Performance Evaluation and Model Checking Join Forces

Christel Baier
TU Dresden
D-01062 Dresden
Germany
baier@tcs.inf.tu-dresden.de

Boudewijn R. Haverkort
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
brh@cs.utwente.nl

Holger Hermanns
Saarland University
D-66123 Saarbrücken
Germany
hermanns@cs.uni-sb.de

Joost-Pieter Katoen
RWTH Aachen University
D-52056 Aachen
Germany
katoen@cs.rwth-aachen.de

ABSTRACT

Over more than a century now, a wide variety of system performance evaluation techniques have been developed, based on the theory of stochastic processes. These have shown their practical relevance in guiding system design, and have found their way into software tools. But what is their status nowadays? In industrial practice we see simulation techniques prevailing. Is there any future for classical (analytical) performance evaluation?

In this paper, we will plea for a marriage between classical performance (and dependability) evaluation and state-of-the-art verification techniques. The last decade has shown a number of important joint efforts in this area, resulting in a quantitative system evaluation technology nowadays known as stochastic model checking. Not only does this marriage provide two evaluation types in one —correctness and performance— which is of utmost importance in many newer application areas, it also yields mutually beneficial insights that improve methods and techniques at either end. For these new techniques, we observe a growing application potential, well beyond computer science.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: modeling techniques, performance attributes, reliability, availability, serviceability; D.2.4 [**Software/Program Verification**]: formal methods, model checking, reliability; F.4.1 [**Mathematical Logic**]: temporal logic

General Terms

performance, reliability, verification

1. INTRODUCTION

Consider a major news website like BBC or CNN. Typically, such a site is equipped with a number of machines serving as front-ends to receive incoming requests together with some application servers such as database engines to handle these requests. When a new request arrives, to which server does the dispatcher have to route it? To the machine with the shortest queue, i.e., the queue with the minimal number of outstanding requests? This might be the best decision most times, but not in cases where some of the requests in the shortest queue happen to require a very long service time, e.g., because they involve very detailed queries. And what to do when the servers differ in computational capabilities? And what to do when multiple hosts have the same queue length? Well, the "join-the-shortest queue" policy might be adequate in most cases, but surely not in all. Its adequacy also depends on what quantity —or *measure*— one is interested in. This may be the mean delay of service requests, the mean queue length of waiting requests, rejection rates for waiting requests, and so on.

The effect of queue-selection policies on measures of interest or on decisions on how many servers are needed to reduce the waiting time by a given percentage, are answered by *performance evaluation* techniques. This branch of computer (system) science studies the perceived performance of systems based on an architectural system description and a workload model. Prominent techniques to obtain the aforementioned measures of interest are mathematical analysis that is typically focused on obtaining closed-form expressions, numerical evaluation that heavily relies on methods from linear algebra, and (discrete-event) simulation techniques which are based on statistical methods. The study and description of stochastic processes, most notably *Markov chains*, is pivotal for these techniques.

A complementary issue to performance is correctness. The central question is whether a system is conforming to the requirements and does not contain any flaws. Typically updates to our news website are queued, and it is relevant to know whether such buffers may overflow, giving rise to losing —perhaps headline— news items. Can such situations ever occur? Is there a possible scenario in which the dispatcher and application server are mutually waiting for each other, thus effectively halting the system? If such situations make the CNN news website unreachable at U.S. president election day, this has far reaching consequences. And may the content of webpages unexpectedly depend on the ordering of

seemingly unrelated events in the application servers? Such "race conditions" should, if possible, be avoided.

A prominent discipline in computer science to assure the absence of errors, or, complimentarily, to find errors ("bug hunting") is *formal verification*. The spectrum of key techniques in this field ranges from run-time verification, i.e., checking properties while executing the system, over deductive techniques such as theorem proving, to model checking. The latter is a highly automated model-based technique assessing whether a system model, i.e., the possible system behavior, satisfies a property describing the desirable behavior. Typically, properties are expressed in temporal extensions of propositional logic, and system behavior is captured by Kripke structures, i.e., finite-state automata with labeled states. Traditionally, such models do not incorporate quantitative information like timing or likelihoods of event occurrences.

The purpose of this paper is to report on combining performance evaluation with model checking. Although these fields have been developed by different research communities in the past, over the last decade we have seen an integration of these two techniques for system analysis. Significant merits of this trend are: (i) a major increase of the applicability to real cases, and (ii) an impulse in the further development for both fields.

2. A HISTORIC ACCOUNT

To appreciate the benefits of combining performance evaluation and model checking, it is worthwhile to reflect on past and recent developments. We aim to shed light on the hidden assumptions associated with these developments. For more details on performance evaluation we refer to [22, 10], for details on model checking we refer to [11, 7].

Performance Evaluation

Single queues. Performance evaluation dates back to the early 1900s, when Erlang developed models to dimension the number of required lines in analogue telephone switches, based on the calculation of call loss probabilities. In fact, he used a queueing model, in which a potentially infinite supply of customers (callers) compete for a limited set of resources (the lines). The set of models and the theory that evolved from there, is nowadays known as queueing theory. It has found, through the last century, wide applicability especially in telecommunications. Characteristic for most of these models is the competition for a single scarce resource at a time, leading to models with a single queue. A large variety of modeling assumptions were made, e.g., regarding the number of available servers (lines), buffering facilities, scheduling strategies, job discrimination, and the timing involved. The timings were assumed to follow some continuous-time distribution, most often a negative exponential distribution, leading to (what we now call) Markovian models. These models were subsequently analysed, using calculus, to obtain such quantities as mean number of customers queued, mean delay, sometimes even the delay distribution, or the call blocking probability ("hearing a busy tone"). Many of these measures are available in closed form; at other times, numerical recipes were proposed, e.g., to derive such measures from explicit expressions in the Laplace domain. Important to note is that model construction, as well as solution, was (and still is) seen as a craft, only approachable by experts.

Networks of queues. In the late 1960s, computer networks, networked computer systems, and time-sharing computer systems came into play. These systems have as distinguishing feature that they serve a finite customer population; however, they comprise multiple resources. This led to developments in the area of *queueing networks*, in which customers travel through a network of queues, are served at each queue according to some scheduling discipline, and are routed to their next point of service, and so on, until returning to the party that originated the request. Efficient algorithms to evaluate networks of queues to obtain a set of "standard measures" such as mean delays, throughputs and mean queue lengths, were developed in the 1970s [40, 51]. A variety of software tools emerged supporting these algorithms that typically have a polynomial complexity in the number of queues and customers.

Stochastic Petri nets. In early 1980s, new computer architectures asked for more expressive modeling formalisms. In particular, parallel computers motivated modeling notions to spawn customers and to recombine smaller tasks into larger ones (fork/join queues), as well as the simultaneous use of multiple resources at the same time needed to be studied. Clearly, these concepts could not be expressed using queueing networks. This led, among others, to the proposal to extend Petri nets —originally developed to model concurrency— with a notion of time, leading to (generalised) stochastic Petri nets (SPNs) [2]. Here, the tokens can either play the role of customers or of resources. Two observations are important. First, due to the increase in expressivity, specialised algorithms such as available for queueing networks, can typically not be used anymore. Instead, the SPN models need to be mapped to an underlying stochastic process, a Markov chain, that is solved by numerical means. Hence, the state space of the model has to be generated explicitly, and the resulting Markov chain has to be solved numerically (linear equation solvers). The computational complexity of these state-based methods is polynomial in the number of states, but this often is, in turn, (often) a high-degree polynom in the SPN size. Secondly, as a result of the new solution trajectory, tool support became a central issue. Results achieved in this area also inspired new numerical algorithms for extended queueing network models. With hindsight, SPNs can be considered as the first "product" of the marriage between the field of performance evaluation and the field of formal modeling. In the 1990s, this trend continued and led to probabilistic variants of guarded command languages and of process algebras, the latter focusing on compositionality.

Nondeterminism. All of the above models are full stochastic models; that is, at no point in the model can some behavioural alternatives be left unspecified. For instance, the join-the-shortest-queue strategy referred to in Section 1 leaves it open how to handle the case of several equally short queues. This choice cannot be left open with the above methods; leaving such a choice is regarded as underspecification. What typically happens is that these cases are dealt with probabilistically, e.g., by assigning probabilities to the alternatives. That is, *nondeterminism* is seen as a problem that has to be removed before analysis can take place. This is important especially for modeling formalisms as SPNs; tools supporting the evaluation of these models will either detect and report such nondeterminism through

a "well-specified check" or will simply insert probabilities to resolve it. In this case, analysis is carried out under a hidden assumption, and there is no guarantee that an actual implementation will exhibit the assumed behaviour, nor that the performance derived on the basis of this assumption is achieved.

Trends. The last 20 years have seen a variety of developments in performance evaluation, mostly related to specific application fields, such as the works on effective bandwidth [42], network calculus [46], self-similar traffic models [47] and traffic (and mobility) models [43] (for communication network dimensioning purposes). A more general concept has been the development of fluid models to avoid the state space explosion problem (e.g., [45]) by addressing a large denumerable state space as a single continuous state variable. Furthermore, queueing network models have been extended with layering principles to allow for the modeling of software phenomena [52]. Finally, work on matrix-geometric methods [48] has led to efficient analysis methods for large classes of queueing models.

Model checking

Proof rules. The fundamental question "when and why does a software not work as expected?" has been the subject of intensive research since the early days of computer science. Software quality is typically based on peer review, i.e., manual code inspection, extensive simulation, and testing. These rather ad-hoc validation techniques have severe limitations and restrictions. Research in the field of *formal verification* has led to complementary methods that are aimed at establishing software correctness with a very high level of confidence. The origins of a sound mathematical approach towards program correctness —at a time where programs were described as flow diagrams— can be traced back to Turing in the late 1940s. Early attempts to assess the correctness of computer programs were based on mathematical proof rules that allow to reason in a purely syntax-based manner. In the 1960s, these techniques were developed for sequential programs, whereas about a decade later, this approach was generalized towards concurrent programs, in particular shared-variable programs.

Temporal logic. These syntax-based approaches are based on an interpretation of programs as input/output transformers and serve to prove partial correctness (i.e., soundness of output values for given inputs, provided the program terminates) and termination. Thanks to an important insight in the late 1970s by Pnueli, one recognized the need for concurrent programs to not only make assertions about the starting and final state of a program, but also about the states during the computation. This led to the introduction of temporal logic in the field of formal verification [50]. Proofs, however, were still conducted mainly by hand along the syntax of programs. Proofs for programs of realistic size, though, were rather lengthy and required a good dose of human ingenuity. In the field of communication protocols, the first techniques appeared towards automated checking of elementary properties [53].

Model checking. In the early 1980s, an alternative to using proof rules was proposed that *checks* systematically whether a (finite) *model* of a program satisfies a given property [11, 7]. The pioneers Clarke, Emerson, and Sifakis, received the ACM Turing Award 2007 for this breakthrough; it was the first step towards the fully automated verification of concurrent programs. How does model checking work? Given a model of the system (the possible behavior) and a specification of the property to be considered (the desirable behavior), model checking is a technique that systematically checks the validity of the property in the model. Models are typically nondeterministic finite-state automata, consisting of a finite set of states and a set of transitions that describe how the system evolves from one state into another. These automata are usually composed of concurrent entities and are often generated from a high-level description language such as Petri nets, process algebras, `Promela`, or Statecharts. Properties are specified in temporal logic such as CTL (Computation Tree Logic), an extension of propositional logic that allows one to express properties that refer to the relative order of events. Statements can either be made about states or about paths, i.e., sequences of states that model system evolution.

The backbone of the CTL model checking procedure is a recursive descent over the parse tree of the formula under consideration where temporal conditions (e.g., a reachability or an invariance condition) are checked using fixed point computations. The class of path properties expressible in CTL is restricted to local conditions on the current states and its direct successors, constrained reachability conditions—is a goal state reachable by not visiting certain states before?— and their duals.

More complex path properties such as repeated reachability or progress properties which, e.g., can state that whenever a request enters the news website, it is served eventually, can be specified in LTL (Linear Temporal Logic). The rough idea of model checking LTL specifications is to transform the formula at hand into an automaton (recognizing infinite words) and then to analyze the product of this automaton with the system model by means of graph algorithms.

The strength of model checking is not in providing a rigorous correctness proof, but rather the ability to generate diagnostic feedback in the form of counterexamples (i.e., error traces) in case a property is refuted. This information is highly relevant to find flaws in the model and in the real system.

Taming state space explosion. The time and space complexity of these algorithms is linear in the size of the finite-state automaton describing the system. The main problem is, however, that this size may grow exponentially in the number of program and control variables, and in the number of components in a multi-threaded or distributed system. Since the birth of model checking, effective methods have been developed to combat this state explosion problem. Prominent examples of such techniques are: symbolic data structures [39], partial-order reduction [49], casting model checking as SAT-problems [38], or abstraction techniques [11]. Due to these techniques, together with unremitting improvements of underlying algorithms and data structures and hardware technology improvements, model checking techniques that a decade ago only worked for simple examples, are nowadays applicable to more realistic designs. State-of-the-art model checkers can handle state spaces of about 10^9 states using off-the-shelf technology. Using clever algorithms and tailored data structures, much larger state spaces (up to 10^{120} states [41]) can be handled for specific problems and reachability properties.

Quantitative aspects. From the early 1990s on, various

extensions of model checking have been developed to treat aspects such as time and probabilities. Automata have been equipped with clock variables to measure the elapse of time (resulting in timed automata), and it has been shown that despite the infinite underlying state space of such automata, model checking of a timed extension of CTL is still decidable [37]. LTL has been interpreted over (discrete) probabilistic extensions of automata, focusing on the probability that an LTL formula holds, and probabilistic variants of CTL have been developed, as we will elaborate in more detail later on. For an overview, see [7, Ch. 10]. The combination of timing aspects and probabilities started about two decades ago and is highly relevant for this paper.

Various software tools have been developed that support model checking. Some well-known model checking tools are: `SPIN` for LTL, `NuSMV` for CTL (and LTL), `Uppaal` for timed CTL, and `PRISM` for probabilistic CTL.

3. LET'S JOIN FORCES

Developments in performance evaluation are tending towards more complex measures of interest, and focus on more complex system behavior. On the other hand, quantitative aspects such as timing and random phenomena are becoming more and more important in the field of model checking. Performance evaluation and model checking have thus grown in each other's direction, simply because from either end, it was felt that the methods in isolation did not answer the questions that were at stake. Let us discuss the reasons for this, and the benefits of combining these methods.

Individual shortcomings

Why is a performance (or a dependability) evaluation of a system in itself not good enough? And why is a formal verification of a system insufficient to validate its usefulness? These questions are best answered by taking a simple system design example, for instance a reliable data transmission protocol such as TCP. Such a protocol relies on a number of ingredients that, when suitably combined, result in the desired behavior: reliable, end-to-end in-order delivery of packets between communicating peers. These ingredients comprise timers, sequence numbers, retransmissions and error detecting codes.

A typical performance model will take into account the TCP timing and retransmission aspects, whereas the error correction will mostly be included as a random phenomenon. For the sake of simplicity, sequence numbers are neglected. This results in a model that can be analysed using either a closed-form formula or some numerical technique, that, *under the assumption that the model is functionally correct*, gives a certain mean performance, measured as throughput or mean packet delay. However, the obtained quantities do not say anything about the question of whether the packets do arrive correctly at all, hence, whether the protocol is correct. Conversely, a classical functional model of the above sketched protocol will most probably result in a correctness statement of the form "all packets will eventually arrive correctly". This, however, gives no information at all about perceived delays and throughputs. Needless to say, one cannot simply "add up" the results of both analyses, as they result from two different —and possibly quite unrelated— models.

The key challenge lies in developing an integrated model. Preferably, the user, i.e., the system architect or design engineer, just has to provide a single model (as engineering artefact) that forms the basis for both types of analysis. To improve the efficiency, additional property-dependent abstraction techniques can be applied to abstract away from all details of the model that are irrelevant for the property to be checked. For example, checking whether a purely functional property holds for a Markov model requires an analysis of the underlying graph structure, and one can ignore all stochastic information.

Benefits

Modeling and measure specification. An important advantage of using temporal logics (or automata) to specify properties of interest —in fact guarantees on measures of interest— is the possibility to decribe properties at the same abstraction level as the modeling of the stochastic process. Up to now, it has been tradition to specify measures of interest such as "what is the probability to fail within deadline d?" at state level, i.e., in terms of the states and their elementary properties (logically speaking, atomic propositions). Sometimes reward structures have been added at state level to quantify the use of resources such as queue occupancies and the like. This stands in sharp contrast with the description of the models themselves, which is mostly done using high-level modeling formalisms such as queueing networks, SPNs, stochastic automata networks, or stochastic process algebra. Temporal logics close this paradigm gap between high-level and state-based modeling as they allow to specify properties in terms of the high-level models, e.g., in terms of the token distribution among places in a Petri net. By the use of temporal logics, modeling and measure specification become treated at an equal footing.

An example logic with semantics interpretation is illustrated in Figure 1. Instances of this generic logic arise by considering special types of stochastic processes, e.g., for an interpretation over discrete-time Markov chains (DTMC), $\mathbb{T} = \mathbb{N}$ and we obtain probabilistic computation tree logic (PCTL) [18]. For continuous-time Markov chains (CTMC), the time domain is $\mathbb{T} = \mathbb{R}$, and continuous stochastic logic (CSL) is obtained [4, 6]. Figure 3 presents a small representative example [9, 3], together with some typical logical formulae.

Expressivity and flexibility. The use of logics offers, in addition, a high degree of expressiveness. Simple performance and dependability metrics such as transient probabilities — what is the probability of being in a failure state at time t?— and long-run likelihoods (when the system is observed long enough) can readily be expressed. Most standard performance measures are easy be captured, cf. Table 1 for a selection of properties. More importantly, the use of logics offers an enormous degree of flexibility. Nesting formulas yields a simple mechanism to specify complex measures in a succinct manner. A property like " the probability to reach a state within 25 seconds that almost surely stays safe for the next 10 seconds, via legal states only exceeds $\frac{1}{2}$" boils down to

$$\mathbb{P}_{>\frac{1}{2}}\left(\mathit{legal}\ \mathcal{U}^{\leqslant 25}\ \mathbb{P}_{=1}(\Box^{\leqslant 10}\ \mathit{safe})\right).$$

This immediately pinpoints another advantage: given the formal semantics of the temporal logic, the meaning of the above formula is precise. That is to say, there is no possibility that any confusion might arise about its meaning. Unambiguous measure specifications are of utmost importance.

Let $\mathbb{X}$ be a general stochastic process, i.e, an indexed family $\{X(t) \mid t \in \mathbb{T}\}$ of random variables taking values in the set S. The index set $\mathbb{T}$ denotes the time domain of $\mathbb{X}$ and is either discrete ($\mathbb{T} = \mathbb{N}$) or continuous ($\mathbb{T} = \mathbb{R}$). We suppose that all states have positive probability under the initial distribution μ_{init}, i.e., $\mu_{\text{init}}(s) = \mathbf{Pr}_{\mathbb{X}}(X(0) = s) > 0$ for all states s. For event E, let $\mathbf{Pr}_{\mathbb{X},s}(E)$ denote the probability for E under the condition that s is the start state. Each state is labelled by a set of atomic propositions which can be viewed as state predicates.

Logical formulas (denoted by capital greek letters Φ, Ψ) are given by the grammar:

$$\Phi \quad ::= \quad a \;\Big|\; \Phi \wedge \Psi \;\Big|\; \neg\Phi \;\Big|\; \mathbb{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}^{\mathbb{I}} \Psi) \;\Big|\; \mathbb{L}_{\trianglelefteq p}(\Phi)$$

Here, a is an atomic proposition, $p \in [0,1]$, $\trianglelefteq \in \{\leqslant, \geqslant, >, <\}$ and $\mathbb{I}$ is a closed interval of $\mathbb{T}$.

The semantics of this logic is defined inductively as follows:

$$\begin{aligned}
s \models a \quad &\text{iff} \quad \text{state } s \text{ is labelled with atomic proposition } a \\
s \models \Phi \wedge \Psi \quad &\text{iff} \quad s \models \Phi \text{ and } s \models \Psi \\
s \models \neg\Phi \quad &\text{iff} \quad s \not\models \Phi \\
s \models \mathbb{P}_{\trianglelefteq p}(\Phi\ \mathcal{U}^{\mathbb{I}}\ \Psi) \quad &\text{iff} \quad \mathbf{Pr}_{\mathbb{X},s}\Big\{\exists t \in \mathbb{I}\ (\ X(t) \models \Psi\ \wedge\ \forall t' \in \mathbb{T}\ (\ t' < t \Rightarrow X(t') \models \Phi\ )\ )\Big\} \trianglelefteq p \\
s \models \mathbb{L}_{\trianglelefteq p}(\Phi) \quad &\text{iff} \quad \mathrm{LRA}(s, Sat_{\mathbb{X}}(\Phi)) \trianglelefteq p
\end{aligned}$$

where $Sat_{\mathbb{X}}(\Phi) = \{s \in S \mid s \models \Phi\}$ and for $B \subseteq S$, $\mathrm{LRA}(s,B)$ denotes the "long run average" of being in a state of B for runs starting in state s. Formally, $\mathrm{LRA}(s,B)$ is the expected value of the random variable

$$\lim_{t\to\infty} \frac{1}{t} \int_0^t \mathbf{1}_B(X(\theta))\, d\theta$$

with respect to the probability measure $\mathbf{Pr}_{\mathbb{X},s}$. Here, $\mathbf{1}_B$ denotes the characteristic function of B, i.e., $\mathbf{1}_B(s') = 1$ if $s \in B$ and 0 otherwise.

Derived operators. Let $\mathcal{U}^{\mathbb{T}}$ denote $\mathcal{U}$. Usual propositional operators such as ff, tt, $\vee$ are derivable. The eventually operator $\Diamond^{\mathbb{I}}$ with time bounds given by a time interval $\mathbb{I}$ is obtained by $\Diamond^{\mathbb{I}}\Phi = \mathrm{tt}\mathcal{U}^{\mathbb{I}}\Phi$. To specify that condition Φ holds continuously in the time interval $\mathbb{I}$, the time-constrained always operator $\Box^{\mathbb{I}}$ can be defined by using the duality of "eventually" and "always". For instance, $\mathbb{P}_{\geqslant p}(\Box^{\mathbb{I}}\Phi)$ is a shorthand notation for $\mathbb{P}_{\leqslant 1-p}(\Diamond^{\mathbb{I}}\neg\Phi)$.

Figure 1: A logic for quantitative properties: syntax and semantics

long-run	$\mathbb{L}_{\trianglelefteq p}(up)$
instantaneous	$\mathbb{P}_{\trianglelefteq p}(\Diamond^{[t,t]} up)$
conditional instantaneous	$\mathbb{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}^{[t,t]} up)$
interval	$\mathbb{P}_{\trianglelefteq p}(\Box^{[t,t']} up)$
long-run interval	$\mathbb{L}_{\trianglelefteq p}(\mathbb{P}_{\trianglelefteq q}(\Box^{[t,t']} up))$
conditional interval long-run	$\mathbb{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}^{[t,t']}\, \mathbb{L}_{\trianglelefteq q}(up))$

Table 1: Availability measures and their logical specification

Existing mathematical measure specifications are rigorous too of course, but do not offer the flexibility and succinctness of logics. Temporal logic provides a framework that is based on just a few basic operators.

Many measures, one algorithm. The above concerns the measure specification. The main benefit though is the use of model checking as a fully algorithmic approach towards measure evaluation. Even better, it provides a *single* computational technique for *any* possible measure that can be written. This applies from simple properties to complicated, nested, and possibly hard-to-grasp formulas. For the example logic this is illustrated in Figure 2. This is radically different from common practice in performance and dependability evaluation where tailored and brand new algorithms are developed for "new" measures. One might argue that this will have a high price, i.e., that the computational and space complexity of the exploited algorithms must be extremely high. No! On the contrary, in the worst case, the time complexity is linear in the size of the measure specification (logic fomula), and polynomial (typically of order 2 or 3, at most) in the number of states of the stochastic process under consideration. As indicated in Figure 4, the verification of bounded reachability probabilities in DTMCs and CTMCs —often the most time-consuming ones— is a matter of a few seconds even for millions of states: The space complexity is quadratic in the number of states in the worst case. In fact, as for other state-based performance evaluation techniques this polynomial complexity is an issue of concern as the number of states may grow rapidly.

Perhaps the largest advantage of model checking for performance analysis is that all algorithmic details, all detailed and non-trivial numerical computation steps are hidden to the user. Without any expert knowledge on, say, numerical analysis techniques for CTMCs, measure evaluation is possible. Even better: the algorithmic analysis is *measure-driven*. That is to say, the stochastic process can be tailored to the

given: a stochastic process $\mathbb{X}$ and a logical formula Φ
task: compute $\mathbf{Pr}_{\mathbb{X}}\{X(0) \models \Phi\}$
idea: compute the sets $Sat_{\mathbb{X}}(\Psi) = \{s \in S \mid s \models \Psi\}$ for any subformula Ψ of Φ and return $\sum_{s \in Sat_{\mathbb{X}}(\Phi)} \mu_{\text{init}}(s)$

- $Sat_{\mathbb{X}}(a) = \{s \in S \mid$ state s is labelled with atomic proposition $a\}$
- $Sat_{\mathbb{X}}(\Psi_1 \wedge \Psi_2) \;=\; Sat_{\mathbb{X}}(\Psi_1) \cap Sat_{\mathbb{X}}(\Psi_2)$
- $Sat_{\mathbb{X}}(\neg\Psi) \;=\; S \setminus Sat_{\mathbb{X}}(\Psi)$
- computation of $Sat_{\mathbb{X}}(\mathbb{P}_{\trianglelefteq p}(\Psi_1 \; \mathcal{U}^{\mathbb{I}} \; \Psi_2))$:

  case 1: $\mathbb{I} = [0,t]$ for some $t \in \mathbb{T}$, $t > 0$. Let $\mathbb{Y}$ be the stochastic process that results from $\mathbb{X}$ by making all states where Ψ_2 holds or Ψ_1 is refuted absorbing. That is, if $B = Sat_{\mathbb{X}}(\Psi_2) \cup S \setminus Sat_{\mathbb{X}}(\Psi_1)$, then $\mathbb{Y}$ is given by

  $$Y(t) = \begin{cases} X(t) & : \text{ if } X(t') \notin B \text{ for all } t' < t \\ s & : \text{ if } X(t') = s \in B \text{ for some } t' < t \text{ and } X(t'') \notin B \text{ or } X(t'') = s \text{ for all } t'' < t'. \end{cases}$$

  Apply known methods of performance evaluation to compute the probabilities

  $$p_s \;=\; \mathbf{Pr}_{\mathbb{Y},s}\Big\{X(t) \in Sat_{\mathbb{X}}(\Psi_2)\Big\}$$

  and return $Sat_{\mathbb{X}}(\mathbb{P}_{\trianglelefteq p}(\Psi_1 \; \mathcal{U}^{\mathbb{I}} \; \Psi_2)) = \{s \in S \mid p_s \trianglelefteq p\}$.

  case 2: $\mathbb{I} = [t_1,t_2]$ for some $t_1 > 0$. Let $\mathbb{Y}$ be the stochastic process that arises from $\mathbb{X}$ by making all states refuting Ψ_1 absorbing. Regard the stochastic process $\mathbb{Z}$ that arises from $\mathbb{Y}$ by shifting the time by t_1 time units, i.e., $\mathbb{Z}$ is specified by $Z(t) = Y(t+t_1)$. We then evaluate the formula $\mathbb{P}_{\trianglelefteq p}(\Psi_1 \; \mathcal{U}^{[0,t_2-t_1]} \; \Psi_2)$ over $\mathbb{Z}$ as in case 1 and return

  $$Sat_{\mathbb{X}}(\mathbb{P}_{\trianglelefteq p}(\Psi_1 \; \mathcal{U}^{\mathbb{I}} \; \Psi_2)) \;=\; Sat_{\mathbb{Z}}(\mathbb{P}_{\trianglelefteq p}(\Psi_1 \; \mathcal{U}^{[0,t_2-t_1]} \; \Psi_2)).$$

- Let $B = Sat_{\mathbb{X}}(\Phi)$ and apply known methods of performance evaluation to compute the long run average $\text{LRA}(s,B)$ of being in a state of B for runs starting in state s. Return

  $$Sat_{\mathbb{X}}(\mathbb{L}_{\trianglelefteq p}(\Phi)) \;=\; \{s \in S \mid \text{LRA}(s,B) \trianglelefteq p\}.$$

Figure 2: Schema for model checking stochastic processes

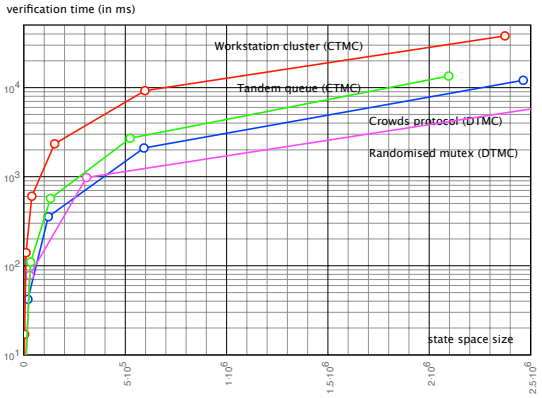


Figure 4: Efficiency of computing reachability probabilities versus the state space size

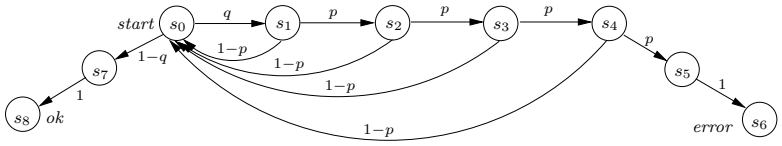
measure of interest prior to any computation, avoiding the consideration of parts of the state space that are irrelevant for the property of interest. In this way, computations have to be carried out only on the fragments of the state space that are relevant to the property of interest. In fact, this generalises the ideas put forward by Sanders and Meyer on variable-driven state space generation in the late 1980s [33].

Dependability evaluation. This measure-driven aspect is even more beneficial in the field of system *dependability evaluation*, a field tightly related to performance evaluation, but especially concerned with evaluating service continuity of computer systems. Questions like "under which system faults can a given service still be provided adequately?" are adressed, and typical measures of interest are system reliability and availability, cf. Table 1. Since the beginning of the 1980s this field has matured significantly, due to the introduction of state-oriented models and the invention of *uniformization* [44]. This facilitated the efficient analysis of time-dependent properties such as reliability or availability evaluation, in combination with high-level model specification techniques such as SPNs. The models that one could analyze now went well above the "standard models" based on reliability block diagrams or fault-trees.

The measures of interest in this field often involve costs, modeling the usage of resources. Extensions of stochastic processes with cost (or reward) functions give rise to a logic where in addition to, e.g., time bounds, conditions about the accumulated reward along an execution path can be imposed. Model checking still goes along the lines of Figure 2, but involves computational procedures that are more time-consuming.

One for free. Is that all? Not quite. An important problem with performance modeling regardless whether one aims at numerical evaluation or at simulation, is to check the functional correctness of the model. For a stochastic Petri net specification, place and transition invariants are exploited to check for deadlocks and liveness, among others. For a Markov chain model, graph-based algorithms are

The IPv4 zeroconf protocol is a simple protocol proposed by the IETF (RFC 3927), aimed at the self-configuration of IP network interfaces in ad-hoc networks. Such ad-hoc networks must be hot-pluggable and self-configuring. Among others, this means that when a new appliance, hitherto called a newcomer, is connecting to a network, it must be configured with a unique IP address automatically. The zeroconf protocol solves this task using randomization. A newcomer intending to join an existing network randomly selects an IP address, U say, out of the 65024 available addresses and broadcasts a message (called a probe) asking "Who owns the address U?". If an owner of U is present and does receive that message, it replies, to force the newcomer to randomly select another address. Due to message loss or busy hosts, messages may not arrive at some hosts. Therefore a newcomer is required to send a total of 4 probes, each followed by a listening period of 2 seconds before it may assume that a selected address is unused. Therefore, the newcomer can start using the selected IP address only after 8 seconds. Notably, there is a low probability risk that a newcomer may still end up using an already owned IP address, e.g., because all probes were lost. This situation, called address collision, is highly undesirable.



The protocol behaviour of a newcomer is easily modeled by a DTMC depicted above consisting of 9 states [9, 3]. The protocol starts in s_0 where the newcomer randomly chooses an IP address. With probability $q = m/65024$ the address is already owned, where m is the current size of the network. State s_i $(0 < i \leqslant 4)$ is reached after issuing the i-th probe. With probability p no reply is received during 2 seconds on a sent probe (as either the probe or its reply has been lost). State s_8 (labelled *ok*) indicates that eventually a unique address has been selected, while state s_6 (labelled *error*) corresponds to the undesirable situation of an address collision.

For such a model some typical example formulae are:

- On the long run, the protocol will have selected an address: $\mathbb{L}_{\geqslant 1}(ok \vee error)$.
- The probability to end up with an address collision is at most p': $\mathbb{P}_{\leqslant p'}(\Diamond error)$
- The probability to arrive at an unused address within k steps exceeds p': $\mathbb{P}_{>p'}(\Diamond^{[0,k]} ok)$

Many more measures including expected times and accumulated costs can be expressed using extensions of the base logic and model introduced here.

Figure 3: A simple model checking example: The Zeroconf protocol

used to check elementary properties. The good news when employing model checking is that we get this functionality for free. Using the same machinery for validating the measures of interest, functional properties can be checked. Probabilistic model checking thus provides two for the price of one: both performance/dependability analysis and checking functional properties. This forces the user to construct models with a high precision as any tiny inconsistency will be detected. Compare this to simulation model construction in NS2 or OPNET!

Nondeterminism. Sometimes this need for precision might seem as a burden, but it is a vehicle to force the modeler to make hidden assumptions explicit – or to leave them out. For instance, we have discussed the nondeterminism inherent in the join-the-shortest-queue idea, which – unless made concrete – implies that the underlying model is not a stochastic process. Stochastic models with nondeterminism are usually referred to as stochastic decision processes. In these models the future behaviour is not always determined by a unique probability distribution, but by selecting one from a set of them. Temporal logics and verification technology have been extended to this type of models with relative ease for CTL [8] and LTL [14, 36]. Actually, they constitute the genuine supermodel that comprises both the model checking and performance evaluation side as special cases: When transition systems are paired with Markov chains or Markov reward models, the model is known as Markov decision processes. For these models, performance model checking is still possible, but the checker now computes bounds on the performance, in the sense that however the nondeterminism is concretized, the concrete performance figure will stay within the calculated bounds. Whereas for the discrete-time setting, efficient model checking algorithms have been developed, this field is still relatively open in the continuous-time setting.

4. APPEALING APPLICATION AREAS

Over the last five years, a number of stochastic model checking tools have been developed, of which PRISM [20] is by far the most widely used. A number of well-known tools from the performance and dependability evaluation area, like tools for SPNs and stochastic process algebras have been extended with stochastic model-checking features. All these tools automatically generate a Markovian model of some sort, either using symbolic or sparse data structures.

With these tools, a wide variety of case studies have been carried out, amongst others, in application areas such as communication systems and protocols, embedded systems,

systems biology, hardware design, and security, as well as more "classical" performance and dependability studies.

Examples of the latter category, for which CTMCs are a very natural model, include the analysis of various classes of traditional queuing networks and even infinite-state variants thereof, fault-tolerant workstation clusters, and wireless access protocols such as IEEE 802.11. Also system survivability, i.e., the ability of a system (e.g., military or aircraft) to recover predefined service levels in a timely manner after the occurrence of disasters, has been precisely captured using a logic similar to that introduced before, and has been verified for Google-like file systems [12]. The evaluation of a wireless access protocol for ad-hoc networks using model checking could be carried out at far lower cost than using discrete-event simulations [32].

The popularity of Markovian models is rapidly growing due to their application potential in *systems biology*; the timing and probabilistic nature of CTMCs naturally reflect the operations of biological mechanisms such as molecular reactions. In fact, various biological systems have been studied by CTMC model checking in recent years, cf. [26]. Prominent examples include ribosome kinetics, signalling pathways, cell cycle control in Eukaryotes, and enzyme-catalyzed substrate conversion. In particular, the possibility to compute time-bounded reachability probabilities is of great importance here as traditional studies focus on steady-state behaviour. Another application area for CTMC model checking is *embedded systems* where the timeliness of communication between sensor and actuator devices, e.g., within cars or between high-speed trains, is of utmost importance. Stochastic model-checking techniques allow us to address the timeliness and the protocols' correctness from a single model. A nice example in this setting is dynamic power management in relation to job scheduling [31].

Examples for the discrete-time setting include several studies of the IPv4 Zeroconf protocol, cf. Figure 3, where next to the probability of eventually obtaining an unused IP address, extensions have been studied with costs, addressing issues such as the number of attempts needed to obtain such address. Security protocols are another important class of systems in which discrete randomness is exploited, e.g., by applying random routing to avoid information leakage. An interesting case is the Crowds protocol [34], a well-known security protocol that aims to hide the identity of web-browsing stations. Checking Markovian models with upto 10^7 states, did provide important information on quantifying the increase of confidence of an adversary when observing an internet packet of the same sender more than once. A novel case study in the field of nanotechnology applies stochastic model checking to quantify the reliability of a molecular switch with increasing memory array sizes [13]. Other natural cases for discrete-time probabilistic models are randomized protocols —in which probabilities are used to break ties— such as consensus and broadcast protocols, and medium access mechanisms such as Zigbee.

To conclude, an interesting case study using DTMCs with non-determinism is the analysis of the *Firewire protocol* (IEEE 1394). This protocol has been developed to allow "plug-and-play" network connectivity for multimedia consumer electronics in the home environment. A key component in IEEE 1394 is a leader election protocol (the "root contention protocol") that exploits a coin-tossing mechanism to break ties. Stochastic model checking revealed that using a *biased* coin instead of the typically used unbiased coin, speeds up the leader election process. This confirmed a conjecture in [35]. This insight would not have been found through "classical" qualitative verification.

5. CURRENT TRENDS AND CHALLENGES

Recent Turing award winner Edmund Clarke points out that probabilistic model checking is one of the brands of verification that requires further developments [41]. We list some of the current trends and major research challenges.

One of the major practical obstacles shared by model-based performance evaluation and model checking is the *state space explosion problem*. To combat the state space explosion problem, various techniques have been developed and successfully applied for model checking Kripke structures [11] (and the literature mentioned there).

For stochastic models the state space explosion problem is even more severe. This is rooted in the fact that the model checking algorithms for stochastic models rely on a combination of model checking techniques for non-stochastic systems, such as graph algorithms, but also mathematical, often numerical methods for calculating probabilities, such as linear equation solving or linear programming.

Many of the advanced techniques for very large non-stochastic models have been adapted to treat stochastic systems, including, for instance variations of decision diagrams, to represent large state spaces symbolically [30]. Complementary techniques attempt to abstract from irrelevant or redundant details in the model and to replace the model with a smaller, but "equivalent" one. Some of them rely on the concept of *lumpability* for stochastic processes, which in the formal verification setting is known as *bisimulation* quotienting, and where states with the same probabilistic behavior are collapsed into a single representative [16, 27].

Other advanced techniques to fight the state-explosion problem include symmetry exploitation [24], partial order reduction [5], or some form of abstraction [28], possibly combined with automatic refinement [15, 19]. All these approaches take inspiration in classical model checking advances, which often get much more intricate to realise, and raise interesting theoretical and practical challenges. Alltogether, they have advanced the field considerably in the ability to handle cases as the ones discussed in the preceding section.

An important feature of model checkers for non-stochastic systems is the generation of *counterexamples* for properties that have been refuted by the model checker. The principal situation is more difficult in the stochastic setting, as for probabilistic properties, say the requirement that a certain undesired event will appear with probability at most 10^{-3}, single "error traces" are not adequate. The generation and representation of counterexamples is therefore a topic of much increasing attention [17, 29] within the community.

To overcome the limitation to finite state spaces, much work has been done to treat *infinite-state* probabilistic systems, in many different flavors [1, 23].

Another topic of ongoing interest lies in combining probabilistic behaviour with continuous dynamics as in timed [25] or hybrid automata, but more work on the tool side is needed to assess the merits of these approaches faithfully. Theorem-proving techniques for analyzing probabilistic systems [21] is also a very promising direction. One of the major open technical problems is the treatment of models with *nonde-*

terminism and continuous distributions. Initial results are interesting but typically subject to (severe) restrictions.

As a final item, we mention the need to tailor the "general-purpose" probabilistic model checking techniques to special application areas. This covers the design of special modeling languages and logics that extend or adapt classical modeling languages and temporal logics by adding features that are specific for the application area.

Acknowledgement. We like to thank Andrea Bobbio, Gianfranco Ciardo, William Knottenbelt, Marta Kwiatkowska, Evgenia Smirni, and the anonymous reviewers for their valuable feedback.

6. REFERENCES

[1] P. Abdulla, N. Bertrand, A. Rabinovich, and Ph. Schnoebelen. Verification of probabilistic systems with faulty communication. *Inf. and Comp.*, 202(2):141–165, 2007.

[2] M. Ajmone Marsan, G. Conte, G. Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM TOCS*, 2(2), 93-122, 1984.

[3] S. Andova, H. Hermanns, and J.-P. Katoen. Discrete-time rewards model-checked. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, LNCS 2791:88–104, 2003.

[4] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. Model checking continuous-time Markov chains. *ACM TOCL*, 1(1):162–170, 2000.

[5] C. Baier, M. Größer, and F. Ciesinski. Partial order reduction for probabilistic systems. In *Quantitative Evaluation of Systems (QEST)*, pages 230–239. IEEE CS Press, 2004.

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

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

[8] A. Bianco and L. De Alfaro. Model checking of probabilistic and non-deterministic systems. In *Foundations of Softw. Technology and Theor. Comp. Science (FSTTCS)*, LNCS 1026: 499–513, 1995.

[9] H. Bohnenkamp, P. van der Stok, H. Hermanns, and F.W. Vaandrager. Cost optimisation of the IPv4 zeroconf protocol. In *Dependable Systems and Networks (DSN)*, pages 531–540, IEEE CS Press. 2003.

[10] G. Bolch, S. Greiner, H. de Meer, K.S. Trivedi. *Queueing Networks and Markov Chains*, Wiley, 1998.

[11] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

[12] L. Cloth and B.R. Haverkort. Model checking for survivability. In *Quantitative Evaluation of Systems (QEST)*, pages 145–154. IEEE CS Press, 2005.

[13] A. Coker, V. Taylor, D. Bhaduri, S. Shukla, A. Raychowdhury, and K. Roy Multijunction fault-tolerance architecture for nanoscale crossbar memories. *IEEE Trans. on Nanotechnology*, 7(2): 202–208, 2008.

[14] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *JACM*, 42(4):857–907, 1995.

[15] P.R. D'Argenio, B. Jeannet, H. Jensen, and K.G. Larsen. Reduction and refinement strategies for probabilistic analysis. In *PAPM-PROBMIV*, LNCS 2399: 335–372, 2002.

[16] S. Derisavi, H. Hermanns, and W.H. Sanders. Optimal state-space lumping in Markov chains. *Inf. Proc. Letters*, 87(6):309–315, 2003.

[17] T. Han, J.-P. Katoen, and B. Damman. Counterexamples in probabilistic model checking. In *IEEE TSE*, 35(2): 241–257, 2009.

[18] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Asp. of Comp.*, 6(5):512–535, 1994.

[19] H. Hermanns, B. Wachter, and L. Zhang. Probabilistic CEGAR. In *Computer-Aided Verification (CAV)*, LNCS 5123: 162–175, 2008.

[20] `www.prismmodelchecker.org`.

[21] J. Hurd, A. McIver, and C. Morgan. Probabilistic guarded commands mechanized in HOL. *Theor. Comp. Sc.*, 346(1):96–112, 2005.

[22] R. Jain. *The Art of Computer System Performance Analysis.* Wiley, 1991.

[23] A. Kucera, J. Esparza, and R. Mayr. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2(1), 2006.

[24] M.Z. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *Computer-Aided Verification (CAV)*, LNCS 4144: 238–248. 2006.

[25] M.Z. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29(1):33–78, 2006.

[26] M.Z. Kwiatkowska, G. Norman and D. Parker. Probabilistic model checking for systems biology. In *Symbolic Systems Biology*, Jones and Bartlett, 2010.

[27] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. and Comp.*, 94(1): 1–28, 1989.

[28] A. McIver and C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Springer, 2005.

[29] A. McIver, C. Morgan, and C. Gonzalia. Proofs and refutation for probabilistic systems. In *Formal Methods (FM)*, LNCS 5014, 2008.

[30] A. Miner and D. Parker. Symbolic representation and analysis of large probabilistic systems. In *Validation of Stochastic Systems. A Guide to Current Research*, LNCS 2925: 296–338, 2005.

[31] G. Norman, D. Parker, M.Z. Kwiatkowska, S.K. Shukla, R. Gupta. Using probabilistic model checking for dynamic power management. *Formal Asp. Comp.*, 17(2): 160-176, 2005.

[32] A. Remke, B.R. Haverkort, and L. Cloth. A versatile infinite-state Markov reward model to study bottlenecks in 2-hop ad hoc networks. In *Quantitative Evaluation of Systems (QEST)*, pages 63–72. IEEE CS Press, 2006.

[33] W.H. Sanders and J.F. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE J. on Selected Areas in Comms.*, 9(1):25–36, 1991.

[34] V. Shmatikov. Probabilistic model checking of an anonymity system. *J. of Computer Security*, 12(3/4):355–377, 2004.

[35] M. Stoelinga. Fun with FireWire: A comparative study of formal verification methods applied to the IEEE 1394 root contention protocol. *Formal Asp. Comp.*, 14(3):328–337, 2003.

[36] M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 327–338. IEEE CS Press, 1985.

The references below will – due to strict space limitations – not be part of the paper version in CACM, but will be part of the version available via the authors' websites.

7. REFERENCES

[37] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real time. *Inf. and Comp.*, 104(2):2–34, 1993.

[38] A. Biere, A. Cimatti, E.M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.

[39] J. Burch, E.M. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking 10^{20} states and beyond. *Inf. and Comp.*, 98(2):142–170, 1992.

[40] J.P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *CACM*, 16(9):527–531, 1973.

[41] E.M. Clarke. The birth of model checking. In *25 Years of Model Checking*, LNCS 5000: 1–26, 2008.

[42] A.I. Elwalid, D. Mitra. Effective bandwidth of general Markovian traffic sources and admission control of high speed networks. *IEEE/ACM Trans. on Networking* 1(3): 329–343, 1993.

[43] W. Fischer, K.S. Meier-Hellstern. The Markov-Modulated Poisson Process (MMPP) Cookbook. *Perform. Eval.* 18(2): 149–171, 1993.

[44] D. Gross and D.R. Miller. The randomization technique as a modelling tool and solution procedure for transient Markov processes. *Operations Research*, 32(2):343–361, 1984.

[45] G. Horton, V.G. Kulkarni, D.M. Nicol, K.S. Trivedi. Fluid stochastic Petri nets: Theory, applications, and solution techniques. *Eur. J. of Operations Research*, 105: 184–201, 1998.

[46] J.-Y. LeBoudec, P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet. LNCS* 2050, 2001.

[47] W.E. Leland, M.S. Taqqu, W. Willinger D.V. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Trans. on Networking* 2(1), 1–15, 1994.

[48] M.F. Neuts. *Matrix-Geometric Solutions in Stochastic Models*. Johns Hopkins University Press, 1981.

[49] D. Peled, V. Pratt, and G. Holzmann, editors. *Partial Order Methods in Verification*, volume 29(10) of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. AMS 1997.

[50] A. Pnueli. The temporal logic of programs. In *18th IEEE Symp. on the Foundations of Computer Science (FOCS)*, pages 46–57. IEEE CS Press, 1977.

[51] M. Reiser and S.S. Lavenberg. Mean-value analysis of closed multichain queuing networks. *JACM*, 27(2):313–322, 1980.

[52] J.A. Rolia, K.C. Sevcik. The method of layers. *IEEE TSE* 21(8): 689–700, 1995.

[53] C.H. West. An automated technique for communications protocol validation. *IEEE TCOM*, 26(8):1271–1275, 1978.