Steady-state property verification: a comparison study

Diana El Rabih, LACL
University of Paris-Est Créteil- Val de Marne,
61 avenue Général de Gaulle 94010, Créteil, France
delrabih@univ-paris12.fr

Gaël Gorgo, LIG
University of Grenoble, 51, av. Jean Kuntzmann,
38330 MONTBONNOT, France
gael.gorgo@imag.fr

Nihal Pekergin, LACL
University of Paris-Est Créteil- Val de Marne,
61 avenue Général de Gaulle 94010, Créteil, France
nihal.pekergin@univ-paris12.fr

Jean-Marc Vincent, LIG
University of Grenoble, 51, av. Jean Kuntzmann,
38330 MONTBONNOT, France
jean-marc.vincent@imag.fr

Model checking of probabilistic models can be done either by numerical analysis or by simulation and statistical methods. In this paper, we compare the efficiency and the scalability of different model checking approaches when they are applied to the verification of steady-state properties of large models. We provide an experimental comparison study between the statistical model checking using perfect sampling implemented in Ψ^2 [15] and proposed in [11, 10] and the numerical method implemented in PRISM [6], for the verification of CSL [2] steady-state properties. We show that the proposed statistical approach lets us to consider very large models.

Model Checking, Probabilistic Verification, Continuous Stochastic Logic (CSL), Perfect Simulation

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 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 given 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, for example Markov Chains. ¿From this model, a wide range of quantitative performance, reliability, and dependability measures of the original system can be computed. These measures can be specified using temporal logics as PCTL for Discrete Time Markov Chains (DTMC) and CSL for Continuous Time Markov Chains (CTMC). These temporal logics are extensions of the Computational Tree Logic (CTL) [3]. The underlying stochastic models which are usually Markov chains is defined by a high-level formalisms such as stochastic Petri nets, stochastic process algebras, and queueing networks, a finite-state model of a real-life system. We consider queueing networks as high level formalism for the case studies in this paper.

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 and statistical techniques based on hypothesis testing and on sampling 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 [12, 17, 13] especially for properties specified by time-bounded until formulas. Moreover, the statistical model checker MRMC [8] has been proposed and extended to support the statistical model checking of CSL steady-state property. 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, for numerical techniques, the PRISM numerical model checker [6] 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 property.

In [5] 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 this work, we compare the efficiency of the numerical model checking implemented in PRISM tool [6] and our statistical model checking approach proposed in [11, 10] which combines perfect sampling and statistical hypothesis testing to study the steady-state properties of large Markovian models. The significant advantage of perfect sampling is that it provides an *unbiased* sampling of the steady-state distribution, hence the accuracy of the verification result only belongs to the statistical testing. In other words, we ensure the correctness of our results considering a specified precision level.

This paper is organized as follows: Section 2 briefly presents the temporal logic CSL. 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. CSL MODEL CHECKING

The Continuous Stochastic Logic (CSL) is a branching-time temporal logic with state and path formulas over CTMCs [2]. Thus it is useful to specify performance and dependability measures as logical formulas over CTMCs. We consider indeed a labelled CTMC defined over a finite state space $\mathcal{X}$. Let AP denote the finite set of atomic propositions. $L : \mathcal{X} \to 2^{AP}$ 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 $\models$, then for all state $s \in \mathcal{X}, \quad s \models true$. Atomic proposition a is satisfied by state s ($s \models a$) iff $a \in L(s)$. The logic operators are obtained using standard logic equivalence rules : $s \models \neg\varphi \ \ iff \ \ s \not\models \varphi$, $s \models \varphi_1 \wedge \phi_2 \ \ iff \ \ s \models \varphi_1 \wedge s \models \varphi_2$.

We give here only the steady-state operator that will be used in this paper. The steady-state operator (formula) $\mathcal{S}_{\bowtie\theta}(\varphi)$ lets us to analyze the long-run behaviour of the system where θ is a probability threshold, $\bowtie$ a comparison operator, for example $\bowtie \in \{<, >, \leq, \geq\}$, φ is a state formula. In this work, we consider ergodic CTMCs, hence there is unique steady-state distribution independent of the initial state. Contrary to the model checking formalism, the satisfaction property is assigned to the model but not to an inital state. We check if the underlying model M satisfies the steady-state property or not. $M \models \mathcal{S}_{\bowtie\theta}(\varphi)$, if the property specified by the steady-state operator $\mathcal{S}$ is satisfied by the model M. Note that for a steady-state property $\psi = S_{\geq\theta}(\varphi)$, the verification 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. STATISTICAL MODEL CHECKING BY PERFECT SAMPLING

In statistical model checkers, it is generally focused on the time-bounded until formulas. Recently, we have proposed to statistically check steady-state properties by means of perfect sampling and hypothesis testing [10, 11]. In this section, we briefly present how sample paths are generated and tested for the statistical model checking of the steady-state formula $\psi = S_{\geq\theta}(\varphi)$.

3.1. 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 π 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

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

with $\{e_n\}_{n\in\mathbb{Z}}$ 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 $\mathbb{P}\left(\eta(i, e) = j\right) = p_{i,j}$ 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 $\{e_n\}_{n\in\mathbb{Z}}$. The sequence of states $\{x_n\}_{n\in\mathbb{Z}}$ defined by equation (1) is called a trajectory.

Trajectories are generated with the same sequence of events $\{e_n\}_{n\in\mathbb{Z}}$. If at time t, two trajectories are in the same state, we say that they couple. Propp and Wilson [9] 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

Algorithm 1 Backward monotone steady-state property-sampling of a Markov chain

Require:
η a monotone transition function
$\{e_n\}_{n\leq 0}$ a backward event process
{This sequence is previously given or built as the execution of the algorithm}
$\mathcal{M}$ the set of extremal elements of the state space $\mathcal{X}$
r_φ the reward function associated to the checked property φ

1: $n \leftarrow 0$
2: **repeat**
3: {Start from the past at time -2^n }
4: $\mathcal{Z} \leftarrow \mathcal{M}$
5: **for** $i = -2^n + 1$ **downto** 0 **do**
6: $\mathcal{Z} \leftarrow \eta(\mathcal{Z}, e_i)$;
7: **end for**
8: { $\eta^{(2^n)}(\mathcal{X}, e_{-2^n+1\to 0}) \subset \mathcal{Z}$ the bounding set of all possible states at time 0 knowing the event process starting at time -2^n }
9: $n = n + 1$
10: **until** $|r_\varphi(\mathcal{Z})| = 1$
11: **return** $r_\varphi(\mathcal{Z})$ { $r_\varphi(\mathcal{Z})$ is reduced to one value, 0 or 1 }

from all possible initial states in $\mathcal{X}$ at time $-\tau$) are coupled before time 0 then the sampled state at time 0 is exactly distributed according to the stationary distribution.

Definition 1 *Given a partial order $\preceq$ on $\mathcal{X}$, an event e is said to be **monotone** if it preserves the partial ordering $\preceq$ on $\mathcal{X}$. That is*

$$\forall (x, y) \in \mathcal{X} \quad x \preceq y \;\Rightarrow\; \eta(x, e) \preceq \eta(y, e)$$

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

According to an order $\preceq$ on $\mathcal{X}$, there exists a set $\mathcal{M}_\preceq \subset \mathcal{X}$ of extremal states (maximal and minimal states). When a Markov chain is monotone, all trajectories issued from $\mathcal{X}$ are always bounded by trajectories issued from $\mathcal{M}_\preceq$. Thus, it is sufficient to compute trajectories issued from $\mathcal{M}_\preceq$ since when they couple, global coupling also occurs. As the size of $\mathcal{M}_\preceq$ is usually drastically smaller than the size of $\mathcal{X}$, monotone perfect sampling [9] significantly improves the sampling time.

Efficiency of simulations is also improved by functional perfect sampling [16]. The algorithm sample a reward value, according to a user defined reward function $r : \mathcal{X} \to \mathcal{R}$; 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 $|\mathcal{X}|$, this technique may lead to an important reduction of the coupling time.

Algorithm 1 is the perfect sampling algorithm that we use in this paper 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; $\mathcal{M} = \{Max, Min\}$.

Note that, as the reward function is monotone, values 0 and 1 cover contiguous zones of the state-space. Then, an interesting phenomenum happens when the property to be checked has a small set of positive states $\{x \in \mathcal{X} | r_\varphi(x) = 1\}$ (φ 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 $|\{x \in \mathcal{X} | r_\varphi(x) = 1\}|$ does not depend on $|\mathcal{X}|$ (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.

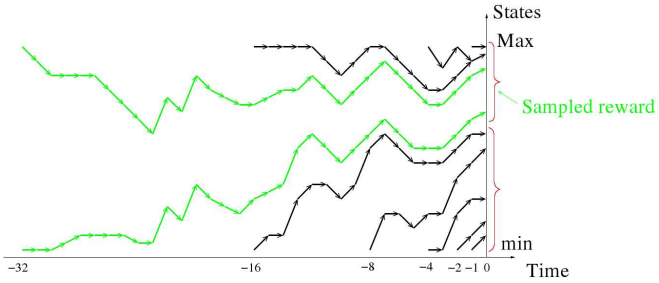

Figure 1: *Example of a backward monotone steady-state property-sampling*

3.2. Statistical hypothesis testing

The probabilistic model checking consists in deciding whether the probability that the considered system satisfies the underlying property φ meets a given threshold θ or not. Let p be the probability that the system satisfies φ, then the verification problem of $\psi = \mathcal{S}_{\geq\theta}(\varphi)$ where φ is for example the availability property of a network system, can be formulated

as an hypothesis testing: $H : p \geq \theta$ against the alternative hypothesis $K : p < \theta$. In fact, the strength of the statistical test was determined by two parameters: α and β, where α is a bound on the probability of accepting K when H holds (known as a type I error, or false negative) and β 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, θ, and the half-width δ 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 φ 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.

Single Sampling Plan (SSP): It is based on the acceptance sampling with fixed sample size and with a given acceptance strength (α, β). 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: $F(m, n, p) = \sum_{i=1}^{m} C(n, i)p^i(1-p)^{n-i}$ with $C(n, i)$ is the combination of i from n. It is required that the probability of accepting H_1 when H_0 holds is at most α, and the probability of accepting H_0 when H_1 holds is at most β. These constraints can be illustrated as below:

- Pr[H_1 is accepted $\mid$ H_0 is true] $\leq$ α which implies $F(m, n, p_0) \leq \alpha \qquad (C1)$
- Pr[H_0 is accepted $\mid$ H_1 is true] $\leq$ β which implies $1 - F(m, n, p_1) \leq \beta \qquad (C2)$

The number of samples n and the acceptance threshold m must be chosen under these constraints and their approximation formulas are given in [19].

3.3. Verification of steady-state properties

We have as input parameters: the model defined by a labelled CTMC, M, the property φ (to be verified on each sample). Formally, the steady-state property is specified $\psi = S_{\geq\theta}(\varphi)$. Moreover, the threshold parameter θ, the indifference region parameter δ, α, β for the strength of statistical hypothesis testing 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 φ:

$$\begin{aligned} r_\varphi(x) &= 1, \text{ if } x \models \varphi \\ r_\varphi(x) &= 0, \text{ otherwise } x \not\models \varphi \end{aligned} \qquad (2)$$

On the other hand, the statistical decision method we use when performing our statistical hypothesis testing on generated samples for the steady state formulas is inspired from the Single Sampling Plan (SSP) method. In SSP method the number of samples n and the acceptance threshold m will be computed by using the approximation formulas given in [19]. In fact, this decision method tests if φ is verified (positive sample) or not (negative sample) on each generated sample path, when counting the number of positive samples. Then it provides decision either *Yes* if the number of positive samples is greater or equal to m (ψ is satisfied) or *No* otherwise (ψ is not satisfied).

4. TOOLS

4.1. Probabilistic Symbolic Model Checker: PRISM

PRISM [6] is a largely used probabilistic model checker developped at the University of Birmingham (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. 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 paper. The user interface and parsers of PRISM are written in Java; the core algorithms are mostly implemented in C++. It also features discrete-event

simulation functionality for generating approximate results to quantitative analysis but it does not support steady-state properties.

We represent in Figure 2, a simplified architecture of the PRISM tool. 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, then it compares the computed probability p to the property threshold, θ 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.

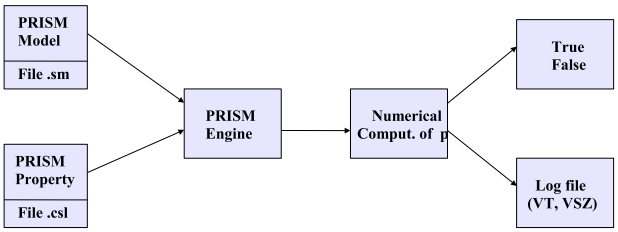

Figure 2: *The PRISM Tool*

4.2. Perfect Sampler 2: Ψ^2

Ψ^2 [15], a performance evaluation software developed by MESCAL INRIA/LIG team (http://psi.gforge.inria.fr/website/Psi2-Unix-Website/Introduction.html), lets 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. All events of this library are monotone. Typical events of queueing networks are client arrivals, end of services and routing from one queue to another. Complex routing strategies had been captured in a common framework, based on index functions [14], 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 Ψ^2. Ψ^2 has been used in various domains, including networks dimensioning, telecommunications systems, resource brokering problems, etc.

Ψ^2 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 ever done in [14].

We represent this tool in Figure 3. As input the Ψ^2 engine takes the model file (Queueing network model) and the property specification file (Reward function). This engine performs a statistical hypothesis testing for the computed probability p (number of positive samples over the total number of samples). It compares the computed probability p to the property 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.

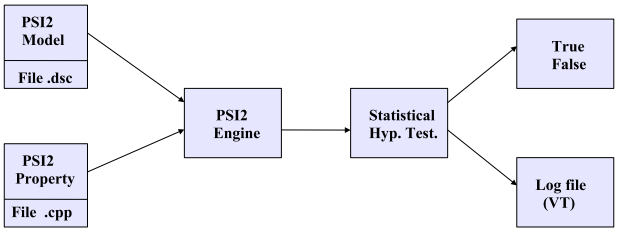

Figure 3: *The Ψ^2 Tool*

5. EXPERIMENTAL COMPARISON STUDY

We now evaluate three case studies, taken from Ψ^2 and PRISM benchmarks, on which we will base our performance and scalability comparison. In fact, we verify the steady-state formula for these three case studies using both the numerical (PRISM tool) and the statistical approach (Ψ^2 tool), by varying the problem size (state space size related to the maximal queue capacity). We illustrate the verification time in seconds for these case studies, as a function of the maximal queue capacity (state space size). Since the considered Markovian models are ergodic, thus the steady-state probabilities are independent of the initial state. Thus, the considered steady-state property is satisfied or not whatever the initial states.

5.1. Case studies

5.1.1. Tandem network

This model is taken from the Ψ^2 benchmark and we implemented it as a PRISM model. We consider b finite buffers in tandem where each buffer is a M/M/1/N_{max} queue (Figure 4). This tandem network is defined by an input Poisson process (rate λ) at the first stage and by an exponential service rates in each stage. Let μ_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_{max} the maximal capacity of each queue. The state space associated to this tandem network is

defined by $(N_{max}+1)^b$. 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 [14]. Thus this considered model is monotone.

Figure 4: *Tandem network*

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 $(N_{max}+1)^b$. 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 $(N_{max}+1)^b$ size. Thus the numerical complexity to solve the underlying model increases rapidly with N_{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:* $S_{\leq\theta}$ (*last-full*) to check whether the probability that buffer b is full in steady-state is less than θ or not.

5.1.2. Multistage Delta Network (MDN)

In telecommunication networks, multistage models are used for modelling switches. This model is taken from the Ψ^2 benchamrk and we implemented it as a PRISM model. We show the monotonicity of this model in the following. The considered model is a delta network with y stages and z buffers at each stage (Figure 5). Thus the total number of queues (buffers) is $b = y * z$. 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 then $(N_{max}+1)^b$, if the maximum queue size is N_{max}. We suppose an homogeneous input trafic with arrival rate λ to the first stage and service rate is μ 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 (τ_{rout1}, τ_{rout2}) with $1 \leq i \leq y-1$. There are events (z external arrivals at the 1^{st} 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 [14]. State labels are defined through atomic propositions depending on the number of packets in queues. For a given $k \in \{0, \cdots, N_{max}\}$, the atomic proposition $a_i(k)$ is true if $N_i \geq k$ and false otherwise. For example, a_i (N_{max}) is true if the i^{th} buffer is full. The underlying CTMC is labelled with these atomic propositions depending on the considered property. It is indeed possible to express different interesting availability and reliability measures for the underlying system by means of these atomic propositions. For instance, with formula ψ=$S_{<\theta}$(a_i (N_{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 θ or not. This lets us also to check the availability property, $S_{>1-\theta}(\neg\, a_i(N_{max}))$.

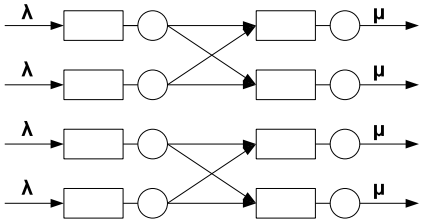

Figure 5: *Multistage Delta network*

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 $a_i(N_{max}), 4 \leq i \leq 7$. 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 ($S_{>1-\theta}(\neg$ *last-stage-full*)).

5.1.3. Tandem Queueing Network with coxian phase (TQN)

This model is taken from PRISM benchmark and we implemented it as a Ψ^2 model. The non-monotonicity of this model is shown in the following. The system consists of an M/ Cox_2 /1 queue sequentially composed with an M/M/1 queue (Figure 6). Let N_{max} to 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 λ, and exit the system from the second queue with rate κ. 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 μ_1, μ_2, and a. Here, μ_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-full* = $(x_1 = N_{max}) \wedge (x_2 = N_{max}) \wedge (x_{ph} = 2)$.

Based on this atomic proposition, we check the following *Steady-state formula:* $S_{\leq\theta}$ (*sys-full*) to check whether the probability that the system is full in steady-state is less than θ or not.

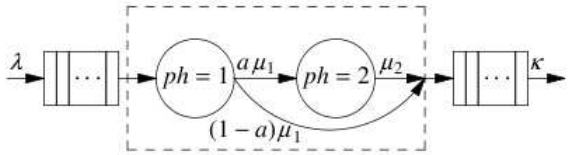

Figure 6: *Tandem queueing network with coxian phase*

Implementation of Coxian events in Ψ^2

In Ψ^2, the evolution of the state of a Markov chain is described by a sequence of events, that is arrival of a client in a queue, routing to another queue, end of service etc. In one state x of the Markov chain, an event e models a transition to another state $x' = \eta(x, e)$ where η is called the transition function of the system.

To model a Coxian server with two phases, we consider 3 events: phase 1 service d_1, phase 1 service skipping the second phase d_1' and phase 2 service d_2.

The implementation of a Coxian server in Ψ^2 then consists in encoding the transition function of these three events in the Ψ^2 event library. If the events are monotone, the transition function is directly encoded according to the dynamics of the system. However, if an event is not monotone, non monotone techniques such as defined in [1] must be involved.

Let $\mathcal{X} = \{0, \ldots, N_{max}\} \times \{1, 2\} \times \{0, \ldots, N_{max}\}$ be the state space of the system, we consider the component-wise partial ordering $\preceq$ on $\mathcal{X}$. Given two states $x, y \in \mathcal{X}$, we have $x \preceq y$ iff $x_1 \leq y_1$, $x_{ph} \leq y_{ph}$, $x_2 \leq y_2$.

Unfortunately, in the case a cox-2 server, events d_1' and d_2 are non monotone, given the following counterexamples :

$$\begin{aligned} \eta((6,2,4), d_1') &= (6,2,4) \\ \eta((1,1,4), d_1') &= (0,1,5) \\ \eta((4,2,1), d_2) &= (3,1,2) \\ \eta((4,1,0), d_2) &= (4,1,0) \end{aligned}$$

In fact, the event d_1' (resp. d_2) has no effect on a state where the server is in phase 2 (resp. phase 1), and the corresponding transition function is the identity function. On another hand, the effect of the event on a phase 1 state (resp. phase 2 state) is the move of a client from Q_1 to Q_2, which increases the load of Q_2 (resp. decreases the load of Q_1) and can result in an order inversion on Q_2 (resp. on Q_1), as in above counterexamples.

The events d_1' and d_2 has been implemented in Ψ^2 by defining an envelope function for each one. The technical details of these envelope functions are beyond the scope of this paper and it is explained in a research report [4]. The interested reader can also refer to [1] for a detailed introduction to the envelope technique. Instead, in the next section, we prove the validity of our implementation using classical goodness-of-fit tests.

N_{max} : Capacity of both the first and the second queue.
n : Number of samples.
$|\mathcal{X}|$: State-space size.
k : number of classes used for the Chi-square test.
D : Chi-square computed distance.
$\chi^2_{(0.95,k-1)}$: 95-percentiles of the Chi-square distribution with $k-1$ degrees of freedom.

N_{max}	n	$\lvert\mathcal{X}\rvert$	k	D	$\chi^2_{(0.95,k-1)}$
2	10000	15	15	13.13	23.68
5	10000	66	66	70.07	84.82
10	10000	231	21	17.98	31.41

Table 1: *Chi-square tests varying the capacity.*
$\lambda = 2$, $\mu_1 = 4$, $a = \frac{1}{2}$, $\mu_2 = 2$, $\kappa = 4$

λ	κ	n	$\lvert\mathcal{X}\rvert$	k	D	$\chi^2_{(0.95,k-1)}$
0.5	8	10000	66	6	5.04	11.07
8	0.5	10000	66	10	3.53	16.92

Table 2: *Chi-square tests varying λ and κ.*
$\mu_1 = 4$, $a = \frac{1}{2}$, $\mu_2 = 2$, $N_{max} = 5$

Model Validation

In this section, we assess the correctness of our implementation of a two phases Coxian server in Ψ^2. The validation method is defined as follows: We compute an empirical estimation of the stationary distribution from a sample of size n, obtained with Ψ^2. Then, the resulting estimation is compared with the exact stationary distribution. We use PRISM (numerical method) to compute the exact distribution, so that we also assess that the same models are considered in both PRISM and Ψ^2. We use a Chi-square test to measure the fitting of a Ψ^2 sample with the theoretical expected distribution. Details on the Chi-square test can be found in [7]. Note that grouping into classes (the distance between the estimated and theoretical distributions is measured on groups of states) is needed for some models, in particular when there are a lots of states with a probability of almost 0.

Results of the Chi-square tests are reported in Tables 1 and 2 for various values of the capacity of queues, the arrival rate λ and the service rate κ.

In every cases, we have $D < \chi^2_{(0.95,k-1)}$, so that the Chi-square test tell us that samples obtained with Ψ^2 are statistically correct with a confidence level of 95%. We also applied this validation method to assess the correctness of our implementation of *Tandem network* and of *Multistage delta network* models in PRISM.

5.2. Experimental setup

Tools and hardware settings The 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 ϵ : the desired convergence error or precision; $maxiternumber$: the maximal number of iterations to obtain the result with certain precision. Ψ^2 tool has four parameters $Nsamp$: the desired number of samples; α : the desired probability of false-positive answer for H.T. ; β : the desired probability of the false-negative answer for H.T. ; δ : the half width of the indifference region. In fact, to match these tools parameters and in order to have a fair comparison we take $\epsilon = 2.\delta$ [18].

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 check 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. Generally, the obtained precision on the collected data will be greater or equal to 10^{-2}.

Memory consumption In the case of the numerical solution method, 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, then the memory is proportional to the number of states. On the other hand, the memory requirements for the statistical approaches 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. Then memory is never exhausted during verification when using the statistical solution method.

5.3. Experimental results

Tandem network verification results

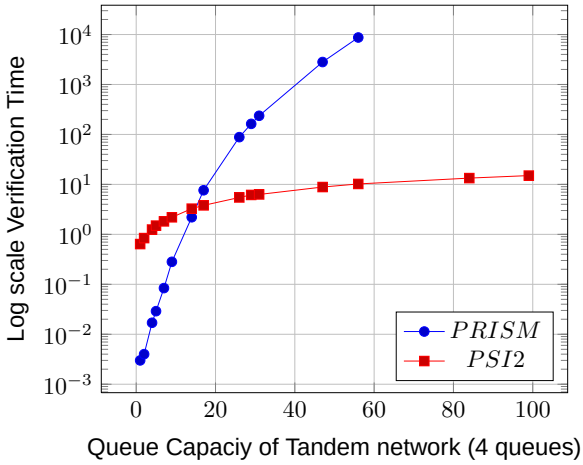

Figure 7: *Verification time as function of queue capacity* N_{max} *for* $S_{<0.001}$ *(last-full) for* x *=4,* ϵ = $10^{-3}/2$ *, and* δ = $10^{-3}/4$

For numerical application, we consider λ =0.9, all service rates will be state-independent with rate $\mu_i = 1,\ 1 \leq i \leq b$.

We give in Table 3 for $b = 4$, for $\theta = 0.001$ and for $\epsilon = 10^{-3}/2$, $\epsilon = 2.10^{-4}$, and $\epsilon = 10^{-4}$, 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 $b = 4$, for $\theta = 0.001$, $\delta = \{10^{-3}/4, 10^{-4}, 10^{-4}/2\}$ respectively, and $\alpha = \beta = 10^{-2}$ the statistical verification time for the same steady-state formula $S_{<\theta}$ (*last-full*) by using our verification method implemented in Ψ^2 (section 3.3). Next we represent part of these results ($N_{max} \leq 99$) in Figures 7, 8, and 9. In fact, for $N_{max} = 99$ we obtain an out of memory message with PRISM.

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.
$PSI2$: statistical model checking time in seconds for the steady-state formula by using our verification method (section 3.3) implemented in Ψ^2 engine.
$outmem$: an out of memory message in PRISM tool.
$iterpr$: a maximal iteration number problem (we consider $maxiternumber = 100000$ in PRISM tool).

Multistage delta network (MDN) verification results

For numerical application, we consider $y = 2\ stages$ and $z = 4\ buffers/stage$, $\lambda = 0.9$, $\mu = 1$, $(\tau_{rout1}, \tau_{rout2}) = (0.8, 0.6)$. Note that, for $y = 4\ stages$ and $z = 8\ buffers/stage$ we have obtain efficient results by using Ψ^2 [11, 10] while it is not possible to do numerical model checking PRISM (memory problem

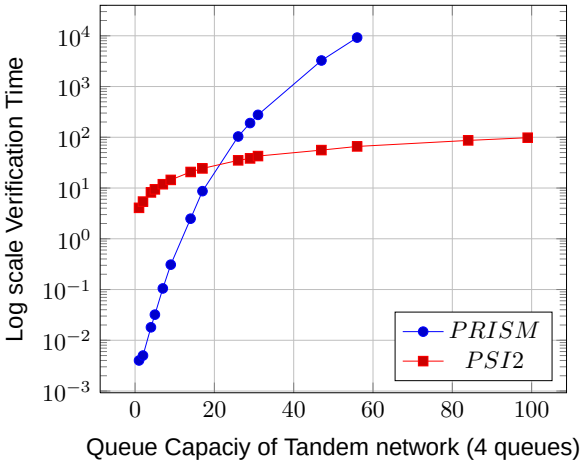

Figure 8: *Verification time as function of queue capacity* N_{max} *for* $S_{<0.001}$ *(last-full) for* $x = 4$ *,* $\epsilon = 2.10^{-4}$ *, and* $\delta = 10^{-4}$

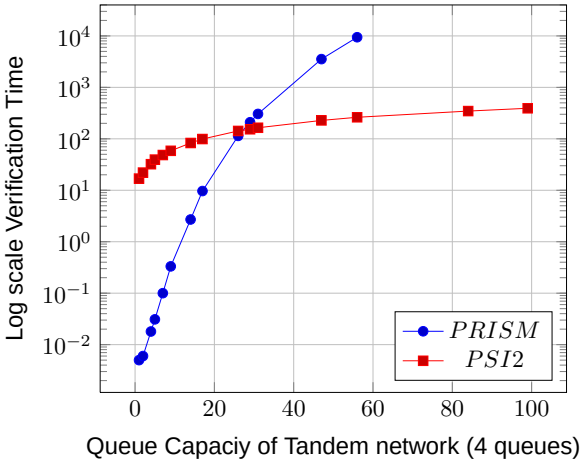

Figure 9: *Verification time as function of queue capacity* N_{max} *for* $S_{<0.001}$ *(last-full) for* $x = 4$*,* $\epsilon = 10^{-4}$*, and* $\delta = 10^{-4}/2$

$(\epsilon,\ \delta)$	N_{max}	$\lvert\mathcal{X}\rvert$	VERIF. TIME in sec $PRISM$	$PSI2$
$(10^{-3}/2, 10^{-3}/4)$	1	$1.60 * 10^1$	0.003	0.6
	2	$8.10 * 10^1$	0.004	0.8
	4	$6.25 * 10^2$	0.02	1.2
	5	$1.29 * 10^3$	0.03	1.5
	7	$4.09 * 10^3$	0.08	1.8
	9	$1.00 * 10^4$	0.28	2.2
	14	$5.06 * 10^4$	2.21	3.2
	17	$1.04 * 10^5$	7.6	3.8
	26	$5.30 * 10^5$	88.3	5.5
	29	$8.10 * 10^5$	162.8	6.1
	31	$1.04 * 10^6$	236.4	6.3
	47	$5.30 * 10^6$	2814	8.8
	56	$1.05 * 10^7$	8716	10.2
	84	$5.22 * 10^7$	$iterpr$	13.4
	99	$1.00 * 10^8$	$outmem$	15.0
	999	$1.00 * 10^{12}$	$outmem$	56.4
	9999	$1.00 * 10^{16}$	$outmem$	219.6
	N_{max}	$\lvert\mathcal{X}\rvert$		
$(2.10^{-4}, 10^{-4})$	1	$1.60 * 10^1$	0.004	4.1
	2	$8.10 * 10^1$	0.005	5.4
	4	$6.25 * 10^2$	0.02	8.2
	5	$1.29 * 10^3$	0.03	9.4
	7	$4.09 * 10^3$	0.10	11.9
	9	$1.00 * 10^4$	0.32	14.4
	14	$5.06 * 10^4$	2.49	20.7
	17	$1.04 * 10^5$	8.67	24.3
	26	$5.30 * 10^5$	103.5	35.1
	29	$8.10 * 10^5$	190.6	38.2
	31	$1.04 * 10^6$	276.6	42.6
	47	$5.30 * 10^6$	3258	55.9
	56	$1.05 * 10^7$	9213	65.9
	84	$5.22 * 10^7$	$iterpr$	86.6
	99	$1.00 * 10^8$	$outmem$	98.1
	999	$1.00 * 10^{12}$	$outmem$	365.3
	9999	$1.00 * 10^{16}$	$outmem$	1314
	N_{max}	$\lvert\mathcal{X}\rvert$		
$(10^{-4}, 10^{-4}/2)$	1	$1.60 * 10^1$	0.005	16.8
	2	$8.10 * 10^1$	0.006	22.1
	4	$6.25 * 10^2$	0.02	32.1
	5	$1.29 * 10^3$	0.03	39.5
	7	$4.09 * 10^3$	0.10	48.4
	9	$1.00 * 10^4$	0.33	58.5
	14	$5.06 * 10^4$	2.71	82.9
	17	$1.04 * 10^5$	9.66	99.0
	26	$5.30 * 10^5$	114.0	142.3
	29	$8.10 * 10^5$	209.9	153.4
	31	$1.04 * 10^6$	304.7	164.1
	47	$5.30 * 10^6$	3542	227.9
	56	$1.05 * 10^7$	9424	262.2
	84	$5.22 * 10^7$	$iterpr$	346.2
	99	$1.00 * 10^8$	$outmem$	392.4
	999	$1.00 * 10^{12}$	$outmem$	1464
	9999	$1.00 * 10^{16}$	$outmem$	5303

Table 3: *Tandem network: Verification time as function of state space size* $\lvert\mathcal{X}\rvert$ *and of queue capacity* N_{max} *for* $S_{<0.001}$ *(last-full)*

for N_{max}=1) in this case due to the huge state space size $O((N_{max}+1)^{32})$.

We give in Table 4 for $\theta = 0.001$ and for $\epsilon = 10^{-3}/2$, $\epsilon = 2.10^{-4}$, and $\epsilon = 10^{-4}$, 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 $\theta = 0.001$, $\delta = \{10^{-3}/4,\ 10^{-4},\ 10^{-4}/2\}$ respectively, and $\alpha = \beta = 10^{-2}$, the statistical verification time for the same steady-state formula $S_{<\theta}$ (*last-stage-full*) by using our verification method. We represent part of these results ($N_{max} \leq 10$) in

$(\epsilon,\ \delta)$	N_{max}	$\lvert\mathcal{X}\rvert$	VERIF. TIME in sec $PRISM$	$PSI2$
$(10^{-3}/2, 10^{-3}/4)$	2	$6.5 * 10^3$	1.8	2.2
	3	$6.5 * 10^4$	2.1	2.9
	4	$3.9 * 10^5$	12.1	3.5
	5	$1.6 * 10^6$	74.9	3.9
	6	$5.7 * 10^6$	426	4.3
	7	$1.6 * 10^7$	1689	4.6
	8	$4.3 * 10^7$	4768	4.8
	9	$1.0 * 10^8$	8931	5.0
	10	$2.1 * 10^8$	$outmem$	5.2
	99	$1.0 * 10^{16}$	$outmem$	6.8
	999	$1.0 * 10^{24}$	$outmem$	6.9
	9999	$1.0 * 10^{32}$	$outmem$	7.1
	N_{max}	$\lvert\mathcal{X}\rvert$		
$(2.10^{-4}, 10^{-4})$	2	$6.5 * 10^3$	1.8	13.6
	3	$6.5 * 10^4$	2.2	18.3
	4	$3.9 * 10^5$	13.2	22.1
	5	$1.6 * 10^6$	82.6	24.9
	6	$5.7 * 10^6$	431	27.0
	7	$1.6 * 10^7$	1782	28.9
	8	$4.3 * 10^7$	5349	30.5
	9	$1.0 * 10^8$	9906	31.7
	10	$2.1 * 10^8$	$outmem$	32.8
	99	$1.0 * 10^{16}$	$outmem$	43.3
	999	$1.0 * 10^{24}$	$outmem$	43.4
	9999	$1.0 * 10^{32}$	$outmem$	43.7
	N_{max}	$\lvert\mathcal{X}\rvert$		
$(10^{-4}, 10^{-4}/2)$	2	$6.5 * 10^3$	1.8	54.5
	3	$6.5 * 10^4$	2.3	73.1
	4	$3.9 * 10^5$	14.1	88.2
	5	$1.6 * 10^6$	88.5	99.3
	6	$5.7 * 10^6$	434	110
	7	$1.6 * 10^7$	1895	117
	8	$4.3 * 10^7$	6125	123
	9	$1.0 * 10^8$	10135	127
	10	$2.1 * 10^8$	$outmem$	131
	99	$1.0 * 10^{16}$	$outmem$	174
	999	$1.0 * 10^{24}$	$outmem$	175
	9999	$1.0 * 10^{32}$	$outmem$	177

Table 4: *MDN: Verification time as function of state space size $|\mathcal{X}|$ and of queue capacity N_{max} for $S_{<0.001}$ (last-stage-full)*

Figures 10, 11, and 12. In fact, for $N_{max} = 10$ we obtain an out of memory message with PRISM.

Tandem network with coxian phase(TQN) verification results

For numerical application, we consider $\lambda = 4 \times N_{max}$, $\mu_1 = 2$, $\mu_2 = 2$, $a = 0.1$ and κ =1.

We give in Table 5 for $\theta = 0.001$ and for $\epsilon = 10^{-3}/2$, $\epsilon = 2.10^{-4}$, and $\epsilon = 10^{-4}$, the numerical verification time for the considered steady-state formula $S_{<\theta}$ (*sys-full*) by using PRISM Hybrid engine and Jacobi iterative method. Also we give in the same table for $\theta = 0.001$, $\delta = \{10^{-3}/4,\ 10^{-4},\ 10^{-4}/2\}$ respectively, and $\alpha = \beta = 10^{-2}$, the statistical verification time for the same steady-state formula $S_{<\theta}$ (*sys-full*) by using our verification method. Next we represent part of these results ($N_{max} \leq 1023$) in Figures 13, 14, and 15. In fact, for $N_{max} = 7500$ we obtain an out of memory message with PRISM.

Discussions

Our results confirm that for steady-state property, our statistical method scales better with the size

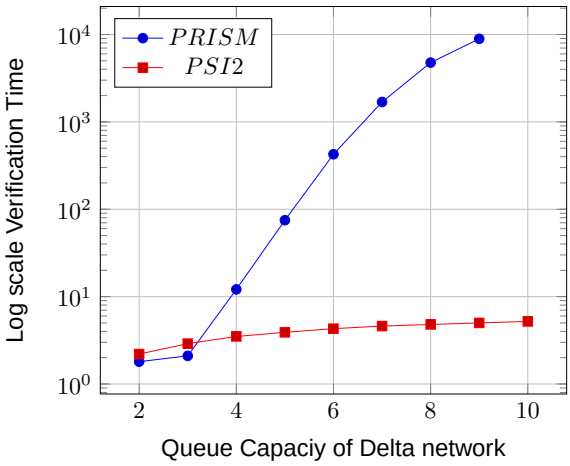

Figure 10: *MDN: Verification time as function of queue capacity N_{max} for $S_{<0.001}$ (last-stage-full) for ϵ = $10^{-3}/2$, and δ = $10^{-3}/4$*

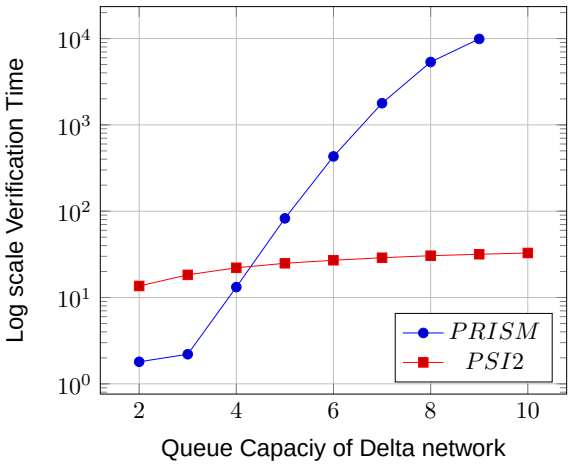

Figure 11: *MDN: Verification time as function of queue capacity N_{max} for $S_{<0.001}$ (last-stage-full) for ϵ = 2.10^{-4} , and δ = 10^{-4}*

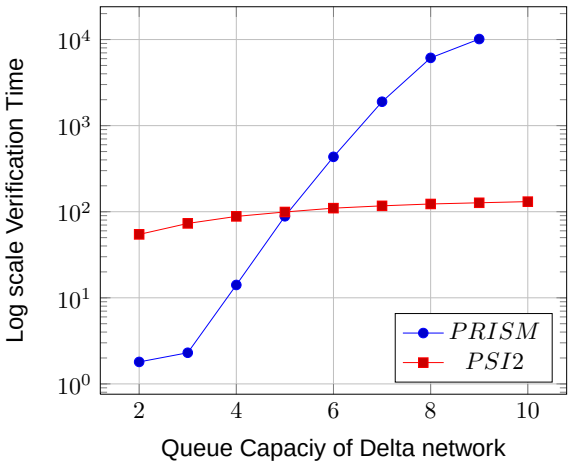

Figure 12: *MDN: Verification time as function of queue capacity N_{max} for $S_{<0.001}$ (last-stage-full) for ϵ = 10^{-4} , and δ = $10^{-4}/2$*

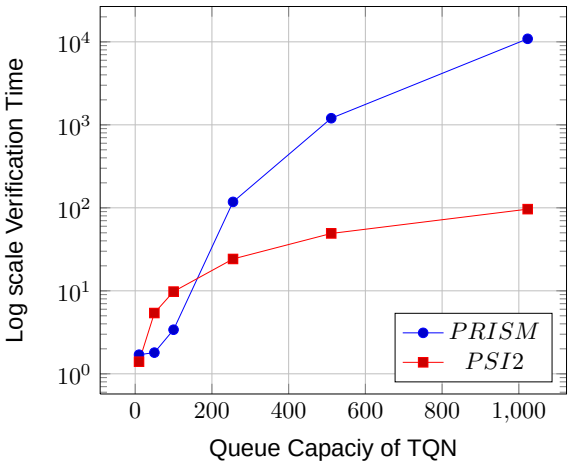

Figure 13: *TQN: Verification time as function of queue capacity* N_{max} *for* $S_{<0.001}$ *(sys-full) for* $\epsilon = 10^{-3}/2$ *, and* $\delta = 10^{-3}/4$

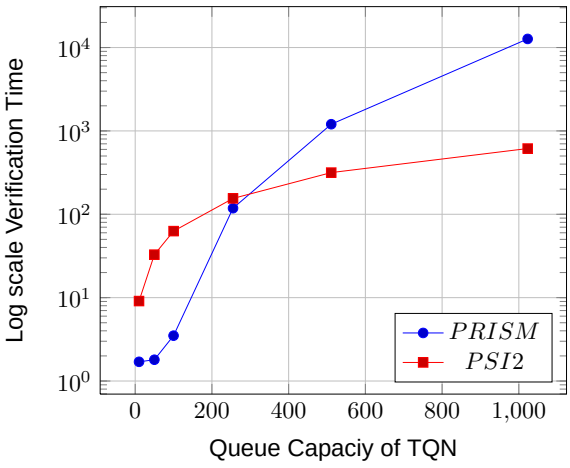

Figure 14: *TQN: Verification time as function of queue capacity* N_{max} *for* $S_{<0.001}$ *(sys-full) for* $\epsilon = 2.10^{-4}$ *, and* $\delta = 10^{-4}$

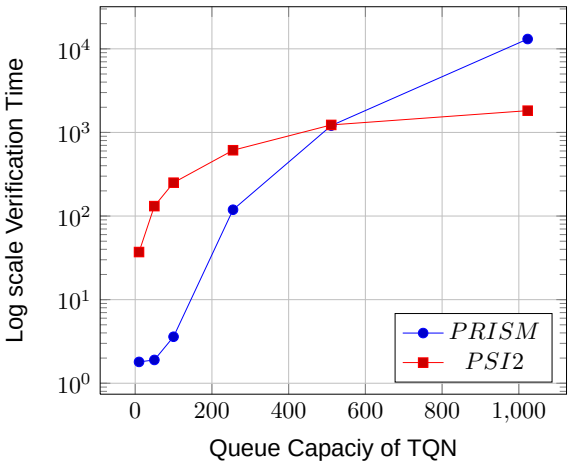

Figure 15: *TQN: Verification time as function of queue capacity* N_{max} *for* $S_{<0.001}$ *(sys-full) for* $\epsilon = 10^{-4}$ *, and* $\delta = 10^{-4}/2$

$(\epsilon,\ \delta)$	N_{max}	$\lvert\mathcal{X}\rvert$	VERIF. TIME in sec $PRISM$	$PSI2$
$(10^{-3}/2, 10^{-3}/4)$	10	$2.3 * 10^2$	1.7	1.4
	50	$5.1 * 10^3$	1.8	5.4
	100	$2.0 * 10^4$	3.4	9.8
	255	$1.3 * 10^5$	117.9	24.2
	511	$5.2 * 10^5$	1201	49.1
	1023	$2.1 * 10^6$	10879	96.4
	5000	$5.1 * 10^7$	$iterpr$	493.2
	7500	$1.1 * 10^8$	$outmem$	819.6
	10000	$2.1 * 10^8$	$outmem$	925.3
	N_{max}	$\lvert\mathcal{X}\rvert$		
$(2.10^{-4}, 10^{-4})$	10	$2.3 * 10^2$	1.7	9.1
	50	$5.1 * 10^3$	1.8	32.8
	100	$2.0 * 10^4$	3.5	62.8
	255	$1.3 * 10^5$	118.3	154.9
	511	$5.2 * 10^5$	1203	316.3
	1023	$2.1 * 10^6$	12689	614.7
	5000	$5.1 * 10^7$	$iterpr$	3058
	7500	$1.1 * 10^8$	$outmem$	4615
	10000	$2.1 * 10^8$	$outmem$	5793
	N_{max}	$\lvert\mathcal{X}\rvert$		
$(10^{-4}, 10^{-4}/2)$	10	$2.3 * 10^2$	1.8	37.1
	50	$5.1 * 10^3$	1.9	131.4
	100	$2.0 * 10^4$	3.6	250.7
	255	$1.3 * 10^5$	119.0	612.4
	511	$5.2 * 10^5$	1208	1230
	1023	$2.1 * 10^6$	13105	1822
	5000	$5.1 * 10^7$	$iterpr$	12258
	7500	$1.1 * 10^8$	$outmem$	19153
	10000	$2.1 * 10^8$	$outmem$	23219

Table 5: *TQN: Verification time as function of state space size* $\lvert\mathcal{X}\rvert$ *and of queue capacity* N_{max} *for* $S_{<0.001}$ *(sys-full)*

of the state space. Moreover it is generally faster. However, high accuracy comes at a greater price than for numerical method. Tables 3, 4, and 5 and their corresponding figures show the verification time in seconds as function of state space size in the three case studies. In fact, 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 Ψ^2 is the fastest. Moreover, Tables 3, 4 and 5 show the memory limits of the PRISM tool for large state space size while the Ψ^2 tool scales better, is efficient and do not have any memory limitation for these large state space size. For *Tandem network* case, we obtain an out of memory message in PRISM tool from the value $\lvert\mathcal{X}\rvert = 1.0 * 10^8$ (corresponding to $N_{max} = 99$). For *Multistage delta network* case, we obtain an out of memory message in PRISM tool from the value $\lvert\mathcal{X}\rvert = 2.1 * 10^8$ (corresponding to $N_{max} = 99$). Moreover, for *Tandem network with coaxian phase* case, we obtain an out of memory message in PRISM tool from the value $\lvert\mathcal{X}\rvert = 1.1 * 10^8$ (corresponding to $N_{max} = 7500$). On the other hand, we have variated the precision parameters of numerical (ϵ) and of statistical (δ) methods. Thus we have note from Tables 3, 4 and 5 that the numerical verification time dependence on ϵ is negligible while the statistical verification time dependence on δ is considerable. Moreover, we refer to section 3.1 to explain why in some of our

case studies, the obtained statistical verification time does not depend on $|\mathcal{X}|$. Finally note that, we have used single acceptance sampling method in our statistical method implemented in Ψ^2. However, even if we have not use sequential acceptance sampling which is more efficient than single one, we have obtain more efficient and more scalable results than numerical method.

6. CONCLUSION AND FUTURE WORKS

In this paper, we have empirically compared numerical and statistical solution techniques for probabilistic model checking on case studies taken from the PRISM and Ψ^2 benchmarks. We focused our attention on steady-state properties. For these properties, we have found that our statistical method implemented in Ψ^2 scales better with the state space size and it is faster than PRISM tool especially for large models. In fact, we aim to find the limiting problem sizes for the considered case studies. We see that the statistical approach scales well with the problem size and it lets us to consider very large models. We consider to compare our statistical method (single and sequential acceptance sampling) with the one implemented in MRMC tool.

Acknowledgments This work is supported by a French research project, ANR-06-SETI-002

REFERENCES

[1] J. M. Vincent A. Busic and B. Gaujal. Perfect simulation and non-monotone (markovian) systems. In *VALUETOOLS08*. ACM, 2008.

[2] C. Baier, B. Haverkort, H. Hermanns, and J.P. Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.

[3] C. Baier and J. P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[4] G. Gorgo. Envelope perfect sampling of 2-phases coxian service in queueing networks. inria. 2010.

[5] G. Norman H. L. S. Younes, M. Z. Kwiatkowska and D. Parker. Numerical vs. statistical probabilistic model checking. *STTT*, 8(3):216–228, 2006.

[6] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. In *TACAS06*, volume 3922 of *LNCS*, pages 441–444, 2006.

[7] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[8] J. P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker mrmc. In *QEST09*, pages 167–176. IEEE Computer Society, 2009.

[9] D. Propp and J. Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9(1 and 2):223–252, 1996.

[10] D. El Rabih and N. Pekergin. Statistical model checking for steady state dependability verification. In *DEPEND09*, pages 166–169. IEEE Computer Society, 2009.

[11] D. El Rabih and N. Pekergin. Statistical model checking using perfect simulation. In *ATVA09*, volume 5799 of *LNCS*, pages 120–134, 2009.

[12] K. Sen, M. Viswanathan, and G. Agha. Vesta: A statistical model-checker and analyzer for probabilistic systems. In *QEST05*, pages 251–252. IEEE Computer Society, 2005.

[13] R. Lassaigne T. Hérault and S. Peyronnet. Apmc 3.0: Approximate verification of discrete and continuous time markov chains. In *QEST06*, pages 129–130. IEEE Computer Society, 2006.

[14] J. M. Vincent. Perfect simulation of monotone systems for rare event probability estimation. In *Winter Simulation Conference*, pages 528–537. ACM, 2005.

[15] J. M. Vincent and J. Vienne. ψ^2 a software tool for the perfect simulation of finite queueing networks. In *QEST07*, pages 113–114. IEEE Computer Society, 2007.

[16] J.M. Vincent and C. Marchand. On the exact simulation of functionals of stationary markov chains. *Linear Algebra and its Applications*, 386:285–310, 2004.

[17] H. L. S. Younes. Ymer: A statistical model checker. In *CAV05*, volume 3576 of *LNCS*, pages 429–433, 2005.

[18] H. L. S. Younes. Error control for probabilistic model checking. In *VMCAI06*, volume 3855 of *LNCS*, pages 142–156, 2006.

[19] H. L. S. Younes and R. G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.*, 204(9):1368–1409, 2006.