerical verification time for the considered steady-state formula $S_{< \theta}$ (last-full) by using PRISM Hybrid engine and Jacobi iterative method. Also we give in the same table for $b = 4$, $\theta = 0.001$, $\delta = \delta' = \{10^{-3}/4, 10^{-4}/2\}$ respectively, and $1 - \xi = \alpha = \beta = 10^{-2}$ the verification time for the same steady-state formula $S_{< \theta}$ (last-full) by using statistical verification methods implemented in MRMC and in $\Psi^2$ (section 3). Next we present part of these results ($N_{\max} \leq 99$) in the Figures (a1), (a2), (b1) and (b2). In fact, for $N_{\max} = 99$ we obtain an out of memory message with PRISM and with MRMC.

Moreover, we consider $b = 10$ and for the same values of model parameters and of tools parameters as in the case of $b = 4$, we give in Table 4 the numerical verification time by using PRISM Hybrid engine and Jacobi iterative method. Also we give in the same table for $b = 10$, the verification time for the steady-state formula $S_{< \theta}$ (last-full) by using statistical verification methods implemented in MRMC and in $\Psi^2$ (section 3). Next we present part of these results ($N_{\max} \leq 6$) in the Figures (g1), (g2), (h1) and (h2). In fact, for $b = 10$ we obtain an out of memory message with PRISM and with MRMC for $N_{\max} = 6$.

In all of the tables and figures we denote by:

PRISM: numerical model checking time in seconds for the steady-state formula by using PRISM hybrid engine.

$\Psi^2$: statistical model checking time in seconds for the steady-state formula by using our verification method (section 3) implemented in $\Psi^2$ engine.

MRMC: statistical model checking time in seconds for the steady-state formula by using MRMC engine. Note that we have used MRMC with PHC options, where P (pure simulation), H (hybrid regeneration method), C (fixed sample size) options.

outm: an out of memory message in PRISM tool or in MRMC tool.

itrpr: a maximal iteration number problem (we consider maxiternumber = 100000 in PRISM tool).

5.5.2 Multistage delta network (MDN) verification results

For numerical application, we consider $y = 2$ stages and $z = 4$ buffers/stage, $\lambda = 0.9$, $\mu = 1$, $(\tau_{\text{rout1}}, \tau_{\text{rout2}}) = (0.8, 0.6)$. Note that, for $y = 4$ stages and $z = 8$ buffers/stage we have obtain efficient results by using $\Psi^2$ [15, 14] while it is not possible to do numerical model checking PRISM nor statistical model checking MRMC (memory problem for $N_{\max}=1$) in this case due to the huge state space size $O((N_{\max} + 1)^32)$.

We give in Table 2 for $\theta = 0.001$ and for $\epsilon = \{10^{-3}/2, 10^{-4}\}$, the numerical verification time for the considered steady-state formula $S_{< \theta}$ (last-stage-full) by using PRISM Hybrid engine and Jacobi iterative method. Also we give in the same table for $\theta = 0.001$, $\delta = \delta' = \{10^{-3}/4, 10^{-4}/2\}$ respectively, and $1 - \xi = \alpha = \beta = 10^{-2}$, the verification time for the same steady-state formula $S_{< \theta}$ (last-stage-full) by using statistical verification methods implemented in MRMC and in $\Psi^2$ (section 3). We present part of these results ($N_{\max} \leq 10$) in the Figures (c1), (c2), (d1) and (d2). In fact, for $N_{\max} = 10$ we obtain an out of memory message with PRISM and with MRMC.
Table 1: Tandem Network (4 buffers): Verification time as function of state space size $|\mathcal{X}|$ and of queue capacity $N_{max}$ by varying $\lambda$ for $S_{<0.001}$ (last-full)

| $N_{max}$ | $|\mathcal{X}|$ | $\lambda = 0.5$, $\mu = 1$ | $\lambda = 1.5$, $\mu = 0.8$ |
|---|---|---|---|
| | PRISM | $\Psi^2$ | MRMC | PRISM | $\Psi^2$ | MRMC |
| 1 | $1.60 \times 10^4$ | 0.004 | 4.1 | 11.82 | 0.000 | 6.8 | 53.72 |
| 2 | $8.10 \times 10^4$ | 0.005 | 5.4 | 15.12 | 0.006 | 7.1 | 26.98 |
| 5 | $1.29 \times 10^5$ | 0.03 | 9.4 | 28.14 | 0.03 | 11.5 | 41.65 |
| 9 | $1.00 \times 10^4$ | 0.32 | 14.4 | 44.74 | 0.33 | 15.8 | 59.76 |
| 17 | $1.04 \times 10^5$ | 8.67 | 24.3 | 75.34 | 9.66 | 29.1 | 97.32 |
| 29 | $8.10 \times 10^5$ | 190.6 | 38.2 | 234.1 | 209.9 | 43.4 | 313.5 |
| 31 | $1.04 \times 10^6$ | 276.6 | 42.6 | 257.3 | 304.7 | 47.1 | 486.3 |
| 56 | $1.05 \times 10^6$ | 9213 | 65.9 | 418.3 | 9424 | 71.3 | 593.1 |
| 84 | $5.22 \times 10^6$ | itrpr | 86.6 | outm | itrpr | 92.5 | outm |
| 99 | $1.00 \times 10^6$ | outm | 98.1 | outm | outm | 105.4 | outm |
| 999 | $1.0 \times 10^{12}$ | outm | 365.3 | outm | outm | 415.2 | outm |
| 9999 | $1 \times 10^{12}$ | outm | 1315 | outm | outm | 1419 | outm |

Table 2: MDN: Verification time as function of state space size $|\mathcal{X}|$ and of queue capacity $N_{max}$ by varying $(\epsilon, \delta, \delta')$ for $S_{<0.001}$ (last-stage-full)

| $N_{max}$ | $|\mathcal{X}|$ | $\epsilon = 10^{-4}/2$, $\delta = 10^{-4}/4$ | $\epsilon = 10^{-4}$, $\delta = 10^{-4}/2$ |
|---|---|---|---|
| | PRISM | $\Psi^2$ | MRMC | PRISM | PS12 | MRMC |
| 2 | $6.5 \times 10^6$ | 1.8 | 2.2 | 12.08 | 1.8 | 54.5 | 69.6 |
| 3 | $6.5 \times 10^6$ | 2.1 | 2.9 | 2.9 | 2.3 | 73.1 | 69.0 |
| 4 | $3.9 \times 10^6$ | 12.1 | 3.5 | 3.05 | 14.1 | 88.2 | 81.5 |
| 5 | $1.6 \times 10^6$ | 74.9 | 3.9 | 32.14 | 88.5 | 99.3 | 232.1 |
| 6 | $5.7 \times 10^6$ | 426 | 4.3 | 49.17 | 434 | 110 | 270.3 |
| 7 | $1.6 \times 10^7$ | 1689 | 4.6 | 131.5 | 1895 | 117 | 332.5 |
| 8 | $4.3 \times 10^7$ | 4768 | 4.8 | 586.7 | 6125 | 123 | 794.9 |
| 9 | $1.0 \times 10^8$ | 8931 | 5.0 | 1412 | 10135 | 127 | 1926 |
| 10 | $2.1 \times 10^8$ | outm | 5.2 | outm | outm | 131 | outm |
| 99 | $1.0 \times 10^{16}$ | outm | 6.8 | outm | outm | 174 | outm |
| 999 | $1 \times 10^{12}$ | outm | 6.9 | outm | outm | 175 | outm |
| 9999 | $1 \times 10^{12}$ | outm | 7.1 | outm | outm | 177 | outm |

(a1) Queue Capacity of TN4 ($\lambda = 0.5$), ($\delta = 10^{-4}/2$)
(b) Queue Capacity of TN4 ($\lambda = 1.5$, $\delta = 10^{-4}/2$)

(c) Queue Capacity of MDN ($\epsilon = 10^{-3}/2$ and $\delta = 10^{-3}/4$)

(d) Queue Capacity of MDN ($\epsilon = 10^{-4}$ and $\delta = 10^{-4}/2$)
(e1) Queue Capacity of TQN ($\epsilon = 10^{-3}/2$ and $\delta = 10^{-3}/4$)

(f1) Queue Capacity of TQN ($\epsilon = 10^{-4}$ and $\delta = 10^{-4}/2$)

(g1) Queue Capacity of TN10 ($\epsilon = 10^{-3}/2$ and $\delta = 10^{-3}/4$)
(h₁) Queue Capacity ofTN10 ($\epsilon = 10^{-3}/2$ and $\delta = 10^{-4}/2$)

(a₂) Queue Capacity of TN4 ($\lambda = 0.5$, ($\delta = 10^{-4}/2$)

(b₂) Queue Capacity of TN4 ($\lambda = 1.5$, ($\delta = 10^{-4}/2$)
(e_2) Queue Capacity of MDN ($\epsilon = 10^{-3}/2$ and $\delta = 10^{-3}/4$)

(d_2) Queue Capacity of MDN ($\epsilon = 10^{-4}$ and $\delta = 10^{-4}/2$)

(e_2) Queue Capacity of TQN ($\epsilon = 10^{-3}/2$ and $\delta = 10^{-3}/4$)
(f2) Queue Capacity of TQN ($\epsilon = 10^{-3}/2$ and $\delta = 10^{-4}/2$)

(g2) Queue Capacity of TN10 ($\epsilon = 10^{-3}/2$ and $\delta = 10^{-3}/4$)

(h2) Queue Capacity of TN10 ($\epsilon = 10^{-3}/2$ and $\delta = 10^{-4}/2$)
5.5.3 Tandem network with coxian phase (TQN) verification results

For numerical application, we consider \( \lambda = 4 \times N_{\text{max}}, \mu_1 = 2, \mu_2 = 2, a = 0.1 \) and \( \kappa = 4 \). We give in Table 3 for \( \theta = 0.001 \) and for \( \epsilon = \{10^{-3}/2, 10^{-4}\} \), the numerical verification time for the considered steady-state formula \( S_{<\theta} (\text{sys-full}) \) by using PRISM Hybrid engine and Jacobi iterative method. Also we give in the same table for \( \theta = 0.001, \delta = \delta' = \{10^{-3}/4 \text{ or } 10^{-4}/2 \} \) respectively, and \( 1 - \xi = \alpha = \beta = 10^{-2} \), the verification time for the same steady-state formula \( S_{<\theta} (\text{sys-full}) \) by using statistical verification methods implemented in MRMC and in \( \Psi^2 \) (section 3). Next we present part of these results \( (N_{\text{max}} \leq 1023) \) in Figures \( (e_1), (e_2), (f_1) \) and \( (f_2) \). In fact, for \( N_{\text{max}} = 7500 \) we obtain an out of memory message with PRISM and with MRMC.

| \( N_{\text{max}} \) | \(|\mathcal{X}| \) | \( \epsilon = 10^{-3}/2, \delta = 10^{-1}/4 \) | \( \epsilon = 10^{-4}, \delta = 10^{-4}/2 \) |
|------------------|----------------|-----------------|-----------------|
| PRISM | \( \Psi^2 \) | MRMC | PRISM | \( \Psi^2 \) | MRMC |
| 10 | \( 2.3 \times 10^2 \) | 1.7 | 1.4 | 0.08 | 1.8 | 37.1 | 12.67 |
| 50 | \( 5.1 \times 10^3 \) | 1.8 | 5.4 | 0.01 | 1.9 | 131.4 | 15.23 |
| 100 | \( 2.0 \times 10^4 \) | 3.4 | 9.8 | 0.05 | 2.6 | 250.7 | 31.56 |
| 255 | \( 1.3 \times 10^5 \) | 117.9 | 24.2 | 32.14 | 119.0 | 612.4 | 912.4 |
| 511 | \( 5.2 \times 10^5 \) | 1201 | 49.1 | 70.93 | 1208 | 1230 | 1735 |
| 1023 | \( 2.1 \times 10^6 \) | 10879 | 96.4 | 131.50 | 13105 | 1822 | 2479 |
| 5000 | \( 5.1 \times 10^6 \) | \text{itrpr} | 493.2 | 586.70 | \text{itrpr} | 12258 | 15321 |
| 7500 | \( 1.1 \times 10^7 \) | \text{outm} | 819.6 | \text{outm} | \text{outm} | 19153 | \text{outm} |
| 10000 | \( 2.1 \times 10^7 \) | \text{outm} | 925.3 | \text{outm} | \text{outm} | 23219 | \text{outm} |

Table 3: TQN: Verification time as function of state space size \(|\mathcal{X}| \) and of queue capacity \( N_{\text{max}} \) by varying \( (\epsilon, \delta, \delta') \) for \( S_{<0.001} (\text{sys-full}) \)

| \( N_{\text{max}} \) | \(|\mathcal{X}| \) | \( \epsilon = 10^{-3}/2, \delta = 10^{-1}/4 \) | \( \epsilon = 10^{-4}, \delta = 10^{-4}/2 \) |
|------------------|----------------|-----------------|-----------------|
| PRISM | \( \Psi^2 \) | MRMC | PRISM | \( \Psi^2 \) | MRMC |
| 1 | \( 1.02 \times 10^2 \) | 0.023 | 0.2 | 633.7 | 0.029 | 14.1 | 1039 |
| 2 | \( 5.90 \times 10^2 \) | 1.27 | 20.9 | 334.4 | 1.31 | 26.4 | 485.1 |
| 3 | \( 1.04 \times 10^3 \) | 42.78 | 42.1 | 854.1 | 46.06 | 47.3 | 993.7 |
| 4 | \( 9.76 \times 10^3 \) | 591.7 | 65.3 | 1429 | 661.9 | 73.5 | 1547 |
| 5 | \( 6.04 \times 10^4 \) | 5821 | 86.2 | 1810 | 6315 | 93.1 | 1976 |
| 6 | \( 2.82 \times 10^5 \) | \text{outm} | 98.8 | \text{outm} | \text{outm} | 196.1 | \text{outm} |
| 7 | \( 1.0 \times 10^6 \) | 311.9 | \text{outm} | \text{outm} | \text{outm} | 619.6 | \text{outm} |
| 99 | \( 1.0 \times 10^9 \) | \text{outm} | 2311 | \text{outm} | \text{outm} | 4926 | \text{outm} |
| 999 | \( 1.0 \times 10^{10} \) | \text{outm} | 719.6 | \text{outm} | \text{outm} | 15156 | \text{outm} |
| 9999 | \( 1.0 \times 10^{10} \) | \text{outm} | 921.3 | \text{outm} | \text{outm} | 19216 | \text{outm} |

Table 4: Tandem Network (10 buffers): Verification time as function of state space size \(|\mathcal{X}| \) and of queue capacity \( N_{\text{max}} \) by varying \( (\epsilon, \delta, \delta') \) for \( S_{<0.001} (\text{last-full}) \)

6 Discussions

Tables 1, 2, 3, and 4 and their corresponding figures represent the verification time in seconds as function of state space size (maximal queue capacity). In the four tables we also show the memory limits of PRISM and of MRMC for large state space size while our statistical method implemented in \( \Psi^2 \) tool scales better, is efficient and do not have any memory limitation for these large state space size. Our results confirm that for steady-state property, our statistical method implemented in \( \Psi^2 \) scales better with the size of the state space. Moreover it is
generally faster than the numerical method implemented in PRISM and than the statistical method implemented in MRMC. However, high accuracy comes at a greater price than for numerical method.

In fact, when comparing the efficiency of PRISM (numerical) and of Ψ² (statistical) we found that: For the smaller models (monotone and non-monotone cases), the PRISM tool is slightly faster. However, for the larger models (monotone and non-monotone cases), our statistical method implemented in Ψ² is faster than PRISM.

When comparing the efficiency of MRMC (statistical) and of Ψ² (statistical) and when comparing the scalability of the three considered tools we found that: For Tandem network (4 buffers) case, Ψ² is faster than MRMC for large state space size $|\mathcal{X}|$ (due to monotonicity of model), and we obtain an out of memory message in PRISM and in MRMC from the value $|\mathcal{X}| = 1.0 \times 10^8$ (corresponding to $N_{\text{max}} = 99$). For Multistage delta network case, MRMC is slightly faster for the small $|\mathcal{X}|$ while Ψ² becomes faster than MRMC for large $|\mathcal{X}|$ (due to the considered rare event property), and we obtain an out of memory message in PRISM and in MRMC from the value $|\mathcal{X}| = 2.1 \times 10^8$ (corresponding to $N_{\text{max}} = 10$). For Tandem network with coaxian phase case, MRMC is slightly faster for the small $|\mathcal{X}|$ while Ψ² becomes slightly faster than MRMC for large $|\mathcal{X}|$ (due to non monotonicity of model and to the small queue number), and we obtain an out of memory message in PRISM and in MRMC from the value $|\mathcal{X}| = 1.1 \times 10^8$ (corresponding to $N_{\text{max}} = 7500$). For Tandem network (10 buffers) case, Ψ² is faster than MRMC for large $|\mathcal{X}|$ (due to large queue number and to rare regeneration points), and we obtain an out of memory message in PRISM and in MRMC from the value $|\mathcal{X}| = 2.8 \times 10^8$ (corresponding to $N_{\text{max}} = 6$).

In fact, the statistical algorithm of MRMC implements a different criteria (based on confidence intervals) from the acceptance criteria used in model checking by hypothesis testing used in Ψ². This can cause MRMC model checking algorithms to require more samples than needed for the ones based on hypothesis testing used in Ψ².

Moreover, the simulation by regeneration method used in MRMC has some disadvantages over the perfect simulation used in Ψ²:

- The lengths of the regeneration cycle are unpredictable, then it is not possible to plan the simulation time beforehand
- Finding the regeneration point is not trivial since it may require a lot of checking after every event. In fact, when the number of queues in a regenerative system increases, the regeneration points become rarer and regeneration cycles become longer (Tandem Network with 10 buffers case).
- However, not all systems are regenerative
- Many of variance reduction techniques such as antithetic variables for example cannot be used due to the variable length of the regeneration cycles
- In the regeneration method, the mean and variance estimators are biased in the sense that their expected values from a random sampling are not equal to the quantity being estimated
- When using the regeneration method for analysing rare events (rare probabilities), the simulation become longer (Multistage Delta Network case)

Note that, in our experimental comparison study we have variated the precision parameters of numerical ($\epsilon$) and of statistical ($\delta'$ and $\delta$) methods. Thus we can note from the first
four tables that the numerical verification time dependence on $\epsilon$ is negligible while the statistical verification time dependence on $\delta$ and on $\delta'$ is considerable. Moreover, we refer to section 2 to explain why in some of our case studies, the obtained statistical verification time when using $\Psi^2$ does not depend on $|\mathcal{X}|$. Finally, note that even if we do not use sequential acceptance sampling which is more efficient than single one used in our statistical method implemented in $\Psi^2$, we obtain more efficient and more scalable results than other methods.

7 Conclusion and future works

In this research report, we have empirically compared numerical and two different statistical solutions implemented in PRISM, MRMC and $\Psi^2$ tools for probabilistic model checking on case studies taken from the PRISM and $\Psi^2$ benchmarks. We focused our attention on steady-state properties. For these properties, we have found that our statistical method implemented in $\Psi^2$ scales better with the state space size and it is faster than PRISM and MRMC tools especially for large models. In fact, we aim to find the limiting problem sizes for the considered case studies. We see that our statistical approach scales well with the problem size and it allows us to consider very large models. We see that our statistical approach scales well with the problem size, it is generally more efficient than the PRISM (numerical) and MRMC (statistical) approaches and it allows us to consider very large models and to verify rare event properties efficiently.

References


