Scalable Stochastic Modelling for Resilience

Jeremy T. Bradley[1], Lucia Cloth[2], Richard Hayden[1], Leïla Kloul[3]
Philipp Reinecke[4], Markus Siegle[5], Nigel Thomas[6], and Katinka Wolter[4]

[1] Department of Computing, Imperial College London, UK
[2] Department of Applied Information Technology, GU Tech, Oman
[3] Laboratoire PRiSM, Université de Versailles, France
[4] Institute of Computer Science, Freie Universität Berlin, Germany
[5] Department of Computer Science, Universität der Bundeswehr München, Germany
[6] School of Computing Science, Newcastle University

Abstract. This chapter summarises techniques that are suitable for performance and resilience modelling and analysis of massive stochastic systems. We will introduce scalable techniques that can be applied to models constructed using DTMCs and CTMCs as well as compositional formalisms such as stochastic automata networks, stochastic process algebras and queueing networks. We will briefly show how techniques such as mean value analysis, mean-field analysis, symbolic data structures and fluid analysis can be used to analyse massive models specifically for resilience in networks, communication and computer architectures.

1 Introduction

The techniques presented in this chapter represent the state of the art in performance and resilience analysis when it comes to coping with massive state-space models. Many existing analysis techniques rely on generating underlying stochastic models, such as continuous-time Markov chains. Where there is too close a correspondence between the state space of the model and that of the underlying stochastic process, the state-space explosion in the former can lead to intractability in the latter. The presented techniques in this chapter were chosen as they represent instances of the main approaches to state-space reduction in stochastic systems: aggregation, decomposition, symbolic representation and continuum approximation.

We realise that accurate resilience analysis relies on a detailed and complex model. This kind of model generates huge state spaces and computation time if handled naïvely. In this chapter, we are specifically interested in analysis techniques that side-step the state space explosion problem by making use of efficient representation mechanisms. This is necessary if we are to make headway in directly analysing problems in mobile networks (Chapter 29), critical infrastructures (Chapter 7 and 30) and Cloud systems (Chapter 13 and 28).

We summarise the techniques presented below. In each case, we indicate what type of analysis result can be expected to be obtained using this method. It is important to understand how these techniques are relevant and useful to the understanding of resilience of a system. The types of analysis that can be tackled using these techniques can broadly be put into three categories. In the context of resilience analysis: *steady state distributions* are useful for calculating the probability that a fault state is ever reached; *transient distributions* are useful for calculating the probability that a fault state is entered in a particular time window; and *response* or *passage time analysis* is used to specify service level agreements, along the lines of *"The round-trip response of a service request to a virtualised environment should be less that 1.8 seconds, with probability 0.95."*

Decomposition and phase-type representation We start with a technique which combines simulation and decomposition, to generate simple approximate models based on phase-type representation of key portions of a system's operation (Section 2). A grey-box technique such as this avoids explicit investigation or representation of individual states in the system. This will be a useful technique for deriving some aggregate steady-state and transient measures but will probably be most widely used for response-time results.

Product forms and MVA In Section 3, we explore two powerful techniques from queueing theory, product form and mean value analysis (MVA). These powerful techniques can be applied to very large or infinite state systems. They explore balanced flows of traffic between components and in doing so permit compositional rather than system-wide analysis. Product forms will tend to lead to rapid steady-state distribution results, while MVA can calculate mean throughput, response time and job occupancy measures in a system.

Tensor representation Section 4 presents a decomposition technique which avoids the construction of the underlying explicit state space of a continuous-time Markov chain. Instead smaller submatrices are constructed which broadly reflect the components of the stochastic automata network. Analysis techniques exist which maintain the decomposed tensor representation while producing accurate steady-state distribution results.

Symbolic representation The symbolic representation of a stochastic model is described in Section 5. Multi-terminal Binary Decision Diagrams are used to encode efficiently the real rate values of a variety of stochastic models, including CTMCs, DTMCs and MDPs (Markov Decision Processes). Steady-state analysis, transient analysis and passage-time analysis are all possible using exclusively MTBDD-based algorithms.

Mean-field analysis In Section 6, we present developments in mean-field analysis as applied to massively distributed systems that are made up of identical communicating components. Applied to discrete-time systems, mean-field techniques generate sets of deterministic mean-difference equations (MDEs). Solving these MDEs gives access to both transient and passage-time measures. They provide an interesting comparison to the continuous counterpart, fluid analysis, described in the next section.

Fluid analysis Finally, in Section 7, we show how a continuum approximation of the dynamics of a stochastic system can be encoded in a system of ordinary differential equations (ODEs). Fluid analysis can be applied to large distributed systems that are comprised of groups of components of different types. Usually generated from a higher level formalism such as a stochastic process algebra, the ODEs can encode steady-state, transient and passage time questions. Higher moments of various model quantities are also available to capture measure accuracy.

2 Efficient model representation and simulation

Discrete-event simulation is a widely-used approach for evaluating system properties such as response-time. Compared to analytical closed-form approaches, discrete-event simulation has the advantage of allowing the construction and evaluation of highly-detailed models where the modelled behaviour is not limited by the constraints of the formalism.

Unfortunately, the ability to easily build highly-detailed models often leads to scalability problems. While simulations modelling simple scenarios can be evaluated efficiently, increasing the complexity of the scenario quickly results in simulation models whose evaluation takes too long for the results to be useful. In this case, one obvious solution would be to start from scratch with a less detailed model with better scalability. However, this solution is often not desirable, since the results from a less detailed model may not be sufficiently accurate, and because of time and cost constraints on the modelling process itself.

Increasing model complexity is a well-known problem with analytical approaches as well. These approaches often suffer from state-space explosion as the system to be studied becomes more complex. Various solutions for this problem have been proposed [1]. In particular, decomposition/aggregation methods allow the solution of highly-complex models by splitting the model into submodels, solving the submodels independently, and aggregating the submodel solutions into a solution for the whole system. This approach may be used to reduce both processing time and memory requirements.

The general idea of decomposition and aggregation can also be applied to discrete-event simulations. In this section we describe a hybrid approach that combines decomposition and approximation using stochastic models to allow faster evaluation of discrete-event simulation models. With this approach, the simulation model is split into blocks whose behaviour can be approximated by phase-type (PH) distributions (cf. chapter on phase-type distributions). The phase-type distributions replace the approximated blocks in the simulation. The behaviour of the approximated blocks can then be reproduced by drawing random variates from the approximating phase-type distributions. Since this is more efficient than detailed discrete-event simulation, simulation of the whole system becomes

much faster. This method can be applied when delay metrics are to be determined. Whether the method also applies for other metrics, such as throughput, availability or state probabilities in general, is as of yet unknown. The approach consists of the following steps:

1. **Decomposition of the simulation model into blocks that affect the metrics.** Starting from the complete simulation, one must identify parts that can be approximated by phase-type distributions. These blocks should satisfy a number of criteria: it should be simple to separate the simulation into the blocks, and the blocks should be chosen such that increasing the complexity of the simulation corresponds to multiple application of identical blocks. This step is shown in Figure 1, where block A of the simulation is affected by block B: The effect of block B on the metrics depends on its internal block B1. Increasing the size of system B corresponds to using multiple instances of B1 within B.

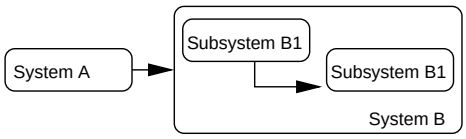


Fig. 1. Example decomposition of a simulation.

2. **Evaluation of simulation blocks.** In order to approximate the behaviour of the simulation blocks identified in step 1 by phase-type distributions, data is needed that shows the effect of the blocks. Typically, this data will be obtained from detailed simulations of the individual blocks, but if a block corresponds to a system where data from measurements is available, such data can also be used.
3. **Approximation using phase-type distributions.** The data obtained in step 2 is approximated by a phase-type distribution. The general process and tools for fitting PH distributions to data sets is described in detail in Chapter 1. For the application in the hybrid approach, the focus should be on capturing those properties well that strongly affect the metric. Furthermore, one should choose phase-type distributions that enable efficient random-variate generation, so as to be able to reproduce the behaviour of the approximated building-blocks efficiently.
4. **Integration of the phase-type distributions into the simulation model.** The phase-type models must be integrated into the simulation. For delay metrics, this requires drawing random variates from the distribution and delaying events caused by the approximated system blocks according to these variates. For common discrete-event simulation toolkits such as NS-2 [2] and OMNeT++ [3], the libphprng [4] library provides the necessary routines for generating PH-distributed random numbers.

5. **System evaluation.** The whole system can now be evaluated. In order to scale system size/complexity, the number of blocks containing the approximating PH distribution is increased. As random-variate generation from a PH distribution is typically more efficient than detailed simulation of the blocks, the model can be expected to scale much better to higher numbers of building-blocks. On the other hand, the approximation process introduces an error, since the PH distribution does not represent all behaviour of the detailed model.

2.1 Illustrative example

In [5] we have investigated timing behaviour in a tree-structured network across a variable number of identical switches. The topology is shown in Figure 2 and each data stream is transmitted in a straight feed-forward fashion.

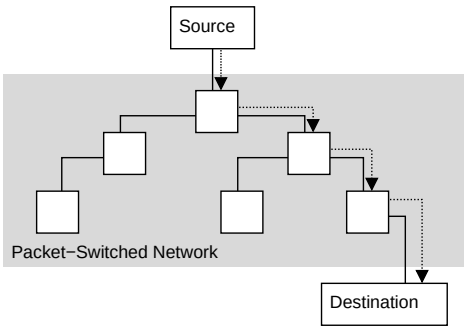


Fig. 2. Data stream in packet-switched network with tree topology

Let us illustrate the approximation technique for the given example. In this example, the transmission delays encountered on the path between the source and the destination are investigated. More precisely, the metric of interest is the 1% quantile of packet delay variation (PDV)[7] of the high-priority packets in a network transporting two types of packets, the high priority traffic and the low priority background data. This implies that the subsystem models must only represent transmission delays, and we can abstract away from other aspects of the network, such as correctness of the content of transmitted messages.

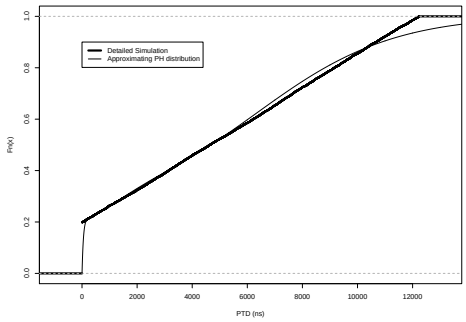
In order to evaluate this scenario, a simulation has been built using the discrete-event network simulator NS-2 [2]. In this simulation, we model the internal behaviour of switches in a detailed manner, and generate streams of high and low priority packets. Then the time is observed that it takes for high-priority packets

[7] Packet delay variation is defined as the difference between the shortest and the longest transmission time, where lost packets are ignored.

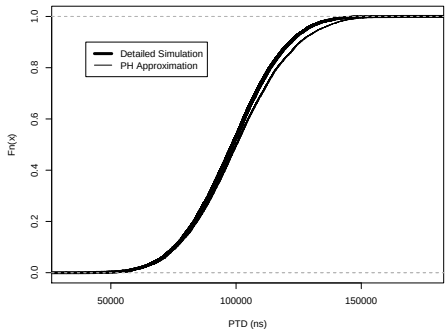
to traverse the network from the source to the destination. As illustrated by the upper curve in Figure 4(a), simulation run-times with this approach increase quickly as the number of switches is increased. Therefore the approximation method is applied as follows:

1. **Decomposition of the simulation model into blocks that affect the metrics.** In this scenario, the obvious block is the individual switch. For simplicity, we assume that the network consists of switches whose behaviour with respect to transmission delay is identical.
2. **Evaluation of simulation blocks.** The transmission delays incurred by each switch must be assessed. As a network consisting of only one switch can still be simulated within an acceptable time, transmission delays for a long simulation run with only one such switch between source and destination are obtained.
3. **Approximation using phase-type distributions.** We fit an acyclic phase-type distribution using the PhFIT tool [6] to the transmission delay of the high-priority stream such that the resulting distribution represents the 1% quantile well. See Chapter 1 for more details on phase-type distributions. The phase-type distribution obtained here represents the transmission delay distribution of a single switch. Figure 3(a) shows the packet delay distribution from step 2 and the cumulative density function of the phase-type distribution fitted to the data. Note that the distribution fits the lower quantiles well and tends to diverge only on the higher quantiles.
4. **Integration of the phase-type distributions into the simulation model.** Now, the phase-type distribution for the packet delay of the switch can be used to simulate the behaviour of the switch. To this end, we build a simple queueing station whose service-time distribution is given by the phase-type distribution fitted to the data. Packets entering the station system are delayed according to the service-time distribution. Note that the presence of a queue is only dictated by the fact that we cannot drop packets, and that this queue does not correspond to any part of the original model. In the considered scenario high-priority packets are sent very infrequently, and therefore queueing is highly unlikely. If the arrival rate of high-priority packets was higher, queueing might occur. The resulting queueing delay would increase the error caused by the approximation, since this queueing is not part of the original simulation.
5. **System evaluation.** The system can now be simulated by transmitting high-priority packets from the source to the destination. Note that it is not necessary to simulate the low-priority stream anymore, since its effect on the packet delay variation is already captured in the service-time distribution.

The advantages of the hybrid approach can be evaluated using a scenario where the number of switches that the data stream has to traverse is increased from 1 to 20.



(a) CDF of simulation data for 1 switch and associated PH approximation.



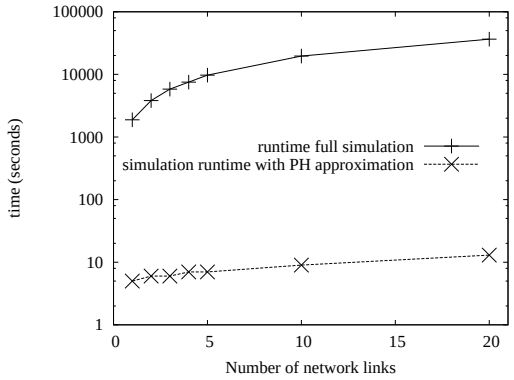
(b) CDF of simulation data and PH approximation with 20 links using full simulation and PH approximation.

Fig. 3. Comparison of simulation data and PH approximation.

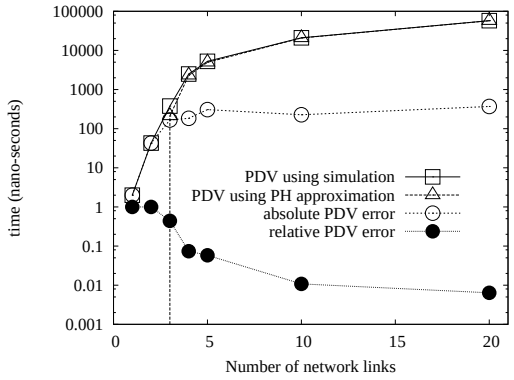
Figure 4(a) shows run-times for the detailed and approximating simulations for increasing number n of links. Note that the run-times for the detailed simulation increase sharply and reach values in the order of days, while run-times with PH approximation stay in the order of hours, even for a high number of links. Figure 4(b) illustrates the error incurred by using PH approximations for the switches instead of simulating them in detail. As expected, the error does increase with n, but it seems to converge, and the relative error even decreases.

2.2 Outlook

The decomposition and approximation approach should be extended in various directions. This has not been done yet and there might arise problems that cannot be foreseen. First, the system model need not be a simulation model. Any stochastic discrete event model, such as a CTMC, should be applicable too.



(a) Simulation runtimes for full simulation and with PH approximation.



(b) PDV error for full simulation and with PH approximation.

Fig. 4. Evaluation of simulation time and the error introduced by approximation.

Second, the approximation technique need not be a PH distribution. Simpler approximations could consist in Bernoulli variables, and more sophisticated approximation may use correlated stochastic processes, that are still smaller than the original model. Third, metrics other than timing metrics should be considered. Often of interest are system or service availability, throughput, or failure characteristics.

3 Product-form solution and mean value analysis

One approach to tackling the state space explosion problem common to all compositional modelling techniques is to break the model into smaller parts that can then be solved separately and the solutions combined in some way to give measures for the whole system. This approach is generally known as model decomposition.

One of the most powerful model decomposition techniques are so called, *product-form solutions*. Essentially, a product-form is a decomposed solution where the steady state distribution of a whole system can be found by multiplying the marginal distributions of its components. Thus, for a system described by the pair $\{S_1, S_2\}$, where S_i is the local state of component C_i a product form solution would have the form

$$\pi(S_1, S_2) = \frac{1}{B}\pi(S_1)\pi(S_2)$$

where $1/B$ is the *normalising constant* ($B \leq 1$), which is necessary if there are combinations of possible local states which are prohibited in the system evolution.

The conditions for such a solution to exist are clearly going to be restrictive, and as such product-form solutions are applicable only in a relatively small number of cases. Despite the restrictions, product-forms are extremely powerful and so the quest for new solutions in stochastic networks has been a major research area in performance modelling for over 30 years, giving rise to a number of seminal results, such as Jackson queueing networks [7] and the BCMP result [8] for closed queueing networks.

A Jackson network is a simple open queueing network with Poisson arrivals, negative exponentially distributed service times and unbounded FCFS queues, where the routing of jobs from one node to another is strictly *a priori*. For stability it is also required that the utilisation at each node is strictly less than one. In such a network each node can be considered as an independent FCFS $M/M/k$ queue and the steady state probability of being in any given system state can be found simply by multiplying together the marginal probabilities of each node's local state.

Jackson's result does not apply to closed networks of $M/M/k$ queues (where no jobs may enter or leave the network) because the population of a closed network

is bounded. This restriction was overcome by Gordon and Newell [9], with the introduction of the normalising constant $1/G(K)$, where K is the population size. If the state is described by the tuple $\mathbf{S} = \{S_1, \ldots, S_n\}$ then $G(K)$ is given by,

$$G(K) = \sum_{\mathbf{S}} \prod_{i=1}^{n} \pi(S_i)$$

The results of Jackson and Gordon and Newell were subsequently generalised by Baskett, Chandy, Muntz and Palacios [8] to allow four possible classes of network for which the product-form solution holds. The first class is FCFS queues with negative exponentially distributed service times. The service rate may be state dependent, which allows the network to be open or closed. The subsequent classes slightly relax the condition of negative exponentially distributed service times, as long as the queueing discipline is either processor sharing, infinite server or LCFS with pre-emptive resume.

The BCMP characterisation of product-form networks greatly extended the potential for applying product-form solutions to real world problems. Subsequently there have been many further results extending the class of queueing networks amenable to product-form solution under specific conditions, for different kinds of queueing network and for different properties. One case worthy of special mention is the work on product-form G-networks by Gelenbe [10]. G-networks are a variant of queueing networks that allow so-called *negative customers*. A negative customer may potentially remove a job from the queue into which it arrives. As well as having external arrivals of negative customers, a normal (or *positive*) customer may become a negative customer with some probability following completion of service. Subject to similar conditions as Jackson's result described above, a G-network can be shown to exhibit a product form solution. This result is more surprising than it might naively appear, as previously it was assumed that there was a relationship between the product-form solution and the existence of *partial balance*. Partial balance does not hold in G-networks, and so the proof of the product form solution changed the understanding of the conditions for product-form solution.

Most attention has been given to queueing networks and their variants (such as G-networks), but there have also been other significant examples, for example [11–13]. Many of the approaches to efficiently solving stochastic process algebra models have been based on concepts of decomposition originally derived for queueing networks [14]. Applying such approaches to stochastic process algebra allows the concepts to be understood in a more general modelling framework and applied to non-queueing models. More recently the Reversed Compound Agent Theorem (RCAT) has been developed [15]. RCAT is a compositional result that finds the reversed stationary Markov process of a cooperation between two interacting components, under syntactically checkable conditions [15, 16]. From this a product-form follows simply. RCAT thereby provides an alternative methodology that unifies many product-forms, far beyond those for queueing networks.

Even when a product-form solution exists, or an approximation to a product-form can be derived, obtaining a numerical solution may still be computationally expensive. In particular, finding the normalising constant can be costly in general. Mean value analysis (MVA) [17] depends on the application of the *arrival theorem*, first derived independently by Sevcik and Mitrani [18] and Lavenberg and Reiser [19]. This theorem states that, subject to certain conditions, a job arriving in a queue will, on average, observe the queue to be in its steady state average behaviour. Combining the arrival theorem with Little's law gives rise to a method for deriving average performance metrics based on steady state averages directly from the queueing network specification, without the need to derive any of the underlying Markov chain.

Stated simply, the basic MVA algorithm, when the population size is N, is as follows:

1. Set the population size, n, to be 1.
2. Compute the delay this single job will experience at each node as it traverses the otherwise empty network (i.e. the average service time for one job).
3. Hence compute the probability that this job will be at any given node at a randomly observed instant. This gives the average queue length at each node when the total population consists of one job.
4. Increment the population size, n.
5. Compute the delay an arriving job will experience at each node (i.e. the average service time for one job plus average service time for the average number of jobs in the queue when the population is $n-1$).
6. Hence compute the average queue length at each node when the total population consists of n jobs.
7. If $n < N$ then go back to step 4.

As such it is relatively computationally efficient as long as the population size is not excessively large. The computational cost for solving a network with a given structure grows linearly with N.

There are many generalisations and approximate solutions of the original mean value analysis algorithm. MVA has been applied to many classes of queueing network, as well as other formalisms, including stochastic process algebra [20]. The key observation in applying MVA to stochastic process algebra is that repeated instances of a component in parallel may be treated as jobs in a queueing network. If the interaction of these components with other components conforms to some simple set of restrictions, then a version of the arrival theorem can be derived relating a component evolving between derivatives (or behaviours) to the steady state solution of a system with one fewer instance of the component.

4 Tensor Representation

The *tensor representation* or *compact representation* has been used for some time as a means to address the problem of state space explosion to which state-based performance modelling formalisms are prone. This technique which aims to keep the size of the model representation as small as possible falls into the *largeness avoidance* category of techniques which also includes techniques such as decomposition, aggregation and symbolic encodings [21]. However, unlike the aggregation which can result in a significant reduction in the size of the state space, the tensor representation is not a state space reduction method, but rather an alternative approach to state space explosion which handles the model solution in a decomposed form. The matrix representation of the Markov process underlying a performance model may be decomposed so that the state space of the model, and its dynamics, are not represented by a single matrix but by a number of smaller matrices. Nevertheless the model is solved as a single entity and the solution is exact, unlike the decomposition technique which, generally, gives rise to approximate solution of the original model [22].

The tensor representation has been developed for several state-based modelling formalisms. The pioneering work in this area was carried out, in 1984, by Plateau on Stochastic Automata Networks (SANs) [23]. Using a technique based on tensor or *Kronecker* algebra, it has been proved [23–25] that this method automatically provides an analytic derivation of a decomposed form of the generator matrix called the *descriptor*. Compared to a monolithic description of the generator, the structure of this descriptor leads to a considerable reduction in memory requirements during the model solution. Moreover, solution techniques have been adapted to this representation [26–28].

In the following, we present the tensor representation in the context of the SAN approach and show how to derive the descriptor expression for the SAN model of the leacky bucket, an admission control mechanism in ATM networks. We finally discuss the impact of the tensor representation on both the memory requirements and the computation time of the matrix-vector multiplications when solving the model, and this regardless of the modelling formalism used.

4.1 Stochastic Automata Networks

In the SAN approach, a system is represented by a number of *automata*, each automaton capturing the dynamic behaviour of a component of the system. Within an individual automaton, the behaviour of a component is captured as a set of *states* and *events* causing transition from one state to another. A label associated with each transition allows us to specify the type of the event, and its occurrence date and probability [29]. The transitions in the network can be of two types: *local* or *synchronised*. A local transition occurs only in an automaton whereas a synchronised transition occurs in several automata at the same time.

More formally, a SAN is a set of N automata in which each automaton A_i, $1 \leq i \leq N$, is defined by the tuple (S_i, L, Q_i) where

- S_i is the set of states of the automaton,
- L is the set of labels. Each label $l \in L$ is a list that may contain either a function τ, or a list of tuples (e, τ_e, p_e), or both of them such that:
  - e is the name of a synchronising event or synchronisation,
  - τ and τ_e are the transition rates, functions defined from $\Pi_{i=1}^{N} S_i$ to $\mathbb{R}^+$,
  - p_e is the probability transition function defining a conditional routing probability on e between local states.
- Q_i is the transition function which associates a label from L with every arc of automaton A_i.

A label on an edge allows us to specify the type and the rate of the transition as follows:

- If $Q_i(x_i, y_i)$ contains a function τ, then we have a local transition to A_i between states x_i and y_i. If τ is not a constant, the transition is still local to automaton A_i, but its rate depends on the state of other automata of the network.
- If $(e, \tau_e, p_e(x_i, y_i)) \in Q_i(x_i, y_i)$, then the transition between states x_i and y_i is a synchronised transition, e and τ_e being the name and the rate of the transition of the synchronising event. $p_e(x_i, y_i)$ is the routing probability between local states x_i and y_i. The distinction between the rate and the probability is required because the first must be unique for a given synchronising event, thus the same on all concerned automata, whilst the probabilities may, and generally will, differ. The rate of synchronising events are determined at the global level.

4.2 The descriptor

In the seminal paper [24], Plateau proved that the generator matrix of the Markov process underlying a SAN model can be analytically represented using Kronecker algebra. This matrix is automatically derived from the SAN description, using the individual automata to generate the sub-matrices in the tensor expression. It has been proved in [23, 24, 29] that, if the states are in a lexicographic order, then the generator matrix Q of the Markov process associated with a continuous-time SAN model is given by:

$$Q = \bigoplus_{i=1}^{N} F_i + \sum_{e \in \varepsilon} \tau_e \left(\bigotimes_{i=1}^{N} R_{i,e} - \bigotimes_{i=1}^{N} \overline{R}_{i,e} \right) \tag{1}$$

where

- N is the total number of automata in the network.
- ε is the set of synchronisations.
- F_i is the local transition matrix of automaton $\mathcal{A}_i$ without synchronisations.
- $R_{i,e}$ is the transition matrix of automaton $\mathcal{A}_i$ due to synchronisation e whose rate is τ_e.
- $\overline{R}_{i,e}$ is a matrix representing the normalisation associated with the synchronisation e on automaton $\mathcal{A}_i$.
- $\bigoplus$ and $\bigotimes$ denote the tensor sum and product, respectively.

Unlike the local transition matrices F_i, the synchronising matrices $R_{i,e}$ are not generators, that is their rows do not sum to zero. The diagonal corrector matrices $\overline{R}_{i,e}$ have been introduced to normalise these synchronising matrices.

In discrete-time, the transition matrix of the Markov process underlying a SAN model is given by an expression similar to equation 1 where the tensor sum $\bigoplus$ is replaced by the tensor product $\bigotimes$. Applying the tensor product on the local transition matrices F_i allows us to catch the phenomenon characterising the discrete-time, that is the occurrence of several events at the same time.

In both the continuous and discrete-time, the solution of the model, that is the steady-state distribution, can then be achieved via the corresponding tensor expression of sub-matrices; the complete generator or transition matrix does not need to be generated.

4.3 Application

The *leaky bucket* is the admission control mechanism developed for ATM networks [30]. Its simplest version [31] consists of two buffers B_c and B_t (see Fig. 5). Whilst the former is used to store the user's data cells (packets), the latter is dedicated to the tokens. At its arrival to the access buffer, a cell is either lost if the buffer is full or stored before being served. The service of a cell consists of assigning to it a token taken from buffer B_t. For the cell, this token constitutes its access permit to the network.

The generation rate of the tokens is equal to either the average throughput or the peak cell rate characterising the user's stream. Therefore, if there are no tokens in B_t while data cells are still arriving to B_c, then the user's throughput does not conform to the throughput he has initially specified. The cells in B_c will have to wait until new tokens are generated and all the cells arriving while B_c is full are lost.

The SAN model As the data packets (cells), in ATM networks, have the same size, a discrete-time performance analysis of the leacky-bucket would be more appropriate. However, in order to keep the global automata simple, the SAN model is built in continuous-time. The model parameters are the following:

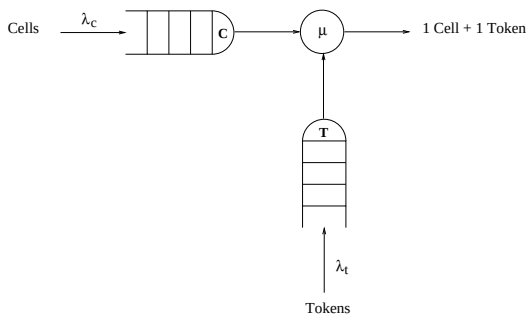


Fig. 5. The leaky bucket mechanism

- cell arrivals to B_c according to a Poisson process of parameter λ_c,
- token arrivals to B_t according to a constant distribution of parameter λ_t,
- cell service times exponentially distributed with rate μ.

The SAN modelling the leaky bucket mechanism consists of two automata, A_1 and A_2 [31]. Automaton A_1 models the number of cells in B_c and A_2 models the number of tokens in B_t. We assume the buffer size limited to two cells. This size remains however sufficient to represent all possible transitions. Thus both A_1 and A_2 have three states, noted s_0, s_1 and s_2.

In the modelled system, the possible events are of three types: the cell arrival with rate λ_c, the token arrival with rate λ_t and the simultaneous departure of a cell and a token with rate μ. Whilst the two first types of events are local events to automaton A_1, and A_2 respectively, the third type of events, noted e_t, is a synchronising event between A_1 and A_2, since it has an impact on both the number of cells in B_c and the number of tokens in B_t. The SAN model $\{A_1, A_2\}$ is depicted in Fig. 6.

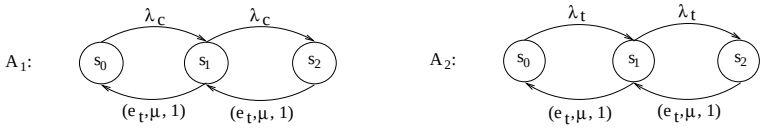


Fig. 6. The SAN model

The descriptor matrices We first build F_1 and F_2, the matrices of the local transitions associated with automaton A_1 and A_2, respectively. These matrices

are the following:

$$F_1 = \begin{pmatrix} -\lambda_c & \lambda_c & 0 \\ 0 & -\lambda_c & \lambda_c \\ 0 & 0 & 0 \end{pmatrix} \qquad F_2 = \begin{pmatrix} -\lambda_t & \lambda_t & 0 \\ 0 & -\lambda_t & \lambda_t \\ 0 & 0 & 0 \end{pmatrix}$$

As only one synchronising event has an impact on automaton A_1, we have only a single matrix of synchronised transitions, noted R_{1,e_t}, for this automation. By considering the associated normalisation matrix $\overline{R}_{1,e_t}$, we have:

$$R_{1,e_t} = \begin{pmatrix} 0\,0\,0 \\ 1\,0\,0 \\ 0\,1\,0 \end{pmatrix} \qquad \overline{R}_{1,e_t} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

Similarly, only one synchronising event has an impact on automaton A_2. Thus we have a single matrix of synchronised transitions, noted R_{2,e_t} for this automation.

$$R_{2,e_t} = \begin{pmatrix} 0\,0\,0 \\ 1\,0\,0 \\ 0\,1\,0 \end{pmatrix} \qquad \overline{R}_{2,e_t} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

Note that the transition rate of the synchronising event e_t, that is $\tau_{e_t} = \mu$ in the descriptor equation 1, is not reported in matrices R_{i,e_t}, $i = 1, 2$; only the transition probabilities are reported. As in continuous-time, the rate of a synchronising event is unique, and thus the same on all automata involved in the synchronisation, this rate appears only once, when the tensor product of matrices R_{i,e_t}, $i = 1, 2$, is performed.

Once all the matrices built, the elements of the generator associated with the SAN model $\{A_1, A_2\}$ can be computed, using equation 1. Thus the complete generator, which is the 9×9 matrix given below, does not need to be generated.

$$Q = \left(\begin{array}{c|ccccccccc} & 0,0 & 0,1 & 0,2 & 1,0 & 1,1 & 1,2 & 2,0 & 2,1 & 2,2 \\ \hline 0,0 & -\lambda & \lambda_t & & \lambda_c & & & & & \\ 0,1 & & -\lambda & \lambda_t & & \lambda_c & & & & \\ 0,2 & & & -\lambda_c & & & \lambda_c & & & \\ 1,0 & & & & -\lambda & \lambda_t & & \lambda_c & & \\ 1,1 & \mu & & & & -(\lambda+\mu) & \lambda_t & & \lambda_c & \\ 1,2 & & \mu & & & & -(\lambda_c+\mu) & & & \lambda_c \\ 2,0 & & & & & & & -\lambda_t & \lambda_t & \\ 2,1 & & & & \mu & & & & -(\lambda_t+\mu) & \lambda_t \\ 2,2 & & & & & \mu & & & & -\mu \end{array}\right)$$

In this representation, where $\lambda = \lambda_c + \lambda_t$, a global state (C, T) consists of the number of cells C and the number of tokens T. Thus, state $(1, 2)$, for example, refers to the system global state when there are 1 cell in B_c and 2 tokens in B_t. Note that all the global states are reachable.

4.4 Memory requirements

Following the development of the tensor representation for the SAN models, Kronecker representation techniques have been proposed for several other state-based performance modelling formalisms such as Petri net based-formalisms [32–40] and stochastic process algebra [41, 42]. In all these models, the *size* of the state space is open to several interpretations:

- the physical space T needed to store the model using the tensor representation;
- the size of the state space $\hat{S}$ of the cartesian product of the model components;
- the size of the reachable state space S.

In general, in the Kronecker representation, the cartesian product space $\hat{S}$ is represented, not the reachable state space S. When $|S| = |\hat{S}|$, the benefit of using the tensor representation may be enormous compared to an explicit saving of the generator as a sparse matrix. Consider, for example, a model which consists of N components and where n_i is the size of component i, $i = 1, \ldots, N$. If the generator is full (no zero elements), the memory needs of the tensor representation are given by $\sum_{i=1}^{N} n_i^2$ whereas the memory requirements of the sparse matrix representation are of the order of $(\Pi_{i=1}^{N} n_i)^2$.

When $|S| \ll |\hat{S}|$, the benefit of the tensor representation may be lost because of the unreachable states. If the probability vectors used in the vector-descriptor multiplications are the extended vectors $\hat{\pi}$, that is with an entry for each unreachable state in $\hat{S}$, the benefit of the tensor representation is lost not memory-wise only, but also because of unnecessary computations when solving the model. Therefore, the probability vectors used in the vector-descriptor multiplications must be reduced to the reachable states entries only (π). While the sparse matrix representation avoids unnecessary computations when solving the model, it remains a memory consuming representation, specially when dealing with big models.

In all the performance modelling formalisms for which a tensor representation has been developed (for instance, SAN, GSPN, PEPA), the model components are connected by synchronisations and/or functions. From previous work on SAN, we know that the use of functions has a positive effect on both the size of the tensor representation and the size of the product state space. In particular, if we remove a function it is generally necessary to introduce an additional component. If the new component has two or more states then we increase both spaces T and $\hat{S}$. However this should not change the reachable state space S. To do so fundamental changes have to be considered in the model.

Sometimes, the use of functions, like in process algebra PEPA [43], allows the tensor representation to be more direct and similar to the one obtained by Plateau

for the SANs. The functional dependency on the state of a component can capture the different apparent rates that the component may express with respect to an action type. There is an implicit assumption that an action type uniquely defines a synchronisation event at the transition system level. This will not generally be the case without restrictions on the use of types within cooperation sets [42, 43].

4.5 The model solution

The space efficiency of the tensor representation is obtained at the expense of an increased computation time. Moreover, the presence of functional rates in the components of a model introduces an extra computing time during the matrix-vector multiplications. Indeed when a model contains functional rates, an appropriate numerical value has to be recomputed each time a functional rate is needed. Thus the presence of the functional rates may constitute a determinant factor in the computing time requirements.

In [44, 26, 24], an efficient vector-descriptor multiplication algorithm, known as the *shuffle* algorithm has been developed to be used when solving the stationary distribution. This algorithm is the basic step in iterative methods such as the Power method and Generalised Minimum Residual (GMRES) method [45]. However, the algorithm, which is very efficient when $|S| = |\hat{S}|$, requires the use of probability vectors $\hat{\pi}$. In [35], the reachability states are stored using MDD (Multi-valued Decision Diagrams) while matrix diagrams are used to store the Kronecker representation. The numerical results show that solving the model using a technique such as Gauss–Seidel requires less iterations and less time per iteration than the shuffle algorithm. However, the matrix diagram solution requires twice the memory size required by the basic Kronecker representation. Alternative approaches have been developed [37, 46, 47]. In these approaches, the reachable state space S is first computed and the model is solved using the reduced probability vectors π. The algorithm proposed in [37, 46] is based on a permutation which reorders the states according to their reachability, and the use of π.

Recently, a new version of the shuffle algorithm called *FR-Sh* (*Fully Reduced Shuffle*) has been proposed in [27, 28]. This algorithm, which uses the probability vectors π, improves the memory needs and the computation time when there are a lot of unreachable states. It has been proved that the algorithm allows an important reduction in the memory requirements, in particular when using iterative methods such as Arnoldi and GMRES.

In [48], iterative methods based on splittings, such as Jacobi and Gauss–Seidel, are proved to be better than the Power method.

4.6 Outlook

Currently there is a great need for a comparison between the different algorithms. In [35], a comparison between matrix diagrams and the original shuffle algorithm showed a substantial advantage of the matrix diagrams in terms of computation time. Several versions of the shuffle algorithms have been investigated in [27, 28], among which the *PR-Sh* (*Partially Reduced Shuffle*) and the *FR-Sh* algorithms. These new versions, in particular the *FR-Sh* algorithm, improve considerably the original shuffle algorithm. These results will be fully validated if the *FR-Sh* algorithm, for example, is compared to the alternative approaches in the literature. Moreover, it will be interesting to consider in the future a combination of the shuffle algorithms and elaborated data structures such as decision diagrams [28]. Such approaches may allow the analysis of larger systems.

5 Symbolic Data Structures in CTMCs and SPAs

This section summarises the state-of-the-art of symbolic approaches to state space representation. In this context, the term "symbolic" – which was originally coined in the context of model checking [49] – refers to the use of decision diagrams as a graph-based data structure for compactly encoding sets of states or transition systems of various kinds. This approach has the potential to handle very large models efficiently while utilising only small amounts of memory.

5.1 Introduction to BDDs

A Binary Decision Diagram (BDD) [50] is a symbolic representation of a Boolean function $f : \mathbb{B}^n \mapsto \mathbb{B}$. Its graphical interpretation is a rooted directed acyclic graph with one or two terminal vertices, marked 1 and 0 (for "true" and "false"). Each non-terminal vertex x is associated with a Boolean variable $\mathsf{var}(x)$ and has two successor vertices, denoted by $\mathsf{then}(x)$ and $\mathsf{else}(x)$. The graph is ordered in the sense that on each path from the root to a terminal vertex, the variables are visited in the same order. A reduced BDD is essentially a collapsed binary decision tree in which isomorphic subtrees are merged and "don't care" vertices are skipped (a vertex is called "don't care" if the truth value of the corresponding variable is irrelevant for the truth value of the overall function). Reduced ordered BDDs are known to be a canonical representation of Boolean functions.

As a simple example, Fig. 7 (a) shows the full binary decision tree for the function $(\bar{\mathsf{a}} \wedge \mathsf{t}) \vee (\mathsf{a} \wedge \mathsf{s} \wedge \bar{\mathsf{t}})$, where all vertices drawn at one level are labelled by the same Boolean variable, as indicated at the left of the graph. The edge from vertex x to $\mathsf{then}(x)$ represents the case where $\mathsf{var}(x)$ is true; conversely, the edge from x to $\mathsf{else}(x)$ the case where $\mathsf{var}(x)$ is false. (In the graphical representation, then-edges are drawn solid, else-edges dashed.) Part (b) of the figure shows the

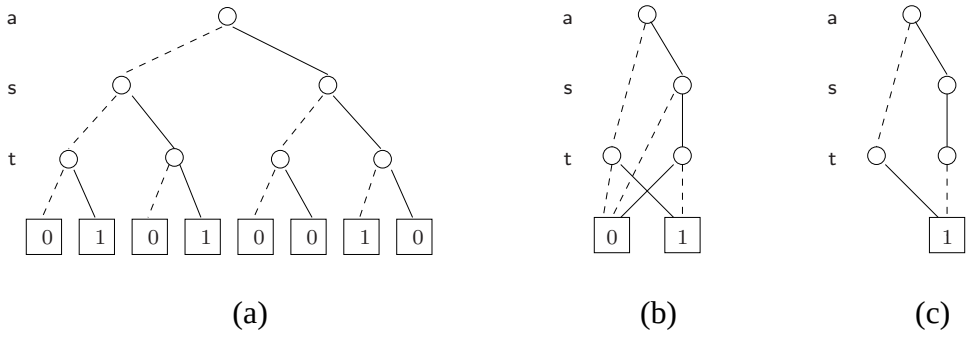


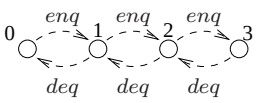
Fig. 7. (a) Binary decision tree, (b) reduced BDD and (c) simplified graphical representation for the Boolean function $(\bar{\mathsf{a}} \wedge \mathsf{t}) \vee (\mathsf{a} \wedge \mathsf{s} \wedge \bar{\mathsf{t}})$

corresponding reduced BDD which can be obtained from the decision tree by merging isomorphic subgraphs and leaving out don't care vertices. For instance, in the diagrams shown in Fig. 7, if $\mathsf{a} = 0$ then s is a don't care variable. As shown in Fig. 7 (c), in the graphical representation of a BDD, for reasons of simplicity, the terminal vertex 0 and its adjacent edges are usually omitted. In all three graphs shown in Fig. 7, the function value for a given truth assignment can be determined by following the corresponding edges from the root until a terminal vertex is reached.

A finite set, e.g. the reachability set of a state-based model, can be represented by a BDD via its characteristic functions, i.e. a function yielding one or zero, depending on whether the corresponding state – encoded as a bitstring – is in the set or not. Similarly, a finite transition system can be represented by a BDD, as illustrated by the following example:

Example: Fig. 8 (left) shows the labelled transition system (LTS) of a simple finite-buffer queueing process. The middle of the figure shows the way transitions are encoded, and the resulting BDD is depicted on the right. Action labels *enq* and *deq* are encoded with the help of two Boolean variables[8] a_1 and a_2. In particular, the encodings of action *enq* resp. *deq* is set to (0,1) resp. (1,0). The LTS has four states, therefore two bits are needed to represent the state number. Note that this BDD uses a special "interleaved" ordering of the Boolean variables encoding the source and target state. This interleaved ordering is a proven heuristics for obtaining small BDD sizes in the context of compositional model construction [51].

[8] Since there are only two distinct actions in the LTS, one bit would be enough to encode the action. However, the encoding 0 is often reserved for the special internal action τ, and in any case it is not mandatory to use the smallest possible number of bits.



	$(\mathsf{a}_1, \mathsf{a}_2, \mathsf{s}_1, \mathsf{t}_1, \mathsf{s}_2, \mathsf{t}_2)$
$0 \xrightarrow{enq} 1$	$(0,1,0,0,0,1)$
$1 \xrightarrow{enq} 2$	$(0,1,0,1,1,0)$
$2 \xrightarrow{enq} 3$	$(0,1,1,1,0,1)$
$1 \xrightarrow{deq} 0$	$(1,0,0,0,1,0)$
$2 \xrightarrow{deq} 1$	$(1,0,1,0,0,1)$
$3 \xrightarrow{deq} 2$	$(1,0,1,1,1,0)$

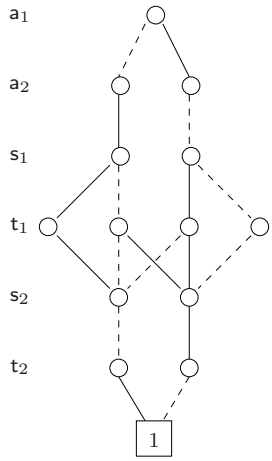


Fig. 8. Queue LTS, transition encoding and corresponding BDD

5.2 Related decision diagram data structures

Over the years, several variants of the basic BDD data structure have been developed, mostly initiated by the wish to find a data structure perfectly suited to a particular verification or analysis problem. In this sections, the most prominent ones are discussed briefly.

A Multi-terminal BDD (MTBDD) is a symbolic representation of a pseudo-Boolean function $f : \mathbb{B}^n \mapsto \mathbb{D}$, where $\mathbb{D}$ is an arbitrary domain [52]. MTBDDs are constructed similarly to BDDs, but – as the name implies – there may be more than two terminal vertices carrying the function values. If one wishes to encode labelled Markov chains symbolically, MTBDDs can be used, where the transition rate of each encoded transition is stored in the corresponding terminal vertex of the MTBDD.

Example: As an example, consider the queueing process from Fig. 8, now with arrival rate λ and service rate μ, as depicted in Fig. 9 (left). Its MTBDD representation is shown in Fig. 9 (right). The set of paths leading to a non-zero vertex is of course the same as in Fig. 8, but the graph now has two branches, one for the *enq*- and one for the *deq*- transitions. We now consider a scaling of this model: The queueing system shown in Fig. 8 has a capacity of 3 customers. It can be generalised to an M/M/1 queue with capacity $c = 2^k - 1$, which means that the labelled Markov chain has 2^k states. One can show that the MTBDD representation of this Markov chain only requires $10k - 2$ MTBDD vertices (by the same argument as the one used in [53]), which means that for a family of models whose state space grows exponentially, MTBDDs provide a representation which only grows linearly! This nice result is of course related to the perfect

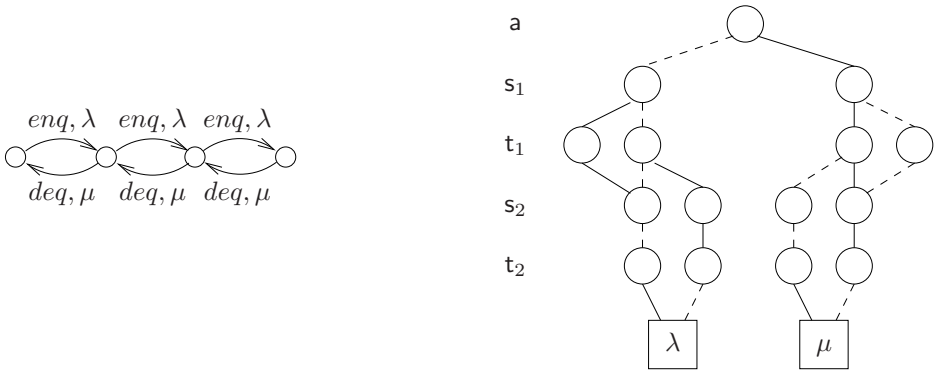


Fig. 9. Labelled Markov chain and corresponding MTBDD

regularity of the M/M/1 model (in particular if the state space is a power of 2), but many case studies have demonstrated the space efficiency of MTBDD representations for a large class of models, especially if used in a compositional context (see section 5.3 below).

An orthogonal strand of research has led to the class of zero-suppressed binary decision diagrams (ZBDD). It is based on the observation that some Boolean functions whose set of minterms contains many negated variables do not have very compact BDD representations. However, if one changes the reduction rules for the decision diagram, more compact representations can be obtained. So, instead of eliminating don't care vertices (as in BDDs), in ZBDDs those vertices are eliminated whose then-successor is the terminal 0-vertex [54]. In other words, if a variable level is skipped from the root to the terminal 1-vertex, this means that the corresponding variable carries the value 0 (in a BDD setting this situation would mean that the corresponding variable is don't care). The use of ZBDDs and their multi-valued variants has been shown to be beneficial for the analysis of Markov reward models [55]. A recent overview of zero-suppressed decision diagrams can be found in [56].

Working with decision diagrams, the branching decision taken at every vertex of the graph does not necessarily have to be a binary decision. This observation led to a large class of multi-valued (or multiway) decision diagrams (MDD) [57], originally employed for logic synthesis and verification. Multiway decision diagrams are also very well suited to encode the reachability set of a decomposed Petri net (i.e. a Petri net whose set of places is partitioned into subsets). The local marking of a subnet is hereby encoded as an integer, and there is a one-to-one correspondence between the subnets and the levels of the decision diagram. The maximum branching factor at a certain level is thus given by the number of reachable markings of that subnet [58].

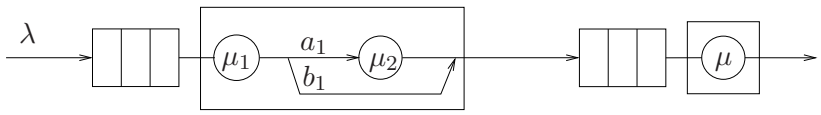


Fig. 10. A tandem queueing network

5.3 Model generation and manipulation

Starting from a high-level model description, such as a Generalized Stochastic Petri Net (GSPN) or a stochastic process algebra model, efficient procedures are needed for generating the symbolic representation of the underlying labelled Markov chain. For Markovian stochastic process algebra, symbolic semantics have been developed which map a given process algebraic model directly to its underlying MTBDD-representation, without generating an intermediate labelled transition system in explicit form [59]. The key point of this mapping consists of the exploitation of the compositional structure of the SPA model at hand: Given M_1 and M_2, the MTBDD representations of two SPA processes P_1 and P_2, the MTBDD representation of their parallel composition $P_1|[S]|P_2$ is obtained as

$$\begin{aligned} \mathsf{M} = \quad & (\mathsf{M}_1 \cdot \mathsf{S}) \cdot (\mathsf{M}_2 \cdot \mathsf{S}) \\ & + \mathsf{M}_1 \cdot (1 - \mathsf{S}) \cdot \mathsf{Id}_2 \\ & + \mathsf{M}_2 \cdot (1 - \mathsf{S}) \cdot \mathsf{Id}_1 \end{aligned} \tag{2}$$

where S is the BDD-encoding of the synchronisation set S, and the Id_i are BDDs denoting stability of processes P_i. This construction guarantees that the size of the resulting MTBDD M is linear in the sizes of the operand MTBDDs and the number of action labels, which is a major source for the compactness of the symbolic representation [60].

Example: As an extension of our previous example, consider the tandem queueing network shown in Fig. 10, where an upstream queue with Coxian service is connected to a downstream queue with exponential service (the downstream queue is actually the one already considered in Sec. 5.2). Each of the queues has finite capacity of $c = 2^k - 1$, yielding a scalable model with altogether $2^k \cdot 2^k \cdot 2 = 2^{2k+1}$ states (the last factor of 2 is due to the two Coxian phases). Fig. 11, cited from [60], shows the growth of this model and of the associated MTBDDs. The last two columns give the numbers of MTBDD vertices for two variants of the symbolic representation. The numbers in column "monolithic" were obtained by directly encoding the labelled Markov chain of the overall model. Clearly, this approach does not lead to compact MTBDDs. In contrast, the numbers in column "compositional" were obtained by constructing the overall MTBDD in a compositional fashion, which means that two MTBDDs (one for the Coxian queue and one for the Markovian queue) were composed according to (2). This yields MTBDDs which grow only linearly with the parameter k, although the state space grows exponentially!

k	c	reachable states	transitions	MTBDD size	
				monolithic	compositional
3	7	128	378	723	148
4	15	512	1,650	1,575	197
7	127	32,768	113,538	11,480	341
10	1,023	2,097,152	7.3308e+06	–	485
14	16,383	5.36871e+08	1.8789e+09	–	677

Fig. 11. Statistics for the tandem queueing network

It is important to note that the above construction (2) yields an encoding of the so-called *potential* transition system which may also include transitions emanating from non-reachable states of the product state space. Symbolic reachability algorithms are employed to determine the set of reachable states. A mapping similar to (2) from the high-level model description to the symbolic representation of the underlying labelled Markov chain is employed in the probabilistic model checker PRISM [61], where users specify their models with the help of a guarded command language, based on Reactive Modules, which also features synchronisation between modules. In PRISM, not only CTMCs but also DTMCs and Markov decision processes can be specified, all of which are internally represented using MTBDDs.

In contrast to these structure-oriented approaches, the activity-local state graph generation scheme [62] does not need any a priori structure information and is therefore applicable to a general class of Markovian models. It creates its own, very fine structure, by considering the local effect of every activity (i.e. event) within the model. Since it is a round-based scheme, where reachability analysis needs to be performed in every round, an efficient variant of symbolic reachability analysis was developed as part of this approach. The activity-local approach is implemented (using zero-suppressed multi-terminal BDDs) in the framework of the Moebius modelling environment [63].

The saturation algorithm, first described in [64] also uses a fixpoint iteration scheme. Several variants of it have been described in the literature which work on different types of decomposition of the high-level model, see e.g. [65]. As the underlying data structures, these algorithms use extensible versions of multi-valued decision diagrams and matrix diagrams [66, 67].

For models with both Markovian and immediate transitions, elimination of the so-called vanishing states is a prerequisite for numerical analysis. An efficient symbolic elimination algorithm, which is implemented in the tool CASPA, has been described in [68]. It consists three steps: 1. Some precomputations (including a realisation of the maximal progress assumption, i.e. the priority of an immediate transition over any timed transition). 2. The main fully symbolic round-based elimination algorithm. 3. A semi-symbolic post-processing for elim-

inating transitions that form an immediate loop or cycle (of which there are usually very few).

Since numerical analysis of large models is expensive (in terms of processing time and memory, see Sec. 5.4), it is desirable to reduce the size of the state space, if anyway possible. Bisimulation minimisation is a fundamental concept on which such a reduction can be based, whereby exact performance and dependability measures of the modelled system are preserved (in the context of Markov chains, bisimulation is known as lumpability). An early approach to symbolic bisimulation minimisation was described in [69], and more recently some very efficient symbolic bisimulation algorithms have been developed [70, 71]. All of these algorithms follow the basic principle of partition-refinement, where initially all states are considered equivalent, and at every step the current state space partitioning is refined according to some lumpability criterion, until stability is reached. However, the representation techniques used by these algorithms for encoding the state space partitions as BDDs are very different, resulting in a runtime-memory tradeoff between the different algorithms.

5.4 Numerical analysis based on the structure of the decision diagram

For computing the desired performance or dependability measures of the modelled system, numerical analysis of the underlying stochastic process needs to be performed. In the case of CTMCs, this means that the vector of stationary probabilities has to be computed, which is typically done using iterative numerical methods such as Jacobi or Gauss-Seidel or their overrelaxed variants (or the well-known uniformization algorithm in case of transient state probabilities). In principle, such numerical calculations – which involve matrix-vector calculations as their basic operations – could work exclusively on symbolic data structures [72]. However, storing vectors of state probabilities symbolically (e.g. as an MTBDD) has proved to be neither memory-efficient nor time-efficient. Therefore, a hybrid scheme was developed [73], where only the matrix of transition rates is stored symbolically as an MTBDD while the vector of probabilities (of only the reachable states) is stored in explicit form as an array. Even with this approach, the probability vector (and not the storage of the matrix) is still the memory-bottleneck for large models! For speeding up the traversal of the MTBDD (which is done for looking up the transition rates), parts of it are sometimes replaced by sparse matrix data structures, which yields a typical time-space tradeoff. Parallel versions of symbolic numerical algorithms have also been developd [74]. In [75], a symbolic version of the multilevel algorithm (a recursive aggregation/disaggregation scheme) was described. Since the MTBDD possesses a recursive block-structure (due to the nature of its composition) this type of algorithm matches very well with the structure of the MTBDD. The multilevel scheme has also been combined with sparse representations of both terminal and intermediate blocks of the matrix, and some further accelerations of the calcula-

tions have been developed [76]. Numerical solution algorithms based on different types of matrix-diagrams have been implemented in the tool SMART [66, 67], as well as approximate algorithms for stationary solution [77].

5.5 Outlook

The "symbolic" approach described in this section is now a mature method implemented in several successful tools (e.g. PRISM [61], SMART [67] and CASPA [68]). It supports all phases of modelling, from state space generation to various forms of (qualitative and quantitative) analysis. Decision-diagram-based techniques are capable of dealing with very large state spaces, thus alleviating the problem of state space explosion. They are therefore among the methods of choice for constructing and analysing detailed and scalable resilience models. However, there are still many remaining research problems, for instance the question of how to further improve the numerical analysis of very large models with the help of approximations or bounding methods.

6 Mean-Field Approximation

Mean-field methods were first used in Physics to describe the interaction of particles in systems like plasma or dense gases. Instead of providing a detailed model, the influence of the mean environment on a single particle is studied. Subsequently mean-field methods were introduced to many other topics, for an overview see the introduction of [78].

The idea of aggregating the influence of the environment can also help in dealing with the state space explosion problem. We consider large networks of identical components, for example, computers in the Internet running the same piece of protocol software. Modelling each component and its interaction with the other components explicitly results in an intractable state space. Instead we focus on an approximating model where only the average impact of the complete system on the evolution of a component is considered. Results for this mean-field approximation model are cheap to compute (matrix-vector multiplications). It allows for statements about the average behaviour of the underlying original model, especially when the number of components is large.

6.1 Computing the Mean-Field

In the following we describe the process of mean-field approximation for discrete-time Markov models. We then illustrate this process with an example.

Discrete-time model for single component At the beginning of the process we have to determine a discrete-time probabilistic model for a single component. It will typically have a relatively small set of states S. The transition probabilities are allowed to depend on N, the total number of components in the system, and on the so-called *occupancy measure* $\mathbf{m}$, a vector containing the fraction of components in each state. The model is then determined by a local probability matrix $P^N(\mathbf{m})$.

The underlying stochastic process Even though we are never going to explicitly construct it, we have to consider some properties of the stochastic process for the complete system. It consists of the parallel composition of the models for all N components. If the model for a single component has K states, the state space for the composed system would have K^N states. Since all components behave identically, we can aggregate the state space to the occupancy measure where we only keep track of the fraction of components in each state. The state space still would consist of $\binom{K+N-1}{K-1}$ states. The mean-field method gives us an approximation for the *transient occupancy measure*, that is, the occupancy measure at a given point in time t.

The deterministic limit process Under certain convergence requirements [79] the local probability matrix has a limit if N goes to infinity:

$$P(\mathbf{m}) = \lim_{N \to \infty} P^N(\mathbf{m})$$

For a given initial occupancy measure $\mu(0)$, the matrix $P(\mathbf{m})$ defines a *deterministic* process

$$\mu(t+1) = \mu(t) \cdot P(\mu(t))$$

This deterministic process approximates the occupancy measure for large N. The values of $\mu(t)$ for $t \in \mathbb{N}$ are easily determined by simple matrix-vector multiplications. Note that the matrix $P(\mu(t))$ has to be recalculated in each step.

Interpretation of results The mean-field method gives us a deterministic approximation of the occupancy measure. For each point in time it predicts a distribution of components over the different possible states. If we evaluated the underlying stochastic process we would get a different type of result: it would result in a distribution over all possible occupancy measures, that is, a distribution of possible distributions. However, the larger N is, the more deterministic the occupancy measure becomes (Central Limit Theorem). It is therefore justified to use the deterministic mean-field approximation of the number of components N is large. Even for small N the approximation gives valuable insight into the average behaviour of the system.

6.2 Illustrating Example

We consider a very simple information dissemination protocol. A large number of network nodes communicates in order to distribute a small but important piece of information. The communication protocol is completely decentralised which makes it robust even if some of the involved network nodes fail. To increase security, a node only accepts and redistributes the information after it has received it twice. This prevents the distribution of incorrect data, resulting from transmission errors or malicious insertion. Figure 12 shows the state-transition diagram for nodes. Each node can be in one of three states: it can be ignorant (state 0), it can be waiting for confirmation (state 1), or it can be knowing (state 2). A node can move from ignorant to waiting state with a certain probability.

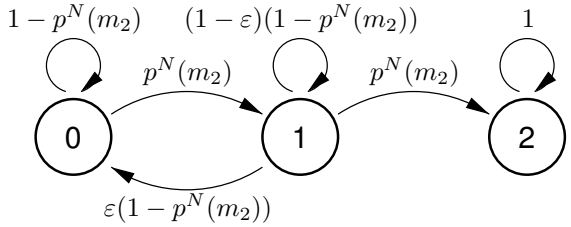


Fig. 12. Probabilistic model representing a single network node

From here it either moves on to knowing, or, if it does not receive the piece of information a second time, it returns to ignorant. Once knowing, it will never get ignorant again — it never forgets.

The probabilities in the model depend on two parameters that can change over the evolution of the system: firstly, it depends on the total number N of nodes in the network, secondly, it depends on m_2, the fraction of already knowing nodes. Additionally there is a *gossip* parameter $g \in [0, 1]$ that reflects the probability that an informed node is going to pass on the information, and a parameter $\varepsilon \in [0, 1]$ that governs the return to state 0 from state 1.

The probability for a single ignorant node to move from state 0 to state 1 then is

$$p^N(m_2) = g \cdot \frac{m_2 \cdot N}{N - 1}$$

which is proportional to the fraction of knowing nodes among the remaining nodes, and g. The higher the fraction of already knowing nodes is in the network, the higher is the probability, that a yet ignorant node gets the information.

A waiting node (state 1) moves to the knowing state again with probability $p^N(m_2)$. It returns to state 0 with probability $\varepsilon(1 - p^N(m_2)$. The parameter ε can be interpreted as the probability for discarding the first receipt of the data.

To represent all transition probabilities in the model, we state a probability matrix that is *local* to each node. It depends on N and the *occupancy measure* $\mathbf{m} = (m_0, m_1, m_2)$.

$$P^N(\mathbf{m}) = \begin{pmatrix} 1 - p^N(m_2) & p^N(m_2) & 0 \\ \varepsilon(1 - p^N(m_2)) & (1-\varepsilon)(1 - p^N(m_2)) & p^N(m_2) \\ 0 & 0 & 1 \end{pmatrix}$$

Considering all N network nodes in parallel would result in a state space with 3^N states. Even if we only recorded the occupancy measure, the state space would still have $\frac{N(N+1)(N+2)}{2}$ states – for three-state components!

Two avoid this blowup, we consider the mean-field limit for the occupancy measure. The limit of the local probability matrix is

$$P(\mathbf{m}) = \lim_{N\to\infty} P^N(\mathbf{m}) = \begin{pmatrix} 1 - g \cdot m_2 & g \cdot m_2 & 0 \\ \varepsilon(1 - g \cdot m_2) & (1-\varepsilon)(1 - g \cdot m_2) & g \cdot m_2 \\ 0 & 0 & 1 \end{pmatrix}$$

For a given *initial* occupancy measure $\mu(0)$ we can then approximate the transient evolution of the occupancy measure by the deterministic process [80, 79]

$$\mu(t+1) = \mu(t) \cdot P(\mu(t))$$

Figure 6.2 shows the deterministic occupancy measure $\mu(t)$ for $t \in [0, 400]$ when at the beginning the fraction of ignorant nodes is 99% and the fraction of knowing nodes is 1%, that is, $\mu(0) = (0.99, 0, 0.01)$. The figure represents the behaviour that could be expected: the fraction of ignorant nodes decreases continually while the number of knowing nodes increases. Since all nodes have to move through the intermediate waiting state, the fraction of nodes in this state first increases and then decreases again.

The mean-field approximation does not give us information about the possible deviations from the computed deterministic value. However, the results depicted in Figure 6.2 allow a statement about the approximate time at which we expect all nodes to be knowing with high probability.

6.3 Outlook

Very often we encounter real-world systems that consist of a large number of identical replicas of the same component. In this section we have shown how the mean-field approximation method can be employed if the components are represented by discrete-time models. The core computation for transient measures then boils down to a series of cheap matrix-vector multiplications. Mean-field analysis is not restricted to discrete-time models. It can also be applied to continuous-time Markov chains [81, 79], to Markov decision processes [82], and probably to many other probabilistic or stochastic modelling classes.

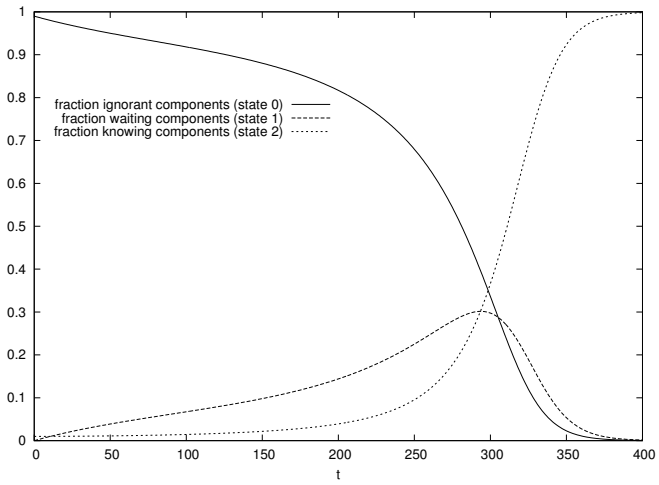


Fig. 13. Mean-field approximation for the illustrating example

7 Fluid analysis in CTMCs and SPAs

Representing the explicit state space of performance models has inherent difficulties. Just as the state-space explosion effects functional correctness evaluation, so it can also be easily a problem in performance models. In particular, classical Markov chain analysis of any variety requires exploration of the global state space and, even for a simple system, this quickly becomes computationally infeasible. One technique that attempts to side-step the state-space explosion is so-called *fluid analysis*.

In the discrete-time world of performance modelling, such techniques have already been used by Benaïm and Le Boudec [83] to good effect in *mean-field* analysis of performance models. Similarly, Bakhshi *et al.* [80, 78] have developed some discrete-time model-specific analysis techniques for gossip protocols using the mean-field technique.

In the field of stochastic process algebras, Hillston developed *fluid-flow* analysis [84] to make first-order approximations of massively parallel PEPA models. Bortolussi [85] presented a formulation for the stochastic constraint programming language, sCCP. Cardelli has a first-order fluid analysis translation to ordinary differential equations (ODEs) for the π-calculus [86]. Petriu and Woodside [87] have successfully used Mean Value Analysis of hierarchical queueing models such as Layered Queueing Networks (LQNs) to obtain mean response time results over large component-based systems.

In this section, we briefly outline a fluid analysis technique that is applied to a variety of the PEPA language known as Grouped PEPA (or GPEPA) [88]. It has the advantage over the original fluid approximation for PEPA or π-calculus of

having higher moment results available. These can be used to calculate variance information and give a notion of accuracy of the first-order prediction. Additionally, higher moments can also be used to create bounding approximations on some varieties of passage-time distribution [89].

7.1 GPEPA

Grouped PEPA (GPEPA) [88] is a simple syntactic extension to PEPA which allows the straightforward identification of models that can be analysed using fluid techniques. Specifically, the component grouping in GPEPA identifies the abstraction level at which the fluid analysis will take place. By adding component group labels to these structures, we also benefit from being able to identify uniquely components in particular states in particular parts of the model structure.

For a detailed summary of PEPA process algebra syntax and operational meaning we refer to the reader to the many papers that have been published discussing and using PEPA. The following is a small selection of such work [90–93].

A GPEPA model is formed by composing multiple labelled component groups together. The grammar for a GPEPA model G is:

$$G ::= G \underset{L}{\bowtie} G \;\mid\; Y\{D\} \tag{3}$$

where Y is a group label, unique to each component group. The term $G \underset{L}{\bowtie} G$ represents synchronisation over action types in the set L, where $L \subseteq \mathcal{A}$ is a set of possible action types in the model.

In this context, a component group is a parallel cooperation of a normally large number of *fluid* components. A fluid component is one whose state changes can be captured by a random variable and ultimately a set of differential equations. Parallel, in this setting, means that there is no synchronisation between individual members of the component group. A component group D is specified as follows:

$$D ::= D \parallel D \;\mid\; P \tag{4}$$

where P is a *fluid component*, a standard PEPA process algebra term. The combinator $\parallel$ represents parallel, unsynchronised cooperation between fluid components.

7.2 A Client–Server Example of a Fluid model

To illustrate more clearly how component groups and fluid components are used together to construct GPEPA models, we consider a GPEPA model of a simple client/server system from [89]. The type of model we wish to consider in this

section is one which exhibits massive parallelism. We present a simple system with n clients and m servers. The system uses a 2-stage fetch mechanism: a client requests data from the pool of servers; one of the servers receives the request, another server may then fetch the data for the client. At any stage, a server in the pool may fail. Clients may also timeout when waiting for data after their initial request. We could capture this scenario of n clients cooperating on the *request* and *data* actions with m resources with the following GPEPA system equation:

$$\boldsymbol{CS}(n, m) \stackrel{def}{=} \mathbf{Clients}\{\boldsymbol{Client}[n]\} \underset{L}{\bowtie} \mathbf{Servers}\{\boldsymbol{Server}[m]\} \tag{5}$$

where $L = \{request, data\}$ and $C[n]$ represents n parallel cooperating copies of component C. Each client is represented as a ***Client*** component and each server as a ***Server*** component. Each client operates forever in a loop, completing three tasks in sequence: *request*, *data* and then *think*; and they may also perform a *timeout* action when waiting for data:

$$\begin{aligned}
\boldsymbol{Client} &\stackrel{def}{=} (request, r_r).\boldsymbol{Client_waiting} \\
\boldsymbol{Client_waiting} &\stackrel{def}{=} (data, r_d).\boldsymbol{Client_think} + (timeout, r_{tmt}).\boldsymbol{Client} \\
\boldsymbol{Client_think} &\stackrel{def}{=} (think, r_t).\boldsymbol{Client}
\end{aligned}$$

The servers on the other hand first complete a *request* action followed by a *data* action in cooperation with the clients but at either stage they may perform a *break* action and enter a broken state in which a *reset* action is required before the server can be used again:

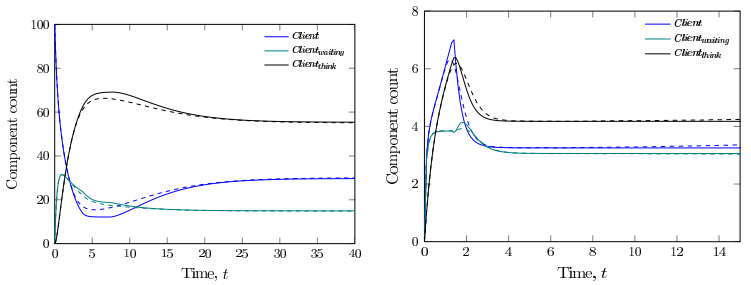
$$\begin{aligned}
\boldsymbol{Server} &\stackrel{def}{=} (request, r_r).\boldsymbol{Server_get} + (break, r_b).\boldsymbol{Server_broken} \\
\boldsymbol{Server_get} &\stackrel{def}{=} (data, r_d).\boldsymbol{Server} + (break, r_b).\boldsymbol{Server_broken} \\
\boldsymbol{Server_broken} &\stackrel{def}{=} (reset, r_{rst}).\boldsymbol{Server}
\end{aligned}$$

The *request* and *data* actions are shared actions between the clients and servers in order to model the fact that clients must perform these actions by interacting with a server. The actions *timeout*, *think*, *break* and *reset* on the other hand are completed independently.

7.3 Generating ODEs for Fluid analysis

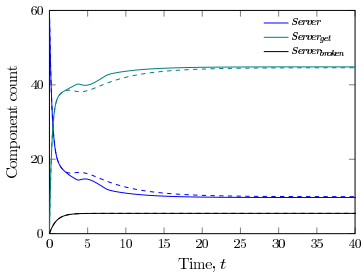
Fluid analysis is used to approximate the mean dynamics of a GPEPA model. That is, at a time t, fluid analysis will tell you how many components in each named component group there are on average. As the size of the component groups increases, so the approximation can be shown to improve [89]. This convergence can also be demonstrated for higher moments of component counts [94].

In this section, we analyse the client/server model of Section 7.2 to show mean component counts for particular client and server states, Figures 14(a) and 14(c).

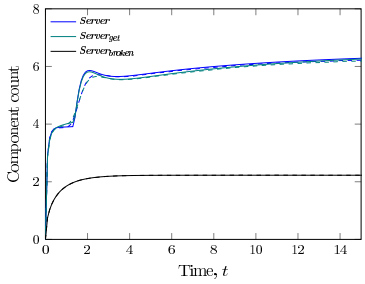


(a) Mean number of Clients

(b) Standard deviation of number of Clients



(c) Mean number of Servers



(d) Standard deviation of number of Servers

Fig. 14. Fluid analysis of the number of components in a Client/Server model. Simulations are given as dashed plots.

We show the standard deviation of the component counts for the client/server model in Figures 14(b) and 14(d). Where there is a local deviation from the simulated results, this is due to the synchronisation model used in PEPA rather than fluid analysis in particular [88, 94].

We will skip any further preamble and write down the generating equation for a set of ODEs that gives a first-order fluid analysis of a GPEPA model. The higher order fluid analysis (that gives quantities can such as variance, standard deviation and skewness) can be found in [88].

Let G be a GPEPA model. We define the evolution of the number of components of type P in group H over time, by $v_{H,P}(t)$. This quantity is defined by a system of first-order coupled ODEs:

$$\dot{v}_{H,P}(t) = \sum_{\alpha \in \mathcal{A}} \underbrace{\left(\sum_{Q \in \mathcal{B}(G,H)} p_\alpha(Q,P)\, \mathcal{R}_\alpha(G, V(t), H, Q) \right)}_{\text{incoming rate}} - \underbrace{\mathcal{R}_\alpha(G, V(t), H, P)}_{\text{exit rate}} \tag{6}$$

for all component group/component state pairs (H, P) in the GPEPA model G. That is, we generate one ODE for each local state in the compositional model. Hence we avoid any association with the global state space.

This can broadly be explained by slitting Equation (6) into two parts, the *incoming rate* and the *exit rate*. The incoming rate part describes the total rate of the model that increases the number of components of type P in group H for a given action α. This is calculated from the product of two quantities:

1. $\mathcal{R}_\alpha(G, V(t), H, Q)$, the overall rate that component type Q enables doing actions of type α, where Q is also in group H, and;
2. $p_\alpha(Q, P)$, the probability that Q can evolve to become component type P doing an action α (it could perform an α and lead to another component type).

This is summed over all possible component types Q in group H to give the incoming rate to P for an action α.

The exit rate from P in group H for an action α is simply $\mathcal{R}_\alpha(G, V(t), H, P)$, overall rate that component type P enables doing actions of type α. The full definition of the component rate function $\mathcal{R}_\alpha(\cdot)$ is given in [88].

Finally, the aggregate rate of change of components of type of P in group H can be shown to equal to the total incoming rate minus the total exit rate, summed over all possible action types α.

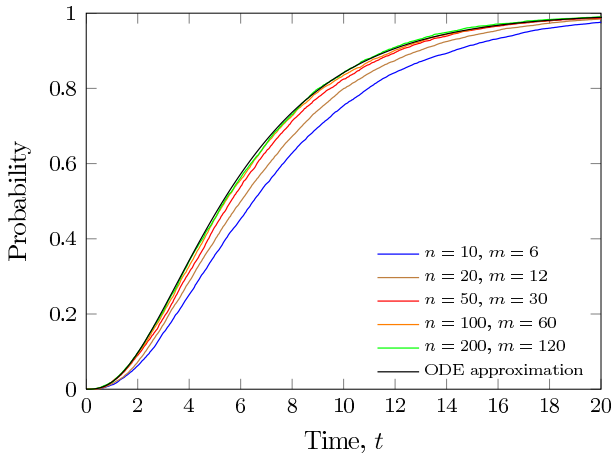


Fig. 15. Fluid passage time analysis of a Client response time in the Client/Server model of Section 7.2

We have shown how first order fluid analysis of large stochastic processes can be used to create passage and response time distributions and related measures [95,

89]. This is done by performing the sort of fluid analysis described here on an appropriately modified model. Figure 15 shows a single Client response time distribution (from steady-state) given all the synchronisations in the system. As with all fluid analysis, as the model gets larger, the distribution predicted by the fluid analysis (black line) becomes more accurate.

7.4 Outlook

Further developments in fluid analysis have shown how other types of passage time distribution can be extracted from massively parallel systems [95]. Additionally, real-valued rewards can be accumulated over the lifetime of a process model to capture quantities such as cost and energy consumption. For smaller scale models, these can be analysed using discrete-state Markov Reward Models [96]; fluid analysis can be used here also to analyse larger scale reward models [97] which would otherwise have a prohibitive state space size.

8 Conclusion

In this chapter we have presented a number of efficient representation and analysis techniques that can aid resilience and performance analysis of large and complex systems. It is worth saying that there is no one best technique in all cases. As with many modelling and analysis problems, the best tool for the job will often depend on the problem being tackled and the type of answer being sought. Some of the techniques discussed can be used to tackle many types of performance and resilience problem, while some are better suited to a particular type of analysis. Some of the techniques will allow the user to recover an explicit state representation, while others consider many model states in aggregation.

For a particular problem a modeller will wish to select those techniques that can produce the desired analysis. However, beyond that, it will be worth the modeller's while to attempt a few of the remaining techniques to see which scales best to their needs. They will find dramatic differences in operation and those differences (in execution time and memory usage) will reflect many issues, such as how the model is originally represented, how many and indeed whether parameters are available to the modeller and the level of complexity that the modeller wishes to capture in their model.

Finally, we would like to emphasise that these descriptions are by necessity brief summaries of much more involved techniques. These topic introductions should be treated as such and many references have been provided for the reader to follow up if more information is required.

Acknowledgements

Jeremy Bradley, Richard Hayden and Nigel Thomas are supported by the UK Engineering and Physical Sciences Research Council on the AMPS project (reference EP/G011737/1). Leïla Kloul is supported by the European Celtic project HOMESNET [98].

References

1. W. J. Stewart, *Probability, Markov Chains, Queues and Simulation. The Mathematical Basis of Performance Modeling.* Princeton University Press, 2009.
2. Various authors, "The Network Simulator ns-2." `http://www.isi.edu/nsnam/ns/`. (last seen May 11, 2010).
3. A. Varga, "The OMNeT++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM'2001)*, June 2001.
4. P. Reinecke and K. Wolter, "Libphprng – A library to generate PH-distributed random numbers," tech. rep., Freie Universität Berlin, 2011. (to appear).
5. K. Wolter, P. Reinecke, and A. Mittermaier, "Evaluation and Improvement of IEEE 1588 Frequency Synchronisation through Detailed Modelling and Simulation of Backhaul Networks." (submitted for publication), 2011.
6. A. Horváth and M. Telek, "PhFit: A general phase-type fitting tool," in *TOOLS '02: Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, (London, UK), pp. 82–91, Springer-Verlag, 2002.
7. J. Jackson, "Networks of waiting lines," *Operations Research*, vol. 5, pp. 518–521, Aug. 1957.
8. F. Baskett, M. Chandy, R. Muntz, and F. Palacios, "Open, closed and mixed networks of queues with different classes of customers," *Journal of the ACM*, vol. 22, pp. 248–260, 1975.
9. W. Gordon and G. Newell, "Closed queueing systems with exponential servers," *Operations Research*, vol. 15, no. 2, pp. 254–265, 1967.
10. E. Gelenbe, "Product-form queueing networks with negative and positive customers," *Journal of Applied Probability*, vol. 28, no. 3, pp. 656–663, 1991.
11. G. Balbo, S. Bruell, and M. Sereno, "Embedded processes in generalized stochastic Petri nets," in *Proc. 9th Intl. Workshop on Petri Nets and Performance Models*, pp. 71–80, 2001.
12. R. Boucherie, "A characterisation of independence for competing Markov chains with applications to stochastic Petri nets," *IEEE Trans. on Software Eng.*, vol. 20, no. 7, pp. 536–544, 1994.
13. W. Henderson and P. Taylor, "Embedded processes in stochastic Petri nets," *IEEE Trans. on Software Eng.*, vol. 17, no. 2, pp. 108–116, 1991.
14. J. Hillston, "Exploiting structure in solution: Decomposing compositional models," in *Lectures on Formal Methods and Performance Analysis* (E. Brinksma *et al*, ed.), vol. 2090 of *Lecture Notes in Computer Science*, pp. 278–314, Springer-Verlag, 2001.
15. P. G. Harrison, "Turning back time in Markovian process algebra," *Theoretical Computer Science*, vol. 290, pp. 1947–1986, January 2003.
16. P. G. Harrison, "Compositional reversed Markov processes, with applications to G-networks," *Performance Evaluation*, vol. 57, pp. 379–408, June 2004.

17. M. Reiser and S. Lavenberg, "Mean value analysis of closed multichain queueing networks," *JACM*, vol. 22, no. 4, pp. 313–322, 1980.
18. K. Sevcik and I. Mitrani, "The distribution of queueing network states at input and output instants," *JACM*, vol. 28, no. 2, pp. 358–371, 1981.
19. S. Lavenberg and M. Reiser, "Stationary state space probabilities at arrival instants for closed queueing networks with multiple types of customers," *Journal of Applied Probability*, vol. 17, no. 4, pp. 1048–1061, 1980.
20. N. Thomas and Y. Zhao, "Mean value analysis for a class of PEPA models," *The Computer Journal*, 2011. `10.1093/comjnl/bxq064`. accepted for publication.
21. K. Trivedi and M. Malhotra, *Reliability and Performability Techniques and Tools: A Survey*. In B. Walke and O. Spaniol, editors, Springer Verlag, Aachen, September 1993.
22. P. Courtois, *Decomposability, queueing and computer system applications*. ACM monograph series, 1977.
23. B. Plateau, *De l'Evaluation du Parrallélisme et de la Synchronisation*. PhD thesis, PhD Thesis, November 1984.
24. B. Plateau, "On the stochastic structure of parallelism and synchronisation models for distributed algorithms," in *Proc. ACM Sigmetrics Conference on Measurement and Modelling of Computer Systems*, 1985.
25. B. Plateau and J. Fourneau, "A methodology for solving markov models of parallel systems," *Journal of Parallel and Distributed Computing*, August 1991.
26. P. Fernandes, B. Plateau, and W. Stewart, "Efficient Descriptor-Vector Multiplications in Stochastic Automata Networks," *JACM*, vol. 3, no. 45, pp. 381–414, 1998.
27. A. Benoit, *Méthodes et Algorithmes pour l'évaluation des performances des systèmes informatiques à grands espaces d'états*. PhD thesis, PhD thesis, 2003.
28. A. Benoit, B. Plateau, and W. Stewart, "Memory-efficient algorithms with applications to the modelling of parallel systems," *Future Generation Computer Systems*, vol. 22, pp. 838–847, 2006.
29. B. Plateau, J. Fourneau, and K. Lee, "PEPS: A Package for Solving Complex Markov Models of Parallel Systems," in *Proc. of the 4th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 1988.
30. I. Cidon and I. Gopal., "PARIS: An Appoach to Integrated High-Speed Private Networks," *International Journal on Digital and Analog Cabled Systems*, vol. 1, pp. 77–86, April-June 1988.
31. L. Kloul, *Méthodes d'évaluation des performances pour les réseaux ATM*. PhD thesis, Université de Versailles-St-Quentin-en-Yvelines, January 1996.
32. S. Donatelli, "Superposed Generalised Stochastic Petri Nets: Definition and Efficient Solution," in *Proc. of 15th Int. Conf. on Application and Theory of Petri Nets* (M. Silva, ed.), 1994.
33. P. Buchholz and P. Kemper, "Numerical analysis of stochastic marked graphs," in *Proc. of Int. Workshop on Petri Nets and Performance Models*, (Durham, NC), pp. 32–41, IEEE-Computer Society Press, oct 1995.
34. J. Campos, S. Donatelli, and M. Silva, "Structured solution of stochastic dssp systems," in *Proc. of Int. Workshop on Petri Nets and Performance Models*, (St Malo, France), pp. 91–100, IEEE-Computer Society Press, jun 1997.
35. G. Ciardo and A. Miner, "A data structure for the efficient kronecker solution of gspns," in *In P. Buchholz editor, Proc. of the 8th International Workshop on Petri Nets and Performance Models (PNPM'99)*, (Saragoza, Spain), pp. 22–31, 1999.

36. S. Donatelli, "Superposed Generalised Stochastic Petri Nets: Definition and Efficient Solution," in *Proc. of 15th Int. Conf. on Application and Theory of Petri Nets* (M. Silva, ed.), 1994.
37. P. Kemper, "Numerical Analysis of Superposed GSPNs," *IEEE Transactions on Software Engineering*, vol. 22, no. 9, pp. 615–628, September 1996.
38. C. Ciardo and M.Tilgner, "On the use of Kronecker operators for the solution of generalized stochastic Petri nets," Tech. Rep. 96-35, Institute for Computer Applications in Science and Engineering, Hampton, VA, May 1996.
39. P. Buchholz, "Hierarchical Structuring of Superposed GSPNs," *IEEE Transactions on Software Engineering*, vol. 25, no. 2, pp. 166–181, March/April 1999.
40. S. Donatelli and P. Kemper, "Integrating synchronization with priority into a kronecker representation," *Performance evaluation*, vol. 44, no. 1–4, pp. 73–96, 2001.
41. P. Buchholz, "Compositional Analysis of a Markovian Process Algebra," in *Proc. of 2nd Process Algebra and Performance Modelling Workshop* (U. Herzog and M. Rettelbach, eds.), 1994.
42. J. Hillston and L. Kloul, "An efficient kronecker representation for pepa models," in *Proceedings of the Joint International Workshop, PAPM-PROBMIV 2001*, vol. 2165 of *LNCS*, (Aachen, Germany), pp. 120–135, Springer, 2001.
43. J. Hillston and L. Kloul., "Formal Techniques for Performance Analysis: blending SAN and PEPA," *Formal Aspects of Computing*, vol. 19, pp. 3–33, 2007.
44. P. Fernandes, *Méthodes numériques pour la solution de systèmes Markoviens à grand espace d'états*. PhD thesis, PhD thesis, 1998.
45. W. Stewart, K. Atif, and B. Plateau, "The numerical solution of stochastic automata networks," *European Journal of Operationnal Research*, vol. 86, pp. 503–525, 1995.
46. P. Kemper, "Reachability analysis based on structured representations," in *Proc. of 17th International Conference on Application and Theory of Petri nets*, pp. 269–288, LNCS,Springer Verlag, 1996.
47. S. D. P. Bucholtz, G. Ciardo and P. Kemper, "Complexity of memory-efficient kronecker operations with applications to the solution of markov models," *INFORMS Journal on Computing*, vol. 3, no. 12, pp. 203–222, 2000.
48. T. Dayar, "Iterative methods based on splittings for stochastic automata networks," *European Journal of Operationnal Research*, vol. 110, pp. 166–186, 1998.
49. K. McMillan, *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
50. R. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. C-35, pp. 677–691, August 1986.
51. R. Enders, T. Filkorn, and D. Taubner, "Generating BDDs for symbolic model checking in CCS," *Distributed Computing*, no. 6, pp. 155–164, 1993.
52. E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao, "Multi-terminal Binary Decision Diagrams: An efficient data structure for matrix representation," in *IWLS: International Workshop on Logic Synthesis*, (Tahoe City), May 1993.
53. H. Hermanns, J. Meyer-Kayser, and M. Siegle, "Multi Terminal Binary Decision Diagrams to Represent and Analyse Continuous Time Markov Chains," in *3rd Int. Workshop on the Numerical Solution of Markov Chains* (B. Plateau, W. Stewart, and M. Silva, eds.), pp. 188–207, Prensas Universitarias de Zaragoza, 1999.
54. S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," in *30th ACM/IEEE Design Automation Conference*, pp. 272–277, 1993.
55. K. Lampka, *A Symbolic Approach to the State Graph Based Analysis of High-Level Markov Reward Models*. PhD thesis, University of Erlangen-Nürnberg, Technische Fakultät, 2007.

56. K. Lampka, M. Siegle, J. Ossowski, and C. Baier, "Partially-shared zero-suppressed multi-terminal BDDs: concept, algorithms and applications," *Formal Methods in System Design*, vol. 36, no. 3, pp. 198–222, 2010.
57. T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli, "Multi-valued decision diagrams: theory and applications," *Multiple-Valued Logic*, vol. 4, no. 1-2, pp. 9–62, 1998.
58. A. Miner and G. Ciardo, "Efficient reachability set generation and storage using decision diagrams," in *Application and Theory of Petri Nets 1999* (H. Kleijn and S. Donatelli, eds.), (Williamsburg, VA, USA), pp. 6–25, Springer, LNCS 1639, 1999.
59. M. Kuntz and M. Siegle, "Deriving symbolic representations from stochastic process algebras," in *Process Algebra and Probabilistic Methods, Proc. PAPM-PROBMIV'02*, pp. 188–206, Springer, LNCS 2399, 2002.
60. H. Hermanns, M. Kwiatkowska, G. Norman, D. Parker, and M. Siegle, "On the use of MTBDDs for performability analysis and verification of stochastic systems," *Journal of Logic and Algebraic Programming*, vol. 56, no. 1-2, pp. 23–67, 2003.
61. M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic model checking for performance and reliability analysis," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 40–45, 2009.
62. K. Lampka and M. Siegle, "Activity-local Symbolic State Graph Generation for High-Level Stochastic Models," in *Proc. of 13th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB)*, (Nürnberg, Germany), pp. 245–263, VDE Verlag, 2006.
63. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. Doyle, W. Sanders, and P. Webster, "The Moebius Framework and its Implementation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 956–969, 2002.
64. G. Ciardo, G. Luettgen, and R. Siminiceanu, "Saturation: An efficient strategy for symbolic state-space generation," in *Proc. TACAS'01*, (Genova, Italy), pp. 328–342, Springer, LNCS 2031, 2001.
65. M. Wan and G. Ciardo, "Symbolic state-space generation of asynchronous systems using extensible decision diagrams," in *Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM '09, pp. 582–594, Springer-Verlag, 2009.
66. G. Ciardo and A. Miner, "A data structure for the efficient Kronecker solution of GSPNs," in *PNPM'99* (P. Buchholz and M. Silva, eds.), pp. 22–31, IEEE computer society, 1999.
67. A. Miner, "Efficient solution of GSPNs using canonical matrix diagrams," in *Petri Nets and Performance models (PNPM'01)* (R. German and B. Haverkort, eds.), pp. 101–110, IEEE Computer Society Press, 2001.
68. J. Bachmann, M. Riedl, J. Schuster, and M. Siegle, "An Efficient Symbolic Elimination Algorithm for the Stochastic Process Algebra Tool CASPA," in *35th Int. Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM'09)*, pp. 485–496, Springer LNCS 5404, 2009.
69. H. Hermanns and M. Siegle, "Bisimulation Algorithms for Stochastic Process Algebras and their BDD-based Implementation," in *ARTS'99, 5th Int. AMAST Workshop on Real-Time and Probabilistic Systems* (J.-P. Katoen, ed.), pp. 144–264, Springer, LNCS 1601, 1999.
70. S. Derisavi, "A Symbolic Algorithm for Optimal Markov Chain Lumping," in *TACAS 2007* (O. Grumberg and M. Huth, eds.), pp. 139–154, Springer, LNCS 4424, 2007.

71. R. Wimmer, S. Derisavi, and H. Hermanns, "Symbolic partition refinement with automatic balancing of time and space," *Performance Evaluation*, vol. 67, no. 9, pp. 815–835, 2010.
72. M. Fujita, P. McGeer, and J.-Y. Yang, "Multi-terminal Binary Decision Diagrams: An efficient data structure for matrix representation," *Formal Methods in System Design*, vol. 10, pp. 149–169, April/May 1997.
73. D. Parker, *Implementation of symbolic model checking for probabilistic systems*. PhD thesis, School of Computer Science, Faculty of Science, University of Birmingham, 2002.
74. M. Kwiatkowska, D. Parker, Y. Zhang, and R. Mehmood, "Dual-Processor Parallelisation of Symbolic Probabilistic Model Checking," in *Proc. 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'04)* (D. DeGroot and P. Harrison, eds.), pp. 123–130, IEEE Computer Society Press, 2004.
75. J. Schuster and M. Siegle, "A symbolic multilevel method with sparse submatrix representation for memory-speed tradeoff," in *14. GI/ITG Conf. Measurement, Modelling and Evaluation of Computer and Communication Systems (MMB08)*, pp. 191–205, VDE Verlag, 2008.
76. J. Schuster and M. Siegle, "Speeding up the Symbolic Multilevel Algorithm," in *6th Int Workshop on the Numerical Solution of Markov Chains (NSMC2010)*, pp. 79–82, 2010.
77. A. Miner, G. Ciardo, and S. Donatelli, "Using the exact state space of a Markov model to compute approximate stationary measures," *Performance Evaluation Review*, vol. 28, pp. 207–216, June 2000. Proc. of ACM SIGMETRICS 2000.
78. R. Bakhshi, L. Cloth, W. Fokkink, and B. Haverkort, "Mean-field framework for performance evaluation of push-pull gossip protocols," *Performance Evaluation*, vol. 68, pp. 157–179, February 2011.
79. J.-Y. L. Boudec, D. McDonald, and J. Mundinger, "A generic mean field convergence result for systems of interacting objects," in *Proc. 4th Conf. on the Quantitative Evaluation of Systems (QEST'07)*, pp. 3–18, IEEE Computer Society, 2007.
80. R. Bakhshi, L. Cloth, W. Fokkink, and B. Haverkort, "Mean-field analysis for the evaluation of gossip protocols," in *Proc. 6th International Conference on the Quantitative Evaluation of Systems (QEST'09)*, pp. 247–256, IEEE Computer Society, 2009.
81. A. Bobbio, M. Gribaudo, and M. Telek, "Analysis of large scale interacting systems by mean field method," in *Proc. 5th Conf. on the Quantitative Evaluation of Systems (QEST'08)*, pp. 215–224, IEEE Computer Society, 2008.
82. N. Gast, B. Gaujal, and J.-Y. Le Boudec, "Mean field for Markov decision processes: from discrete to continuous optimization," tech. rep., arXiv:1004.2342v2, 2010.
83. M. Benaïm and J.-Y. Le Boudec, "A class of mean field interaction models for computer and communication systems," *Performance Evaluation*, vol. 65, no. 11-12, pp. 823–838, 2008.
84. J. Hillston, "Fluid flow approximation of PEPA models," in *QEST'05, Proceedings of the 2nd International Conference on Quantitative Evaluation of Systems*, (Torino), pp. 33–42, IEEE Computer Society Press, September 2005.
85. L. Bortolussi and A. Policriti, "Stochastic concurrent constraint programming and differential equations," in *QAPL'07, 5th Workshop on Quantitative Aspects of Programming Languages*, vol. 190 of *Electronic Notes in Theoretical Computer Science*, pp. 27–42, September 2007.

86. L. Cardelli, "On process rate semantics," *Theoretical Computer Science*, vol. 391, pp. 190–215, February 2008.
87. D. C. Petriu and C. M. Woodside, "Approximate MVA for software client/server models by Markov chain task-directed aggregation," in *3rd IEEE Symposium on Parallel and Distributed Processing*, pp. 322–329, Dec. 1991.
88. R. Hayden and J. T. Bradley, "A fluid analysis framework for a Markovian process algebra," *Theoretical Computer Science*, vol. 411, pp. 2260–2297, May 2010. doi://10.1016/j.tcs.2010.02.001.
89. R. Hayden, *Scalable Performance Analysis of Massively Parallel Stochastic Systems*. PhD Thesis, Department of Computing, Imperial College London, 2011.
90. J. Hillston, *A Compositional Approach to Performance Modelling*, vol. 12 of *Distinguished Dissertations in Computer Science*. Cambridge University Press, 1996.
91. G. Clark, S. Gilmore, J. Hillston, and N. Thomas, "Experiences with the PEPA performance modelling tools," in *UKPEW'98, Proceedings of the 14th UK Performance Engineering Workshop*, Edinburgh, July 1998.
92. J.-M. Fourneau, L. Kloul, and F. Valois, "Performance modelling of hierarchical cellular networks using PEPA," *Performance Evaluation*, vol. 50, pp. 83–99, November 2002.
93. S. Gilmore, J. Hillston, and M. Ribaudo, "An efficient algorithm for aggregating PEPA models," *IEEE Transactions on Software Engineering*, vol. 27, no. 5, pp. 449–464, 2001.
94. A. Stefanek, R. A. Hayden, and J. T. Bradley, "A new tool for the performance analysis of massively parallel computer systems," in *QAPL'10, 8th Workshop on Quantitative Aspects of Programming Languages*, vol. 28 of *Electronic Proceedings of Theoretical Computer Science*, pp. 159–181, 2010.
95. R. Hayden, A. Stefanek, and J. T. Bradley, "Fluid computation of passage time distributions in large Markov models," tech. rep., Dept. of Computing, Imperial College London, November 2010. http://pubs.doc.ic.ac.uk/fluid-passage-time/ (Under review).
96. M. Telek and S. Rácz, "Numerical analysis of large Markovian reward models," *Performance Evaluation*, vol. 36–37, pp. 95–114, Aug. 1999.
97. A. Stefanek, R. Hayden, and J. T. Bradley, "Fluid analysis of energy consumption using rewards in massively parallel Markov models," in *ICPE 2011, 2nd ACM/SPEC International Conference on Performance Engineering, March 14-16, 2011, Karlsruhe, Germany*, pp. 121–131, March 2011.
98. "HOMESNET: Home base station, an emerging network paradigm. CELTIC european project." Available at http://www.celtic-initiative.org/Projects/Celtic-projects/Call6/HOMESNET/homesnet-default.asp, 2009–2011.