Rare event simulation for highly dependable systems with fast repairs

Rare Event Simulation for Highly Dependable Systems with Fast Repairs[1]

Daniël Reijsbergen[a,2], Pieter-Tjerk de Boer[a], Werner Scheinhardt[a], Boudewijn Haverkort[a,b]

[a] *Center for Telematics & Information Technology, University of Twente, Enschede, The Netherlands*
[b] *Embedded Systems Institute, Eindhoven, The Netherlands*

Abstract

Probabilistic model checking has been used recently to assess, among others, dependability measures for a variety of systems. However, the employed numerical methods, such as those supported by model checking tools such as PRISM and MRMC, suffer from the state-space explosion problem. The main alternative is statistical model checking, which uses standard Monte Carlo simulation, but this performs poorly when small probabilities need to be estimated. Therefore, we propose a method based on importance sampling to speed up the simulation process in cases where the failure probabilities are small due to the high speed of the system's repair units. This setting arises naturally in Markovian models of highly dependable systems. We show that our method compares favourably to standard simulation, to existing importance sampling techniques and to the numerical techniques of PRISM.

Keywords: Statistical model checking, Rare events, Importance sampling, Dependable systems.

1. Introduction

The goal of probabilistic model checking is to quantitatively evaluate the validity of performance and dependability properties of stochastic systems. After the system has been modelled as a Markov chain, or specified in terms of a higher-level language such as AADL [23], properties of interest are specified using the logic pCTL [11] or CSL [1]. Then, a model checker is invoked to determine in which states of the Markov chain these properties are satisfied.

Two main approaches to probabilistic model checking have emerged in recent years. In the first (numerical) approach one generates the state space of the Markov model beforehand and then numerically determines in which states the specified pCTL or CSL formula holds [1, 2]. In the second (statistical) approach, the behaviour of the system over time is repeatedly

[1] This paper is based on earlier work; see [20].

[2] Corresponding author
E-mail addresses: {d.p.reijsbergen, p.t.deboer, w.r.w.scheinhardt, b.r.h.m.haverkort}@utwente.nl

simulated in order to draw a conclusion about whether the property is satisfied in a certain state at a given level of confidence [24, 25].

Both of these approaches can experience problems when the probabilities of interest become small. For estimating probabilities using simulation it is a well-known rule of thumb that for a rare event probability p, $100/p$ simulation runs are needed to obtain a reasonable estimate [5]. In modern dependable (embedded) computer and communications systems, interesting probabilities of the order of magnitude of 10^{-8} are not uncommon, and methods to speed up the simulation process receive an increasing amount of attention.

Importance sampling [13] is a sophisticated form of simulation that uses information about the system model to speed up the simulation process. If done correctly, this can lead to large increases in the efficiency. The price that we pay for such better estimates is the loss of generality. Any stochastic system can be simulated naively, but an importance sampling approach that works in one setting will typically fail to perform well in other settings. For example, a system consisting of components that are prone to failure can be highly dependable because the individual component failure rates are low or because the repair rates are high, yet the technique from [22] (called Balanced Failure Biasing) was proven to work well only in the former setting.

As a consequence, we need to restrict ourselves to certain types of *models* and rare events in this paper, and specify how to extract from these models the *information* that we need for efficient simulation. The *models* considered here describe systems consisting of parallel component types as will be explained in detail in Section 2. We are mainly interested in the case where the repair rates are high, as this is a common situation in practical model checking problems for which existing importance sampling approaches have no fully satisfactory answer, but we will also consider the case where the failure rates of components are low. We do not need to impose that the component failure and repair rates remain constant when one or several components have already failed.

We extract the relevant *information* about the system model from the high-level description using a *path-based approach* (see Section 4.2). The approach is similar to the one used in [6], but is more general: the methods of [6] work well in the low failure rate setting but did not work well in the high repair rate setting, and are in that sense similar to Balanced Failure Biasing. Our method works well in both settings. Furthermore, as we restrict ourselves to a certain class of models, the information about these paths is immediately clear, while [6] needed to run a path-finding algorithm similar to Dijkstra's method before simulating.

In addition, although we will initially impose that repair starts immediately when a component fails, we will show in Section 6 how one can drop this assumption. In large-scale storage or computing systems, multiple hot spares are commonly used to achieve redundancy and strategies involving deferred repair are then attractive. In Section 6.4, we discuss the even wider applicability of our method by studying an extension of model of Section 2 in which repairs for all component types are handled by a single repairman who abides by a first-come first-serve repair policy.

The properties of interest are expressed using CSL, as we are interested in the continuous time behaviour of these systems. We will consider both the (transient) unreliability and the (steady-state) unavailability, also to be described in Section 2. Admittedly, this is still

considerably different from being able to evaluate whether an *arbitrary* CSL-formula holds. However, we view our current method as a first step towards more general probabilistic model checking procedures.

The rest of this paper is structured as follows. In Section 2 we introduce the distributed database system that we will use as a case study, specify probabilities of interest and explain how to estimate those probabilities using simulation. In Section 3 we describe importance sampling in the general setting. In Section 4 we introduce our approach and analyse its theoretical strengths and limitations. In Section 5 we evaluate our technique empirically and compare it to standard statistical model checking, to another, very general importance sampling scheme and to the numerical techniques of the model checking tool PRISM. In Section 6 we generalise the results of Section 5 to models in which repair is postponed until multiple components have failed, and discuss the possibility of further generalisation. Section 7 concludes the paper.

2. Model & Preliminaries

As said in the introduction, importance sampling methods use information about the way rare events occur in the model to speed up the simulation. Because this information depends heavily on the model, we must first specify what type of models we will consider. In this section we will first describe our case study and use it to specify what kind of models our method can handle. We will then specify what probabilities we need to estimate and how to do this using standard simulation, as we will need the ideas behind it for our discussion of importance sampling in Section 3. In Section 6 we will discuss two generalisations of the benchmark model and how they impact the efficiency of the simulation method.

2.1. Distributed Database System

The distributed database system is a benchmark problem in the field of dependability evaluation [21]. It was recently studied in [4], and a variant was studied in [8]. It can be seen as part of a more general class of systems consisting of parallel component types. In Section 2.1.1 we will describe the states and transition structure of the system, in Section 2.1.2 we will describe the failure conditions (and, by implication, the labelling of the states) and in Section 2.1.3 we discuss the precise benchmark setting and an extension that we use in the results section.

2.1.1. Model Description

The distributed database system consists of 24 disks that are grouped together in 6 clusters of 4 disks, 4 disk controller units divided into two sets that each access three disk clusters and a processor that accesses the disk controllers. The processor has a spare that takes over in case of failure. There is one repair facility for each of the six disk clusters, one for each of the two sets of disk controllers and one for the processor and its spare. The system is depicted in Figure 1.

We can distinguish 9 component *types*. Types $i = 1, \ldots, 6$ represent the disks in cluster i, types 7 and 8 represent the disk controllers in sets 1 and 2 respectively and type 9 represents

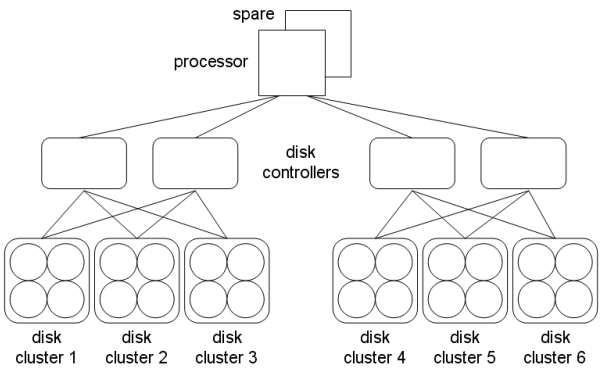

Figure 1: A distributed database system

the processors. The interfailure times and repair durations are assumed to be exponentially distributed. Let $\vec{x}$ be a vector in $\mathbb{N}^9$ in which each element x_i denotes how many components of type i have failed. We call this vector the *state* of the process. The system will be assumed to start in an initial state $\vec{x}_0$ at time $t_0 = 0$.

Let $\mathcal{D} = \{1, \ldots, 9\}$ be the set of component types. The failure and repair rates of components of these types may depend on the current state, so let the failure rates be some nonnegative function $\lambda_i(\vec{x})$ for each component type $i \in \mathcal{D}$ and $\vec{x} \in \mathbb{N}^9$. Let the repair rates similarly be given by nonnegative functions $\mu_i(\vec{x})$. The failures and repairs are called *transitions*. The repair rate of component type i can only be positive when there is at least one failed component of type i, and the failure rate can only be positive if there are components of type i left that are operational. We assume that the system as a whole is fault-tolerant, and that the probability of system failure is low either because of the component failure rates being low or because of the repair rates being high.

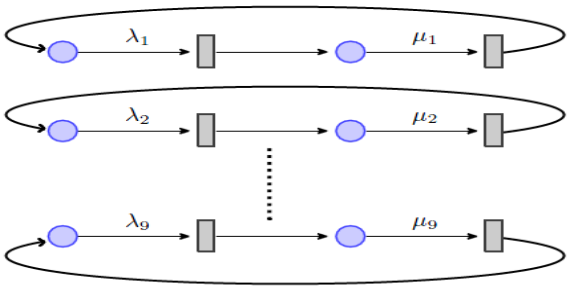

Figure 2: The distributed database system modelled as a stochastic Petri net. The transition intensities are marking-dependent functions.

We also note that the system can be modelled as a stochastic Petri net (SPN; see, e.g., [3]). In Figure 2 the system of the case study is depicted as an SPN. Each state is then

a *marking*, the component types are then *places* and the number of *tokens* in place i then represents the number of failed components of type i.

2.1.2. Operation and Failure

The system is said to be operational if a processor can access all the data in the disks — this condition is satisfied if each of the following subconditions holds: (1) at least one processor is up, (2) at least one disk controller in each of the controller clusters is up, and (3) at least three disks are up in each of the six disk clusters. In general, the method that we introduce in this paper works when system failures occur if for at least one component type i, a specified number n_i of components has failed. System failure in the benchmark setting falls into this category.

Using this definition of system failure we can formalise what kind of measures we will estimate. The *unreliability* is the probability that the system stops being operational at some point before a specified time bound τ. The *unavailability* is the steady-state probability that at some time point t in steady-state the system is not operational. Unreliability properties can be expressed using CSL as $\mathcal{P}_{\bowtie p}(\Diamond^{<\tau} \mathit{fail})$ and steady-state unavailability properties as $\mathcal{S}_{\bowtie p}(\mathit{fail})$, where *fail* is an atomic property that is assigned to all states which represent system failure as defined earlier.

2.1.3. The Benchmark Case and Generalisations

The failure and repair rates of the individual components are given in Table 1. In the

unit	failure rate	repair rate
disks	λ	μ
disk controllers	3λ	μ
processors	3λ	μ

Table 1: Failure and repair rates in the benchmark model

benchmark case (see [4]) we have $\lambda = 1/6000$ and $\mu = 1$. The rates in the literature are per hour, and the time bound τ for the unreliability is 5 weeks, so equal to 840 in this setting. The individual components all have the same failure distribution regardless of how many other components are up. E.g., the total failure rate $\lambda_i(\cdot)$ of type 1 components (the first disk cluster) is 4λ when no components are down, 3λ when one component has failed, and so on. The component repair rate of each i is always μ if $x_i > 0$, because there is only repair facility per type.

We also parameterise the number of spares. We introduce a new parameter n and assume there are n processors, n disk controllers per set and $2n$ disks per cluster. For $n = 2$ we are back in the benchmark case. Let failure in this more general setting be defined to occur when either (1) no processor is up, (2) in one disk controller set, no disk controller is up or (3) in one disk cluster at least n disks are down.

Note that even in this general setting, the state space is to a strong degree lumpable, meaning that the size of the state space can be reduced to facilitate the use of the numerical techniques implemented in tools such as PRISM and Arcade. However, if we change the

single component failure rates such that for they are different for each component type then these techniques will no longer work. Ffor the simulation-based methods, on the other hand, it will not matter. Hence, for the empirical results of Section 5 we will not use state space reduction techniques.

2.2. Discrete Event Simulation

Now that we have modeled the system and specified probabilities that we want to estimate, we will discuss *how* these probabilities can be estimated. The standard simulation-based approach is called *standard* discrete-event simulation or *Monte Carlo simulation*.

2.2.1. Path Generation

As mentioned in the introduction, we repeatedly simulate the behaviour of the system in order to come up with an estimate. The result of one simulation procedure is called a sample *run* or *path*. We define a *run* (timed path) as a (in our setting finite) series of states and transition times. By a (timeless) *path* we just refer to the series of states that we encountered. Let Ω be the set of all runs, then this is the sample space from which we randomly sample runs ω. We will not further delve into the measure theoretic background of Ω in this paper.

We generate samples from Ω as follows: we start the run at time $t_0 = 0$ in state x_0, which for the unreliability is given by the '*all up*' state $\vec{0}$. Then we consecutively determine which transition is taken and how long it takes until this transition is taken. This is done as follows: let, at step k, $\vec{x}_k$ be the state and let its *exit rate* $\eta(\vec{x}_k)$ be defined as

$$\eta(\vec{x}_k) = \sum_{j' \in \mathcal{D}} \Big(\lambda_{j'}(\vec{x}_k) + \mu_{j'}(\vec{x}_k) \Big) . \tag{1}$$

We pick the transition j of type i as the next transition to be taken with probability

$$p_{\vec{x}_k}(j) = \begin{cases} \dfrac{\lambda_i(\vec{x}_k)}{\eta(\vec{x}_k)}, & \text{if } j \text{ is a failure transition,} \\ \\ \dfrac{\mu_i(\vec{x}_k)}{\eta(\vec{x}_k)}, & \text{if } j \text{ is a repair transition.} \end{cases} \tag{2}$$

Also, let T_k be the time instance at which the k'th step is taken. Then we let the sojourn time $\Delta = T_{k+1} - T_k$ have probability density

$$f_{\vec{x}_k}(\delta) = \eta(\vec{x}_k) e^{-\eta(\vec{x}_k)\delta}. \tag{3}$$

We continue until we can terminate — a condition that depends on the property whose validity we seek to evaluate.

2.2.2. Estimating the unreliability

Let Φ^r be the event that the system hits a failure state before time τ, which we assume to be a model parameter given before the start of the simulation. Then the unreliability is given by $\pi = \mathbb{P}(\Phi^r) = \mathbb{E}(\mathbf{1}_{\Phi^r})$, where $\mathbf{1}_{\Phi^r}(\omega)$ denotes the indicator function which equals 1 if ω satisfies Φ^r and 0 otherwise. For each sample run we can evaluate whether Φ^r was satisfied on that run. So, after having sampled a series of runs $\{\omega_i, \ldots, \omega_N\}$ we can estimate $\mathbb{P}(\Phi^r)$ using

$$\hat{\pi} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\Phi^r}(\omega_i). \tag{4}$$

Let $\hat{\sigma}$ be the sample standard deviation of our series of runs. The 95%-confidence interval [3] for this estimate is then given by (see [16], page 254)

$$\left[\hat{\pi} - 1.96\frac{\hat{\sigma}}{\sqrt{N}} \quad , \quad \hat{\pi} + 1.96\frac{\hat{\sigma}}{\sqrt{N}} \right].$$

2.2.3. Estimating the unavailability

Estimating the steady-state unavailability using simulation is a little bit more tricky. To avoid having to '*warm-up*' the simulation before it reaches approximate equilibrium we apply a renewal argument. We partition the behaviour of the system as time progresses into disjoint *busy cycles*. A busy cycle starts and ends when we enter state $\vec{0}$. Let V be the steady-state unavailability, let Z be the amount of time during which the system is unavailable during a busy cycle and let D be the duration of a busy cycle. Then $\mathbb{E}(V) = \mathbb{E}(Z)/\mathbb{E}(D)$. The ratio estimator $\hat{v}$ is given by

$$\hat{v} = \frac{\hat{z}}{\hat{d}}, \tag{5}$$

where $\hat{z}$ and $\hat{d}$ are the Monte Carlo estimates for $\mathbb{E}(Z)$ and $\mathbb{E}(D)$ respectively. This estimator is biased, but strongly consistent (i.e., $\hat{v} \to \mathbb{E}(V)$ as $N \to \infty$; see [16], page 533). We generate different runs for the estimates $\hat{z}$ and $\hat{d}$ to avoid dependence. This becomes even more necessary when we use importance sampling because techniques that focus on rare events would lead to a large variance of $\hat{d}$ (more details will be given in Section 4.4). The 95%-confidence interval (see [16], pages 532–533) is then given by

$$\left[\hat{v} - 1.96\frac{\hat{\sigma}_v}{\hat{d}\sqrt{N}} \quad , \quad \hat{v} + 1.96\frac{\hat{\sigma}_v}{\hat{d}\sqrt{N}} \right],$$

[3] The confidence interval given here is based on the central limit theorem, which says that for N '*large enough*', the distribution of the sample average can be well approximated by a normal distribution. How large N really needs to be depends, in this case, on the true probability π. If π gets smaller, the approximation gets worse for fixed N. In tables such as Table 9, a typical Monte Carlo estimate will be 0, and a confidence interval based on the normal distribution will be the singleton set containing this point. Instead of giving a more accurate confidence interval based on the binomial distribution, we simply omit the result.

where $\hat{\sigma}_v^2 = \hat{\sigma}_z^2 + \hat{\sigma}_d^2\hat{v}^2$ and $\hat{\sigma}_z^2$ and $\hat{\sigma}_d^2$ are the sample variances of sequences containing the V's and D's respectively.

Although the above estimation procedures work in many cases, the downside is that when the probability that we need to estimate is small the number of runs N that we need in (4) or (5) is enormous. Finding a solution to this problem will be the focus of the next two sections.

3. Principles of Importance Sampling

In this section, we will only describe importance sampling for estimating the probability of failure before time τ, but something similar can be done for the steady-state unavailability. The problem with small probabilities is that the fraction of runs in which a rare event happens is very small. When we apply importance sampling, we carry out a similar stepwise procedure as in Section 2.2.1, but we use a different probability distribution in order to increase this fraction.

In this section, we will first describe importance sampling in Section 3.2 and then introduce the so-called zero-variance estimator on which we base our method in Section 3.3. Before we start with the formal definitions of the aforementioned concepts, we first give an intuitive description in Section 3.1.

3.1. Intuitive Description

Assume that we want to estimate some small probability w. Using standard simulation, we randomly draw zeroes and ones such that the fraction of ones is expected to be w (see (4)). Suppose we now somehow make the probability of drawing a non-zero *twice as large*. Then, if we multiply the value $\mathbf{1}_{\Phi^r}(\omega_i)$ of the ith run in (4) by $\frac{1}{2}$, we obtain an estimator that is unbiased and which has a *lower variance* than the standard estimator. Now suppose we already know w and make drawing a non-zero exactly w^{-1} times as likely. Hence, we draw a non-zero with probability one and multiply each $\mathbf{1}_{\Phi^r}(\omega_i)$ by the precise probability that we wish to estimate, resulting in an estimator with *zero variance*.

Unfortunately, the systems we study are far too complex to '*just*' multiply the probability of drawing a non-zero by some number and multiply by a constant weighting factor. There are too many ways in which the event of interest can occur. We will need to tweak the individual transition probabilities and sojourn time densities, and in order to obtain an efficient new distribution we need to know enough about our system. The basic way to do this in complex stochastic systems will be discussed below.

3.2. Basic Setup of Importance Sampling

Recall that we consider systems consisting of parallel component types. Assume that we can at least divide the transitions into two classes: repair transitions and failure transitions. Also assume that failure transitions have low rates. One could ask how this makes the probability of system failure small. A first answer could be that the low component failure rates cause the failure transitions to rarely '*win the race*' against the repair transitions.

So, assume that at some step of the simulation process we are in a state where at least one repair transition is enabled. The idea is now to use a new probability distribution p^*, which we call the *simulation distribution* (also known as a *change of measure*), for the simulation such that the component failure probabilities are much higher than under the old distribution (2). We compensate for this overestimation by weighting the estimate with the ratio of p and p^* — like the factor $\frac{1}{2}$ in the example in Section 3.1.

Every time a transition is sampled using the new density this weighting factor needs to be considered. The final weighting factor L of each run ω_i, called the *likelihood ratio*, is simply the *product* of the individual ratios $p_{\vec{x}_k}/p^*_{\vec{x}_k}$ in the run. Our new estimator then becomes

$$\hat{\pi} = \frac{1}{N}\sum_{i=1}^{N} L(\omega_i) \cdot \mathbf{1}_{\Phi^r}(\omega_i). \tag{6}$$

It is easy to prove that this estimator is unbiased for any new distribution that assigns positive probability to transitions that have positive probability under the old distribution.

We do not need to restrict ourselves to changing the *transition probabilities*. Note that the system failure probability can also be small because the time interval $[0, \tau]$ is too short for a sufficient number of component failures to occur. To remedy this, we can replace the *sojourn time density* f of (3) by a new density f^* with a higher transition rate. If we also account for the ratios f/f^* in the likelihood ratio L then our estimator remains unbiased and, if done correctly, has an even lower variance.

3.3. Zero Variance

Consider the following ideal situation: for every state $\vec{x}$ and time points $t \in [0, \tau]$ we already *know* the probability of system failure within $\tau - t$ time units. Call this probability $w(\vec{x}, t)$. Let $\chi(\vec{x}, j)$ be the new state that we obtain if transition j is chosen when we are in state $\vec{x}$, and $\mathcal{J}$ the set of all transition indices. Then we can introduce a new simultaneous density of the transition $j \in \mathcal{J}$ and sojourn time δ, namely

$$p^*_{\vec{x}}(j, \delta) = \frac{p_{\vec{x}}(j, \delta) \cdot w(\chi(\vec{x}, j), t + \delta)}{\int_0^{\tau - t} \sum_{j' \in \mathcal{J}} p_{\vec{x}}(j', \delta') \cdot w(\chi(\vec{x}, j'), t + \delta') d\delta'}, \tag{7}$$

where $p_{\vec{x}}(j, \delta) = p_{\vec{x}}(j) \cdot f_{\vec{x}}(\delta)$. This new simulation density was proven to yield an estimator with zero-variance in [6]. Of course, we do not explicitly know the function w, or else we would not need to simulate. However, we might be able to come up with an approximation $\hat{w}$ for w. Then, we replace the function w in (7) by this approximation. If the simulation distribution associated with the approximation $\hat{w}$ is good enough then we have succeeded in overcoming the main problem facing standard Monte Carlo simulation of rare events.

4. The New Simulation Distribution

The obvious next question is how to find a good way to find an approximation $\hat{w}$ that we can use to replace w in (7). In the following subsection, we will, as a first step, split the joint distribution of (7) into two distributions for the transitions and sojourn times respectively,

and explain how to draw sojourn times in an efficient manner. In the remaining subsections we will find better approximations for w step-by-step.

4.1. Drawing Sojourn Times

If w were known explicitly we would use (7) by first drawing a transition and then selecting a sojourn time conditional on this transition. The latter step, drawing sojourn times δ from (7), can be computationally expensive. Typically, the distribution function of the sojourn time conditioned on a transition j is not invertible, which would force us to resort to accept-reject schemes (see [12], chapter 18).

To avoid this, we apply our first simplification: we use the old density function of δ conditional on transition j to occur before time $\tau - t$. This gives us the following density:

$$f^*_{\vec{x}}(\delta) = \frac{\eta(\vec{x})e^{-\eta(\vec{x})\delta}}{1 - e^{-\eta(\vec{x})(\tau - t)}}.$$

The failure transitions are then drawn with probability

$$p^*_{\vec{x}}(j) = \frac{p_{\vec{x}}(j) \cdot \hat{w}(\chi(\vec{x}, j), t + \delta)}{\sum_{j' \in \mathcal{J}} p_{\vec{x}}(j') \cdot \hat{w}(\chi(\vec{x}, j'), t + \delta)} \tag{8}$$

for some approximation $\hat{w}$ yet to be determined (remember that $\mathcal{J}$ is the set of all transition indices). The technique of conditioning sojourn times on being smaller than $\tau - t$ is called *forcing* (see [18] or [19]). We do this for all transitions individually. One could also draw a whole series of transitions and then condition on the sum of their sojourn times being smaller than $\tau - t$, but then we would have to deal with general sums of exponentially distributed random variables and that is something we want to avoid, as the evaluation of distribution functions of these sums can quickly become computationally expensive.

4.2. Approximating w using Straight Paths

A way of approximating rare event probabilities is to consider only the paths that lead to system failure of components of a certain type i (that is, $x_i \geq n_i$) *without cycles*. We define a cycle as a sequence of states in which the first and last state are the same. It necessarily contains at least one failure and one repair transition.

Consider any state $\vec{x} \neq \vec{0}$ and a cycle starting in $\vec{x}$ of which the first transition is a component failure. For this cycle to occur, the failure transition must have occurred before a repair transition. When the component failure rates are made smaller or the repair rates are made higher, the occurrence of paths that contain these cycle becomes less likely. Accordingly, the straight paths — i.e., those without cycles — become a better approximation. This motivates the following, most basic, approximation for w.

Let a *straight path to failure* be a path that ends in a system failure state and which contains only failure transitions of a single component type. Let d be the number of types. From each state $\vec{x}$, we have d straight paths to failure, one for each type k. Let ν_k failures remain until the critical level n_k is reached, and let the vector of states that are seen along

this path be denoted by $[\vec{x}_{k,0}, \ldots, \vec{x}_{k,\nu_k}]$, with $\vec{x}_{k,0}$ equal to $\vec{x}$. The probability of this path being taken equals

$$p([\vec{x}_{k,0}, \ldots, \vec{x}_{k,\nu_k}]) = \prod_{i=0}^{\nu_k - 1} \frac{\lambda_j(\vec{x}_{k,i})}{\eta(\vec{x}_{k,i})},$$

where $\eta(\vec{x})$ is defined as in (1). We can then use

$$\hat{w}^*(\chi(\vec{x}, j), t_i) = \hat{w}^*(\chi(\vec{x}, j)) = \sum_{k=1}^{d} \prod_{i=0}^{\nu_k - 1} \frac{\lambda_j(\vec{x}_{k,i})}{\eta(\vec{x}_{k,i})} \tag{9}$$

as a time-independent approximation of w. From now on, the $*$ in $\hat{w}^*$ indicates that we only use the straight paths as an approximation.

4.3. Probability Contribution of Paths with Cycles

Unfortunately, the approximation (9) is too crude. One shortcoming of $\hat{w}^*$ is that the most likely path from a state $\vec{x}'$ to system failure might not be one of the d straight paths. In many cases, the most likely path is the path in which the system first returns to state $\vec{0}$ and then takes one of the straight paths that determine $\hat{w}^*(\vec{0})$. This can be seen in Figure 3, which depicts the state space of a simplified model with only two component types. Starting from state $\vec{x}'$ in Figure 3, the dashed line path (the one going '*north*') is, when the rates are realistic, much less likely to occur than the solid line path because the former contains one more step where a failure transition needs to win the race from a repair transition.

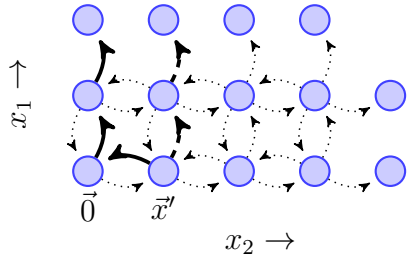

Figure 3: A model consisting of two types of components. For one type there are many more spares than the other, but its components fail more quickly.

Accordingly, we also consider the straight paths from state $\vec{0}$ for our approximation $\hat{w}(\vec{x}')$. From state $\vec{x}'$, the system returns to state $\vec{0}$ with probability almost equal to one. Therefore, for each state $\vec{x}$ we can use the sum of $\hat{w}^*(\vec{x})$ and $\hat{w}^*(\vec{0})$ instead of just the former. As a consequence, the jump from state $\vec{0}$ to state $\vec{x}'$ will more often be taken under the new distribution. This is desirable — paths that contain cycles between state $\vec{0}$ and the states where one component has failed are almost equally likely as the straight paths from $\vec{0}$.

However, the contribution of $\hat{w}^*(\vec{0})$ needs to be time-dependent — cycles to the 'all up' state $\vec{0}$ are *only* likely when the 'all up' state's exit rate $\eta(\vec{0})$ is high enough compared to the remaining time $\tau - t$. Otherwise, the extra jumps take too much time, which reduces the likelihood of these paths.

For finding a time-dependent $\hat{w}(\vec{x}, t)$, a crucial insight is that the time-*in*dependent function $\hat{w}^*(\vec{0})$ is still a good approximation for the probability of hitting a system failure state *during a busy cycle*. For ease of notation, let $q \triangleq \hat{w}^*(\vec{0})$ and $\lambda_0 \triangleq \eta(\vec{0})$. When the failure rates are low or the repair rates are high, the duration of the busy cycle is almost completely determined by the time spent in state $\vec{0}$. Therefore, the time it takes before we reach a system failure state is the sum of M busy cycle durations D_i. Here, the durations D_i are all independent and exponentially distributed approximately with the rates λ_0, while the number M follows a geometric distribution with approximate success parameter q. From elementary probability theory we know that this sum follows an exponential distribution with rate $\lambda_0 \cdot q$, hence the probability that this is completed before $\tau - t$ time units has approximate probability $1 - \exp(-\lambda_0 \cdot q(\tau - t))$.

For small x, $1 - e^{-x}$ approximately equals x. Since q is assumed to be small, we can approximate $w(\vec{0}, t)$ using the time-dependent function $q\lambda_0(\tau - t)$. This motivates our final approximation,

$$\hat{w}(\vec{x}, t) = \begin{cases} q \cdot \lambda_0 \cdot (\tau - t) & \text{if } \vec{x} = \vec{0}, \\ \hat{w}^*(\vec{x}) + q \cdot \lambda_0 \cdot (\tau - t) & \text{otherwise.} \end{cases} \tag{10}$$

Using (10) in (8) keeps the estimator efficient when the rarity of the event of interest is not caused by the low component failure rates but rather the high recovery rates. Our numerical results will show that this adaptation is crucial in practical situations.

4.4. Steady State Unavailability

So far, we have described how to estimate the unreliability Φ^r, but a similar approach can be used for the unavailability V. Consider the ratio estimator (5). The problem that we face is that for the vast majority of runs the time fraction $Z(\omega)$ will equal zero, regardless of whether the failure rates were low or the repair rates were high. So, we need to increase the probability of hitting a system failure during a busy cycle.

The procedure will be as follows: we start in the 'all up' state and simulate using (10) substituted into (8) — since this is a steady-state performance measure we set $\tau - t \equiv 0$ in (10), thus effectively disabling returns to state $\vec{0}$. We stop when we reach system failure and from then on simulate using the old distribution until we reach state $\vec{0}$ [13]. Meanwhile, we record the amount of time during which the system was in a failed state.

If we would apply the same distribution as we used for (Z), the paths that immediately fall back to the 'all up' state before reaching a system failure state are never sampled because $\hat{w}(\vec{0}, t) \equiv 0$. This has no effect on the unbiasedness of the estimator $\hat{z}$ because paths that immediately fall back to $\vec{0}$ contribute nothing to $\mathbb{E}(Z)$. However, they do contribute heavily to $\mathbb{E}(D)$. Therefore, to avoid bias and inconsistency in $\hat{d}$ we generate two series of runs, one for Z with importance sampling and one for D without importance sampling [10], and substitute them into (5).

5. Results

In this section, we demonstrate that our method produces good results in practice. We compare our method to a few other well-known techniques. The first of these is the standard

Monte Carlo method. A more efficient method is that of forcing (see Section 4.1) combined with balanced failure biasing (BFB). Under BFB, the total probability of a component failure is set to $\frac{1}{2}$, uniformly distributed over the individual component types (and similarly for the repairs — for more information, see [22]). The third simulation method found in the result tables of this section are the estimates produced with the new method using the approximation in (10), abbreviated as Path-IS. Finally, we will compare our method to the numerical methods of the model checking tool PRISM.

When we display the experimental results in a table, we first give the statistical estimates. These are either the standard Monte Carlo estimates as in (4) or importance sampling estimates as in (6), which will be clear from the context, and are given in the form of a 95%-confidence interval. To the right of the estimates, we state the number of simulation runs used to produce these estimates. The number of simulation runs for each method was picked such that the computation time was comparable to that of PRISM. In the last row(s), we display the numerical solutions and the number of states in the PRISM and Arcade models (the latter tool uses lumping/bisimulation minimisation to reduce the size of the state space). The exact computation and simulation times are specified in the text.

5.1. Experimental Setup

Next, we will describe how we will test the strength of our method. Of course, there is a fundamental difference between the numerical approach and the statistical approach in the sense that numerical methods (if they converge) give an almost perfect (depending on the stopping criterion) approximation after some fixed time interval. On the other hand, statistical methods produce confidence intervals that can be made as narrow as one would like, depending on how much time one is willing to spend. The best way to say something about the applicability of an approach for the user is to look at the wall-clock time.

We used a computer with a 2.8 GHz Intel® Core™ 2 Duo processor (32-bit) and 3 GB of RAM, running Windows XP. All simulations were run with a simple Java program that generated (pseudo-)random numbers using a fast Mersenne twister [17]. We used version 3.3.1 of PRISM.

5.2. Unavailability

Of the two measures discussed in this paper, the unavailability is the easiest to approximate. Because it considers the system when it is in equilibrium, no information about the transient behaviour of the system is needed. Numerical methods to analytically determine or iteratively approximate it are well-established.

First, we will show in Table 2 that our results are consistent with the other tools and the literature, namely [4]. The unavailability in [4] was only given in one significant digit, and the total run time was not specified. When we lower the component failure rate parameter λ from 1/6000 to $\frac{1}{6} \cdot 10^{-6}$, we get similar results, with the exception of standard Monte Carlo. This is displayed in Table 3. Increasing μ from 1 to 1000 gives us equivalent results, as depicted in Table 4 (note that the unavailability values for $\lambda = \frac{1}{6} \cdot 10^{-6}$ and $\mu = 1000$ are exactly the same. This is not a coincidence, as the solution depends only on the transition rates through the ratio λ/μ). In all these cases PRISM does better than the simulation

	$\hat{v}$ (10^{-6})	# samples
MC	3.677 ± 0.778	388 196
BFB	3.647 ± 0.104	169 484
Path-IS	$\mathbf{3.511 \pm 0.035}$	**79 611**
	v (10^{-6})	# states
PRISM	3.498	421 875
Arcade	3	2 100

Table 2: Unavailability ($\hat{v}$) results for the benchmark case. $\lambda = 1/6000, \mu = 1, n = 2$.

	$\hat{v}$ (10^{-12})	# samples
MC	5.847 ± 11.460	386 538
BFB	3.532 ± 0.105	165 943
Path-IS	$\mathbf{3.521 \pm 0.036}$	**78 179**
	v (10^{-12})	# states
PRISM	3.500	421 875

Table 3: Unavailability ($\hat{v}$) results when $\lambda = \frac{1}{6} \cdot 10^{-6}; \mu = 1, n = 2$.

	$\hat{v}$ (10^{-12})	# samples
MC	—	384 418
BFB	3.504 ± 0.102	165 115
Path-IS	$\mathbf{3.465 \pm 0.035}$	**76 923**
	v (10^{-12})	# states
PRISM	3.500	421 875

Table 4: Unavailability ($\hat{v}$) results when $\mu = 1000$; $\lambda = 1/6000, n = 2$.

approaches discussed so far — indeed, for models with small state spaces PRISM's steady-state techniques can be preferred to simulation, regardless of λ or μ. However, if we increase the number of spare components n to 3, the size of the state space increases about 18-fold, as can be seen in Table 5. This causes PRISM's computation time to increase, from about 3.4 seconds for Tables 2–4 to 113.1 seconds for Table 6. When we increase n even further,

n	# states	# non-zeros
2	421 875	5 737 500
3	7 529 536	111 329 568
4	66 430 125	1 027 452 600
5	382 657 176	6 087 727 800
6	1 655 595 487	26 853 394 932

Table 5: State space sizes and numbers of non-zero entries in the transition rate matrix of the models built by PRISM for different values of n.

we hit tougher boundaries on the applicability of numerical methods due to the state space explosion problem. For $n \geq 4$, the amount of memory that our system has available for "*creating* [a] *vector for diagonals*" is insufficient and PRISM terminates without giving a

	$\hat{v}$ (10^{-9})	# samples
MC	7.235 ± 6.135	12 977 468
BFB	5.656 ± 0.151	3 434 986
Path-IS	$\mathbf{5.580 \pm 0.015}$	**1 315 050**
	v (10^{-9})	# states
PRISM	5.578	7 529 536

Table 6: Unavailability ($\hat{v}$) results when $n = 3$; $\mu = 1, \lambda = 1/6000$.

solution (even after adjusting the memory usage maxima in PRISM's settings). For $n = 6$, Path-IS still produces accurate estimates when we set the simulation time to a mere 60 seconds, as can be seen in Table 7. BFB underestimates the unavailability, a well-known phenomenon when the change of measure being used is not suitable for the problem [7].

	$\hat{v}$ (10^{-16})	# samples
MC	—	6 708 624
BFB	0.148 ± 0.225	803 752
Path-IS	$\mathbf{1.173 \pm 0.016}$	**205 654**
	v (10^{-16})	# states
PRISM	—	1 655 595 487

Table 7: Unavailability ($\hat{v}$) results when $n = 6$; $\mu = 1, \lambda = 1/6000$.

5.3. Unreliability

The unreliability is (from a theoretical point of view) a more interesting case than the unavailability because, unlike the latter, the former is not known in closed form for the models that we consider [9] — hence, we simply have to use numerical and/or statistical methods. First, note that we have defined the unreliability to refer to the probability of system failure before some time point τ (in this case 840 hours), allowing the repair of components in this time interval. In [4] and [21], component repairs were *not* allowed to occur.

In Table 8, we display the results for the benchmark case (no repairs allowed). Because PRISM's numerical evaluation was very quick (0.235 seconds), we gave the statistical methods more time (60 seconds), as the purpose of Table 8 is only to show that our results are consistent with the literature even when the repair transitions are disabled. Again, no run time was given for Arcade in [4]. Note that standard Monte Carlo and BFB give the best results in this setting because their simplicity allows them to sample many more runs within the (real) time constraint. When we allow repairs to occur the unreliability drops to approximately 0.0029. It takes PRISM little more than 30 seconds to compute this probability. This computation time does not depend on λ, as it took a comparable amount of time to generate the results of Table 9, where we lowered λ to $\frac{1}{6} \cdot 10^{-6}$.

However, when we increase μ, the time that PRISM needs to produce a solution increases along with it. The applied numerical methods require that the transition rate matrix be uniformised, and the uniformisation rate increases linearly in μ. PRISM's computation time

	$\hat{\pi}$	# samples
MC	0.5981 ± 0.0003	8 304 940
BFB	0.5976 ± 0.0003	5 116 887
Path-IS	$\mathbf{0.5977 \pm 0.0019}$	**93 526**
	π	# states
PRISM	0.5980	421 875
Arcade	0.5980	2 100

Table 8: Unreliability ($\hat{\pi}$) results without repair when $\mu = 0$; $n = 2$, $\lambda = 1/6000$.

in turn increases linearly in the product of the uniformisation rate and the mission time (see [12], chapter 15). Because the uniformisation rate is so much higher than the original exit rate of the 'all up' state, many unnecessary self-loops are taken into account. This can heavily slow down the computation. On the other hand, the accuracy of the Path-IS

	$\hat{\pi}$ (10^{-9})	# samples
MC	—	18 438 588
BFB	2.936 ± 0.024	1 042 866
Path-IS	$\mathbf{2.937 \pm 0.001}$	**992 231**
	π (10^{-9})	# states
PRISM	2.936	421 875

Table 9: Unreliability ($\hat{\pi}$) results when $\lambda = \frac{1}{6} \cdot 10^{-6}$; $n = 2$, $\mu = 1$.

estimate remains constant as μ increases since the jumps out of the 'all up' state still occur with the same low rate. A few estimates together with PRISM computation times are given in Figure 4. Notice that when $\mu = 100$, PRISM takes over half an hour to produce an approximation, while our simulation method can produce a decent estimate in 10 seconds.

For high μ (and high τ), the confidence intervals of BFB are also noticeably wider than those of Path-IS. For $\mu = 1000$ (10 second run time), they were 2.943 ± 0.013 and 2.920 ± 0.358 (times 10^{-6}) respectively. Again, discussing why goes beyond the scope of this paper.

For higher n PRISM again starts to suffer from the state-space explosion problem. We omit results for this scenario as they are comparable to the results for the unavailability when n is high.

6. General Busy Cycle Durations

Consider again a system that consists of several component types with a dedicated repair unit for each type. We previously assumed that repair would start immediately after a component failure, but now assume that, for one or multiple component types, the cost of having the repairman come over is high. In this setting, it might be more cost-efficient to initiate repair for a component type only after *multiple* components of this type have failed. This repair strategy is called *deferred repair*. Generalised versions of BFB were proposed in [14] and [15] for estimating steady-state measures in this model setting.

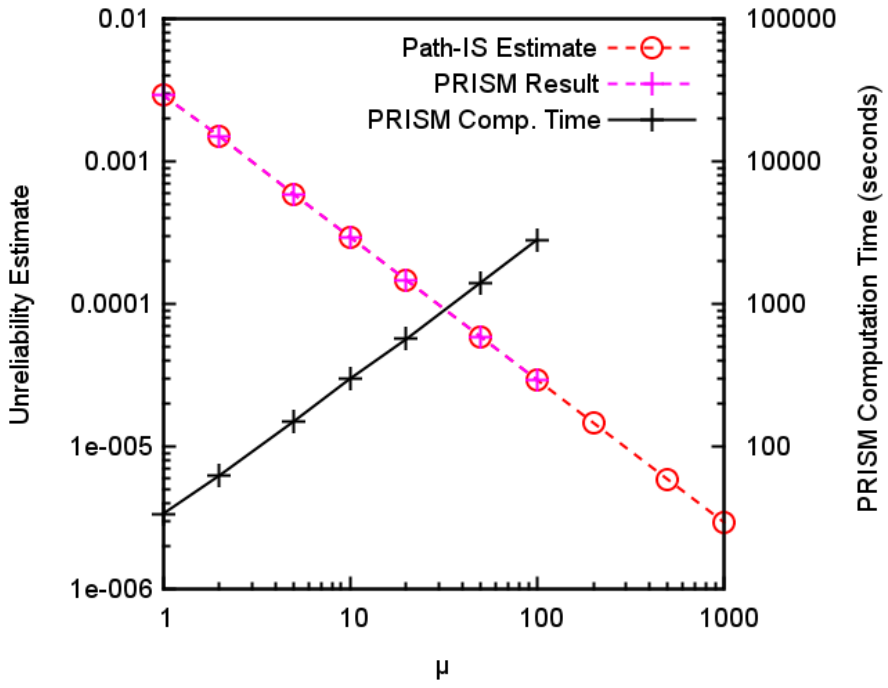

Figure 4: Estimates for the probability π (the unreliability) from PRISM (dashed, crosses) and Path-IS (dashed, circles) on left vertical axis; PRISM run time (solid, crosses) on right vertical axis. The Path-IS run time was only 10 seconds, but the bounds of the 95%-confidence interval were still not distinguishable from the estimate at this scale.

We cannot trivially assume that the method described in Section 4 also works well in this new setting; the analysis in this section relies on the busy cycle durations being approximately memoryless, but no matter how small λ/μ is in this new scenario, there will be more than one exponential phase in which a non-negligible amount of time is spent before system failure. Since μ is very high compared to λ, a non-negligible amount of time is spent *only* in states in which no repair transitions are allowed.

In other words, we need to divide the state space into two subsets: a *typical* set in which repair transitions are not allowed and in which the system spends the bulk of its time ($\{\vec{0}\}$ in the setting of Sections 2-4, but a larger number of states in the current setting), and an *atypical* set outside of it. Therefore, the probability distribution of the time until system failure will depend on the phase in the typical region, and phase-type distributions are not memoryless in general. Furthermore a good approximation $\hat{w}(\vec{x}, t)$ will require that information about a larger set of typical paths is used than just the straight paths of Section 4.2. These paths may '*turn*' inside the typical region, and are straight only after entering the atypical region.

In this section we will show how to overcome these problems. In Section 6.1 we will describe how the loss of the memoryless property can be overcome, by showing that in an appropriate limiting regime the importance of the current phase will vanish and the time

until system failure will again have a memoryless distribution. In Section 6.2, we will study the nature of the larger set of relevant paths, and in Section 6.3 we will demonstrate the necessity of the analysis in this section by empirically comparing the generalised method (which we shall call Path-IS-G) to the method of Section 4, standard simulation, BFB (see Section 5) and the numerical techniques of PRISM.

6.1. Non-Memoryless Busy Cycles

As we will only discuss the problem of non-memorylessness in this section, we consider a very simple model in which the need for extra paths does not occur. This is the case when our system consists of only one component type. A model of such a system is depicted in Figure 5, where the squares represent states in the typical set (i.e. where repair transitions are not yet allowed) and the circles represent states in the atypical set.

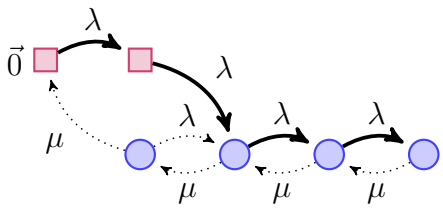

Figure 5: A model with a single component type. The repairman starts work only after the second component has failed.

For small λ/μ the duration of a busy cycle will be almost completely determined by the sojourn times in the typical set. Clearly, such a duration approximately has an Erlang(2,λ) distribution, for which the memoryless property does not hold. In more general situations, the distribution of the time spent in the typical set could have any phase-type distribution.

Although the time within a busy cycle is no longer memoryless, the distribution of the number of busy cycles needed before system failure remains geometric and, hence, memoryless. We know that λ/μ is small, so q, the probability of system failure during a single busy cycle, will also be small. As a consequence, the expected number of busy cycles before system failure will be large. Interestingly, it can be shown that for small values of q the time until system failure again approximately has an exponential distribution, with rate $q/\mathbb{E}(D)$; see Theorem 1 in the Appendix. Using the same linear approximation as in Section 4.3 we obtain an approximation $\hat{w}(\vec{x},t)$ similar to (10) with λ_0 replaced by $(\mathbb{E}(D))^{-1}$.

6.2. Non-Rare Paths

The next step is to extend the result of Section 6.1 to a system with *multiple* component types — a model of such a system is depicted in Figure 6(a). It still holds for this system that the duration of the busy cycle is approximately equal to the time spent in the typical set. However, when a system has a large number of component types with comparable failure rates, it is likely that between failures of two components of the same type, a component of another type fails. The '*straight*' paths of Section 4.2 do *not* account for such behaviour. As a consequence, the probability q of failure during a busy cycle is underestimated, and

the resulting change of measure will be inefficient because the probability of falling back to the origin is made too low.

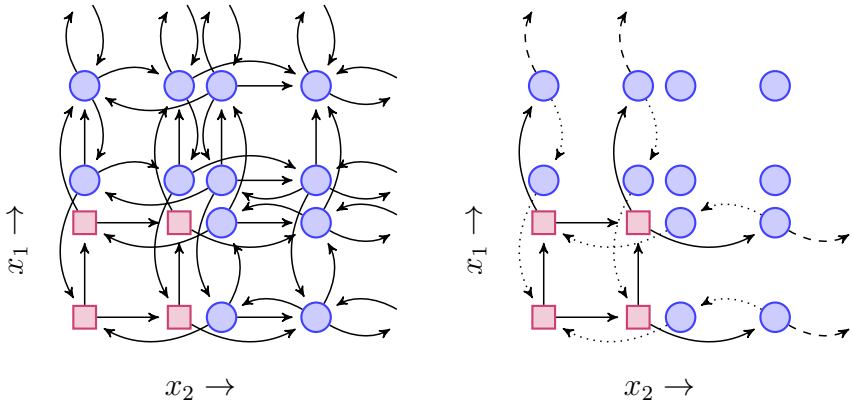

Figure 6: A model consisting of two types of components, where for each type, repair starts when two components of that type have failed. The square states represent the typical set, the round states represent the atypical set. In (a) we display all possible transitions; in the square states none of the repairs are yet enabled. In (b) we display only the most important transitions; dashed: transitions in a typical successful busy cycle; dotted: transitions in a typical *un*successful busy cycle; solid: transitions that occur in both.

Another consequence of the added component types is that the expected duration of a busy cycle may be very long. Not only does this impact the efficiency with which we can estimate $\mathbb{E}(D)$, it may also imply that the assumption, made in Section 6.1, that the probability of system failure during a busy cycle q is small while τ is relatively large compared to $\mathbb{E}(D)$, is violated for realistic parameter settings.

Hence, we alter the definition of the busy cycle: we now say that a *generalised* busy cycle D' starts and ends when the system jumps from the *atypical* to the *typical* set. Note that the starting times of the busy cycles are no longer i.i.d., but this has a negligible impact. Hence, we can still justifiably approximate the time until system failure by a geometrically distributed number of (generalised) busy cycle durations.

We approximate the probability q of failure in such a generalised busy cycle using the following strategy: we use simulation using the *original* probability distribution (see Section 2.2) to estimate $\mathbb{E}(D')$. This means that we choose a number N_D a priori or let it depend on a bound on the run time. Because a fixed time bound leads to strongly varying accuracies of $\mathbb{E}(D')$ for different n and λ (the number of sample generated per time unit depends strongly on these model parameters) we fixed $N_D = 500$ for the results of Section 6.3.

Whenever a new generalised busy cycle starts during this first simulation round, we record the state $\vec{x}$ in which the typical set is entered. We then find all paths that first move (in any way) through the typical set and then follow a straight path to system failure. In the example of Figure 6(b), there would be four of those paths from $\vec{0}$ and three from the states $(0, 1)$ and $(1, 0)$ in the typical set. Then we use the sum of the probabilities of all those paths — we denote this value by $\hat{q}^*(\vec{x})$ — as a single *sample* that is used to find an

estimate for q. Hence, our approximation for q is a random variable Q whose expected value we estimate using the same simulation runs as the ones for $\mathbb{E}(D')$. Finally, after having generated N_D samples, we use the sample means as estimates for $\mathbb{E}(D')$ and $\mathbb{E}(Q)$, which leads to our final approximation:

$$\hat{w}(\vec{x},t) = \begin{cases} \mathbb{E}(Q)\cdot(\mathbb{E}(D'))^{-1}\cdot(\tau-t), & \text{if } \vec{x}=\vec{0}, \\ \hat{q}^*(\vec{x}) + \mathbb{E}(Q)\cdot(\mathbb{E}(D'))^{-1}\cdot(\tau-t), & \text{otherwise.} \end{cases} \tag{11}$$

In Section 6.3 we will only display results for the unreliability, but one could also use the change of measure in (11) to estimate the steady-state unavailability. One would then again set $(\tau - t)$ to zero and the only change would be the number of paths used for $\hat{w}^*$. Note that for the ratio estimator (5) we still use busy cycle durations D instead of D' as the latter ones do not form a renewal process.

6.3. Simulation Results

In this section, we will empirically compare the simulation distribution based on (10) (Path-IS) to the more general simulation distribution based on (11) (Path-IS-G). We introduce a vector of repair triggers $r_i, i \in \{1,\ldots,9\}$. Repair of component of type i is turned on when the number of failed components of type i reaches the trigger r_i, and then switched off when all failed components of that type have been repaired. Typically, this will increase the expected generalised busy cycle duration but also increase the probability q of failure during such a cycle both due to an increased number of ways failure can occur (more paths to failure) and the fact that fewer failure transitions need to '*win the race*' against repair transitions.

6.3.1. Single Component Type

We will first consider the increase in expected busy cycle duration by considering the processor set in isolation. We increase τ from the benchmark 840 to 8 400 because in the new setting the generalised busy cycle will take so long that the assumption that cycles are completed before system failure is no longer valid. The CTMC that underlies the model will then look like the one in Figure 5 — there are r_i states in the typical set and n states in the atypical set. For Figure 7, we let $r_i = n/2$ and show the estimates and numerical results of PRISM for increasing n. The probability of interest can be seen to decrease exponentially. In this situation, PRISM is still clearly superior as the model size is very small when only one component type is modeled.

The 95%-confidence intervals in Figure 7 were too narrow to be visible. However, to get an idea of how broad they are compared to each other, we display the relative error $\hat{\sigma}/(\hat{\pi}\sqrt{N})$ in Figure 8. At $n = 2$, the two methods are still the same as repair starts immediately after a component has failed. However, for larger values of n the generalised approximation Path-IS-G clearly outperforms the original Path-IS. When n (and, hence, all r_i) grows higher, the difference between the two '*estimates*' for $\mathbb{E}(D)$ and $\mathbb{E}(D')$ increases. As a side note: for higher τ the efficiency of the change of measure based on (10) will decrease further.

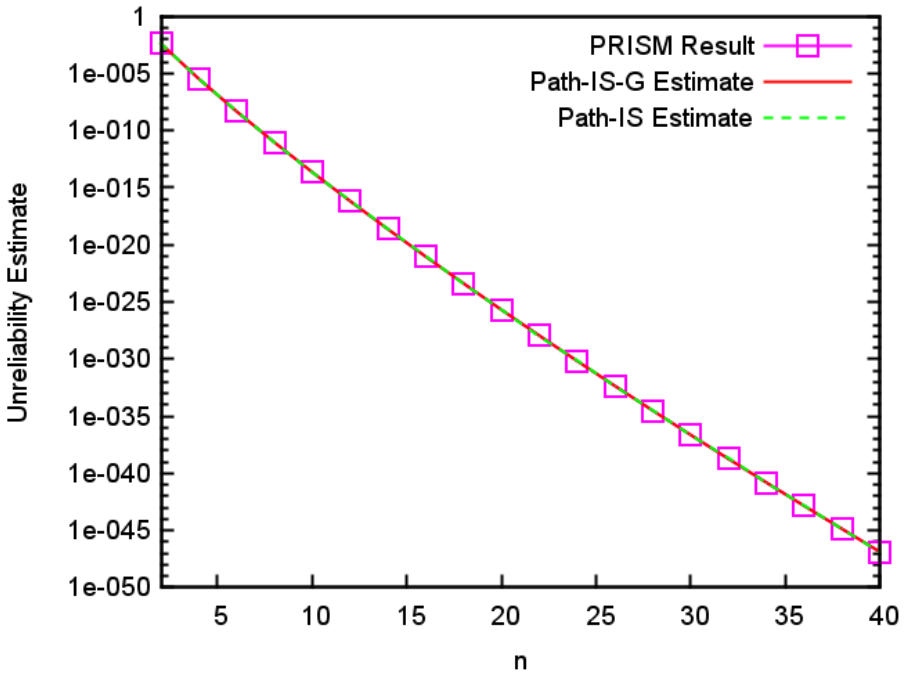

Figure 7: Unreliability ($\hat{\pi}$) results for the model with one component type. $\lambda = 1/6000$, $\mu = 1$, $\tau = 8\,400$, $r_1 = n/2$. $N = 1000$ samples were drawn for each simulation — the run time for each simulation was less than five seconds. PRISM run time was negligible.

6.3.2. Multiple Component Types

Whereas PRISM is still the most efficient tool for analysing small models such as the Distributed Database System (DDS) with only one component type, its performance is much worse for models with larger values of n. For $n = 3$ and $r_i = 2$ for all i, the size of the state space is 32 768 000 compared to the 7 529 536 of Table 5 — the number of non-zero entries in the transition rate matrix increases by a similar factor. PRISM runs out of memory in this situation, so in order to be able to compare the methods we remove two disk sets from the model, resulting in seven component types. This model only has 512 000 states and 5 478 400 transitions. It takes PRISM about 44 seconds to find a solution. In Table 10 we combine this result with the results of simulations with a similar run time.

As expected, Table 10 shows that a change of measure based on (11) outperforms one based on (10). The difference is not dramatic, however. This is because the longer busy cycle durations and higher probability of success during a busy cycle cancel to some extent, causing the rates in the exponential approximation in the two methods to differ by a factor of only 2.2.

Note that the results produced by Path-IS are still usable; however, once we raise τ, its performance will again deteriorate. The performance of the two methods does not worsen for higher μ, which can be seen in Table 11 for which we raised the value of μ from 1 to 1000. PRISM is still able to produce a result in this situation but only after 10 hours, whereas

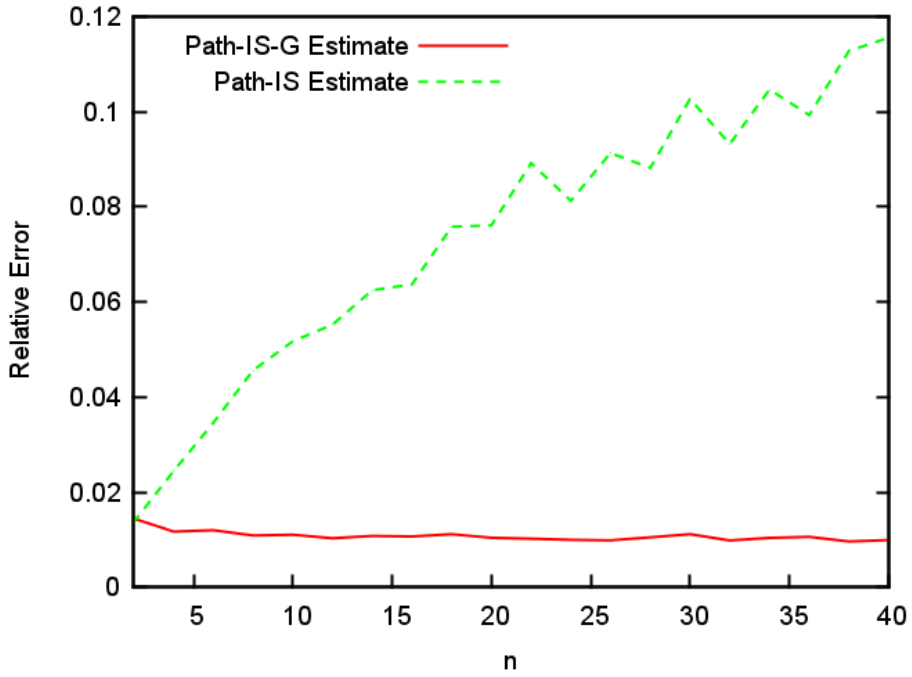

Figure 8: Relative errors $\hat{\sigma}/(\hat{\pi}\sqrt{N})$ for the model with one component type. $\lambda = 1/6000$, $\mu = 1$, $\tau = 8\,400$ $r_1 = n/2$. $N = 1000$ samples were drawn for each simulation — the run time for each simulation was less than five seconds.

	$\hat{\pi}$ (10^{-4})	# samples
MC	9.296 ± 0.503	1 410 356
BFB	8.691 ± 0.833	382 539
Path-IS	9.483 ± 0.183	24 319
Path-IS-G	$\mathbf{9.313 \pm 0.105}$	**19 197**
	π (10^{-4})	# states
PRISM	9.366	512 000

Table 10: Unreliability ($\hat{\pi}$) results for the model with 7 component types. $r_i = 2$; $n = 3$, $\lambda = 1/6000$, $\mu = 1$, $\tau = 840$.

the run time of the simulation-based methods was 44 seconds. In Figure 9, the relative error $\hat{\sigma}/(\hat{\pi}\sqrt{N})$ is displayed for increasing values of μ. We can see that the relative error of Path-IS is not only higher but also much more rugged, which shows that the estimate is more unreliable. More importantly, the relative error of Path-IS-G remains almost constant; this implies that, unlike Monte Carlo simulation, the time complexity of Path-IS-G does not depend on μ and thereby not on p, the probability that we seek to estimate.

6.4. Generalised Repair Strategies

The defining characteristic of the method presented in this paper is that it is *path-based*; it works well if failures mostly occur in a manner that corresponds to one of the

	$\hat{\pi}$ (10^{-7})	# samples
MC	6.696 ± 13.12	1 413 429
BFB	—	301 322
Path-IS	9.458 ± 0.178	23 872
Path-IS-G	$\mathbf{9.350 \pm 0.103}$	**19 566**
	π (10^{-7})	# states
PRISM	9.388	512 000

Table 11: Unreliability ($\hat{\pi}$) results for the model with 7 component types. $\mu = 1000$; $n = 3$, $r_i = 2$, $\lambda = 1/6000$, $\tau = 840$.

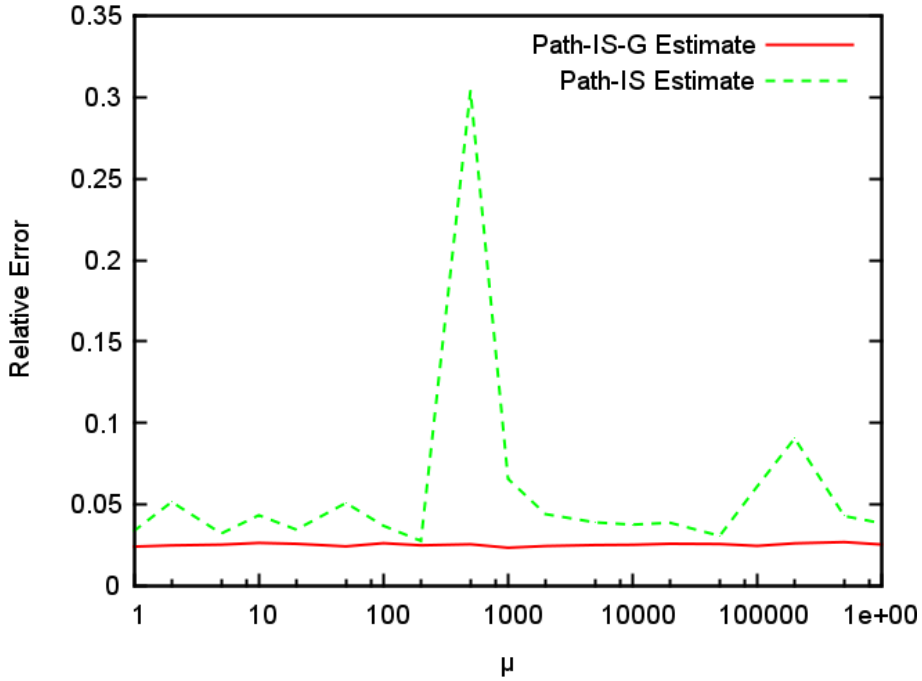

Figure 9: Relative errors $\hat{\sigma}/(\hat{\pi}\sqrt{N})$ for the model with 7 component types. $\lambda = 1/6000$, $n = 3$, $\tau = 840$, $r_1 = \ldots = r_7 = 2$. $N = 1000$ samples were drawn for each simulation. Note the the peaks are not structural, but simply evidence of the whimsicality of the estimator.

dominant paths. In Section 2.1, we outlined a model class for which we can *guarantee* that this assumption is valid. However, the class of models for which our method (or variations thereof) works well is much broader and contains other models with large practical relevance. To be more specific: we formulated two requirements in Section 2.1 that our system model's repair strategy had to meet: *i*) for each component type, repairs had to be handled by a dedicated repair facility, independent of all other component types and *ii*) repairs had to start immediately after the first component of a type had failed. In Sections 6.1-6.3 we have shown how to drop the latter assumption. Additionally, however, one could also think of good reasons to deop the former.

First of all, it is simply unrealistic to assume that two disk sets have two separate repair facilities. A *single* repair facility taking care of all the repairs in the system is often closer to practice. Secondly, models with dependent component types are more theoretically appealing. If all component types are independent, we can compute several important probabilities in the system by first computing them for each component in isolation and then performing a trivial aggregation step.

Of course, generalising to a single repair facility for all component types means we have to specify a repair *policy* that determines how the repair facility decides which component type to work on whenever components of multiple types are down. From the many policies that have been studied in the literature (see also Chapter 6 of [12]), we will zoom in on one policy, namely true FCFS, to illustrate the wider applicability of our method. True FCFS means that individual components are repaired in the order in which they fail.

The impact of this choice of repair strategy on the state space of our model is such that each state in which several components have failed must be duplicated such that all possible orders of failure can be represented, not discriminating between components of the same type. This has a strong negative effect on the performance of PRISM; not only is expressing this model in the PRISM modelling language a challenge, there is also a dramatic blow-up in the size of the state space. In the benchmark setting, the size of the state space goes from 421 875 states to 2 123 047 371 states.

In Table 12, we have displayed simulation results for both the old situation and the new one. Each simulation was run for 33.609 seconds, which is the amount of time needed by PRISM to compute the exact probability in the old setting. The difference between the results of the two repair strategies is marginal: only the path-based IS estimates differ significantly. The reason is that for the system to reach a state in which the two strategies result in different outgoing transition structures, a failure transition must win the race from a repair transition. Since the probability of this happening is assumed to be small for Path-IS-G to work well, it will be hard to notice a difference.

	dedicated repair		1 facility, FCFS	
	$\hat{\pi}$ (10^{-3})	# samples	$\hat{\pi}$ (10^{-3})	# samples
MC	2.941 ± 0.106	1 009 973	2.956 ± 0.138	596 136
BFB	2.975 ± 0.176	317 989	2.790 ± 0.222	189 032
Path-IS-G	$\mathbf{2.901 \pm 0.020}$	**21 508**	$\mathbf{2.960 \pm 0.027}$	**11 311**
	π	# states	π	# states
PRISM	2.928	421 875	N.A.	2 123 047 371

Table 12: Unavailability ($\hat{v}$) results when $\lambda = 1/6000$, $\mu = 1$, $n = 2$.

One can check that if we were to switch to deferred repair we would also see that the path-based methods work well if the assumptions are such that there is little difference between dedicated repair and a single repairman. However, if we assume that the components of different types are physically close to each other, this strategy is not only theoretically unappealing but also unrealistic; if there is a single repairman for all component types who would come over to fix broken processors, he would most likely also directly repair a failed

disk even if the repair threshold for disks of its type had not been reached.

So consider the following strategy: we again have deferred repair, but when for *any one* component type i a number r_i components have failed, the repairman will come over and begin repair on *all* broken components and leave when the system is back in the 'all up' state. The repairs are handled in a first-come first-serve fashion irrespective of which component type triggered the repairman to come over.

	$\mu = 1$		$\mu = 1000$	
	$\hat{v}$ (10^{-3})	# samples	$\hat{v}$ (10^{-6})	# samples
MC	1.748 ± 0.061	1 784 962	2.273 ± 2.228	1 759 644
BFB	1.749 ± 0.098	602 621	—	479 383
Path-IS	$\mathbf{1.779 \pm 0.029}$	**20 524**	$\mathbf{1.800 \pm 0.025}$	**20 773**

Table 13: Unavailability ($\hat{v}$) results with one repairman who repairs all broken components of all types before leaving in a first-come serve fashion; $\lambda = 1/6000, n = 3, r_i = 2$.

In Table 13, we display the simulation results for this setting with $r_1 = \ldots = r_9 = 2$, $n = 3$ and λ and μ equal to the benchmark values. We would run into problems if we tried to directly apply the method of Section 6.2 to this situation as even the *typical* set suffers from the state space explosion problem. For example, due to the FCFS policy there are $9! = 362\ 990$ states in which one component of each type has failed, and these are all in the typical set. To circumvent this problem we (incorrectly) assume that for all such states (that differ only in the queue order), the sum of the straight paths leading out of those states will be the same. While this will introduce an error, this error does not get worse as λ gets lower or μ gets smaller, and the resulting estimator remains much better than the standard Monte Carlo estimator whose performance will degrade sharply. In the end, our approximation needs not be exact for the estimate to remain unbiased. As can be seen in Table 13, our method still performs well.

7. Conclusions and Discussion

In this section, we first draw the general conclusions. Then we discuss our method's computational complexity and compare it with other methods. Finally, we discuss generalisations of the method and future work.

7.1. General Conclusions

In this paper we have introduced an efficient simulation technique that is able to estimate dependability measures in situations were system failure is a rare event due to high repair rates or low component failure rates. The approach that we used is based on the zero-variance measure for transient failure probabilities in CTMCs, approximated using the likely paths to failure. We have shown how to apply this method when a more complicated repair strategy such as deferred (or, by analogy, group) repair is used.

We have demonstrated that our technique performs well even for large models as long as the component failure rates are much lower than the repair rates. Also, we have shown that

our method performs well in comparison to other methods. The method of Balanced Failure Biasing combined with forcing is only well-suited for the low λ situations and performs poorly for high μ and n. Numerical techniques, as, e.g., implemented in PRISM, suffer from large state spaces and high uniformisation rates. Table 14 summarises this comparison by presenting the best choice of method for given parameter settings.

Performance measure	low λ	high μ	high n
Unreliability	PRISM	Path−IS	Path−IS
Unavailability	PRISM	PRISM	Path−IS

Table 14: Which method performs best depends on parametric regime and performance measure.

Note that the method is not specifically designed for high n. If n is too high, Path-IS will also start to see worsening performance [4]. However, for n not too large and λ/μ sufficiently small, Path-IS will have good performance in situations where PRISM runs out of memory. Adapting the simulation for high n situations is part of ongoing research.

7.2. Complexity

In Table 15, the time and space complexities of the methods whose efficiencies are compared in this paper are displayed. For the numerical and standard statistical methods the values of the complexities were given in [24] — we rewrote these complexities in terms of the model parameters used in this paper. The expected time complexity of the statistical methods is inversely proportional to p, the probability that we want to estimate. Using importance sampling, we lose this dependence on p. However, d paths of maximum length n need to be evaluated in each step of the simulation for $O(d)$ potential successor states. Still, Path-IS avoids the two main causes of quickly increasing run time (i.e. the state space explosion problem for the numerical methods and event rarity for standard simulation).

Method	Time Complexity	Space Complexity
Numerical	$O((\lambda+\mu)\tau d n^d)$	$O(dn^d)$
Standard Simulation	$O(\lambda\tau d p^{-1})$	$O(d)$
Path-IS	$O(\lambda\tau d^2 n)$	$O(d)$

Table 15: Time and space complexities for different methods (repairs always enabled).

In the setting of Section 6, the number of paths that needs to be evaluated at each step increases; there are $\prod r_i$ states in the typical set and there are at most $d \cdot \prod r_i$ paths inside the typical set that lead to states on the '*edge*' of the typical set. These extra paths lead to an increase in the time complexity of the method. This can be seen by comparing Table 15 with Table 16.

[4] When the level of redundancy is very high, the rarity of the event may not primarily be caused by the ratio λ/μ but by the fact that there is not enough time for all of the components to fail during the mission time. Fortunately, this is not a common scenario in the analysis of multi-component systems, but it might be encountered in other contexts.

To speed up Path-IS-G, we used a caching approach: for each state in the typical set we store the sum of the probabilities of the paths to system failure. This results in a higher memory complexity but cuts the time complexity in the typical set back to only $O(\lambda\tau dn)$ assuming that lookups in the list of stored probabilities do not contribute to the time complexity (this is possible using cleverly linked lists where each entry points to all states that can be reached in one transition). Outside the typical set, the time complexity is again $O(\lambda\tau d^2 n)$.

Method	Time Complexity	Space Complexity
Path-IS-G (no caching)	$O(\lambda\tau d^3 n \prod r_i^2)$	$O(d)$
Path-IS-G (caching)	$O(\lambda\tau d^2 n)$	$O(d \prod r_i)$

Table 16: Time and space complexities for Path-IS-G (repair of type i enabled after r_i components of that type have failed).

7.3. Generalisations and Future Work

In this paper, we have only considered system failures caused by the number of failures of *some* component type i reaching a critical level n_i. The method is also applicable with more general failure conditions, such as simultaneous failure of at least n_i components of each type i. More paths to failure may then need to be considered for a good approximation $\hat{w}$ — perhaps even a number of paths that increases exponentially in the number of component types — but many of them typically have equal probability which makes accounting for them easier. Similarly, it is straightforward to generalise the extension of Section 6 to group repair strategies (see [14]). Another interesting, but more challenging, generalisation is to allow component interdependence, which occurs, for example, when components share repair facilities that follow some predefined repair strategy (see also Section 6.4). Still, when an easily identifiable class of '*straight*' paths adequately describes the typical model behaviour leading up to system failure, the methods presented in this paper can be expected to perform well.

Furthermore, an interesting extension of the method would be to allow the failure and repair time distributions to be any phase-type distribution. Part of the work for this is already done, since the Path-IS-G method discussed in Section 6 already considers phase-type distributions in the 'typical' part of the state-space. For the rest of the state-space, a change of measure for the phase-type failure and repair times would be needed; the so-called exponential change of measure [13] can be expected to work well.

Future work includes extending the method to the case of very large n, which is an interesting scenario in which numerical methods suffer from the state space explosion problem. Finally, we plan to add rewards to the model.

Appendix

The goal of this appendix is to motivate the use of generalised busy cycles by proving that despite the fact that the duration of each busy cycle is no longer exponentially distributed,

the sum of a geometrically distributed sum of these durations is approximately exponentially distributed if the success parameter of the geometric distribution is small. First we will state the theorem, then we explain why we need the theorem, then we prove the theorem and conclude with a short discussion on how we use it in this paper.

Theorem 1. *Let $X_1, X_2, \ldots$ be a sequence of i.i.d. random variables such that $\mathbb{E}(X_1) < \infty$, and let M be a random variable independent of X_i, $i \in \mathbb{N}$, that has a geometric distribution with success parameter q. Let $S_n = X_1 + X_2 + \ldots + X_n$. Then*

$$\lim_{q \downarrow 0} \mathbb{P}(qS_M > t) = e^{-t/\mathbb{E}(X_1)}.$$

The preceding theorem states that for small q, $\mathbb{P}(qS_M > t) \approx \exp(-t \cdot (\mathbb{E}(X_1))^{-1})$, so this implies that $\mathbb{P}(S_M > t) \approx \exp(-t \cdot q \cdot (\mathbb{E}(X_1))^{-1})$ which is what we used in Section 6.1.

Proof. Since a probability distribution is uniquely characterised by its Laplace-Stieltjes Transform (LST) we will consider the LST of qS_M. For the LST of X_1 we know that, since $\mathbb{E}(X_1) < \infty$, we can write

$$\mathbb{E}\left(e^{-sX_1}\right) = 1 - s\mathbb{E}(X_1) + o(s),$$

where $o(s)$ stands for a function $f(s)$ satisfying $\lim_{s \downarrow 0} \frac{f(s)}{s} = 0$. Furthermore, it is known that the probability generating function (PGF) of a geometrically distributed random variable M with probability q is

$$\mathbb{E}(z^M) = \frac{q}{1 - (1-q)z},$$

Then with M geometrically distributed it holds that

$$\begin{aligned}
\mathbb{E}\left(e^{-sqS_M}\right) &= \mathbb{E}\left(\mathbb{E}\left(e^{-sqS_M}|M\right)\right) = \mathbb{E}\left(\left[\mathbb{E}\left(e^{-sqX_1}\right)\right]^M\right) \\
&= \frac{q}{1 - (1-q)\mathbb{E}\left(e^{-sqX_1}\right)} \\
&= \frac{q}{1 - (1-q)(1 - sq\mathbb{E}(X_1) + o(sq))} \\
&= \frac{q}{q + (1-q)sq\mathbb{E}(X_1) + o(sq)} \\
&\xrightarrow{q \downarrow 0} \frac{1}{1 + s\mathbb{E}(X_1)},
\end{aligned}$$

which we recognise as the LST of an exponentially distributed random variable with mean $\mathbb{E}(X_1)$.

□

We note here that in the setting in which we will apply the theorem, not all of its assumptions are satisfied: first of all, even if there is only *one* state in the typical set, M and the series $X_1, X_2, \ldots$ are not completely independent. After all, X_M has a different probability distribution than $X_1, X_2, \ldots, X_{M-1}$ (a busy cycle in which system failure occurred

went '*deeper*' into the state space than a typical busy cycle, so it can be expected to have lasted longer.) However, we expect this not to have much impact on our conclusions for two reasons: first of all, when q goes to zero, M will grow larger and the relative influence of X_M with respect to $X_1, X_2, \ldots, X_{M-1}$ will vanish. Furthermore, in our setting $q \downarrow 0$ because $\lambda \downarrow 0$ or $\mu \to \infty$, and in exactly these two regimes the difference between the distributions of X_M and the preceding busy cycle durations will vanish as the time needed to go 'deeper' into the state space becomes smaller relative to the time spent in the typical set's only state.

In our situation there is more than one state in the typical set, the point of entry into the typical set will have influence on the next point of entry (it is less likely that this is the same state twice in a row). This means that there is correlation between each X_i and X_j, $i, j > 0$, but this vanishes as $|i - j|$ gets bigger. The assumption that we make is that in our limiting regime (i.e. $q \downarrow 0$), the series of busy cycles until system failure becomes so large that the influence of the correlation between the individual random variables becomes smaller as well. Judging by our empirically determined estimator variances, this assumption is justifiable.

Acknowledgements

This work is supported by the Netherlands Organisation for Scientific Research (NWO), project number 612.064.812.

References

[1] A. Aziz, K. Sanwal, V. Singhal, and R.K. Brayton. Verifying continuous-time Markov chains. *Lecture Notes in Computer Science*, 1102:269–276, 1996.

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

[3] F. Bause and P.S. Kritzinger. *Stochastic Petri Nets*. Vieweg, 2002.

[4] H. Boudali, P. Crouzen, B.R. Haverkort, M. Kuntz, and M. Stoelinga. Arcade-A formal, extensible, model-based dependability evaluation framework. In *13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 08), Belfast*, volume 3, pages 243–248. Citeseer, 2008.

[5] J.A. Bucklew and R. Radeke. On the Monte Carlo Simulation of Digital Communication Systems in Gaussian Noise. *IEEE Trans. on Communications*, 51(2), 2003.

[6] P.T. de Boer, P. L'Ecuyer, G. Rubino, and B. Tuffin. Estimating the probability of a rare event over a finite time horizon. In *Proceedings of the 2007 Winter Simulation Conference*, pages 403–411, 2007.

[7] M. Devetsikiotis and J.K. Townsend. An algorithmic approach to the optimization of importance sampling parameters in digital communication system simulation. *IEEE Transactions on Communications*, 41(10):1464–1473, 1993.

[8] A. Goyal, W.C. Carter, E. de Souza e Silva, S.S. Lavenberg, and K.S. Trivedi. The system availability estimator. In *Proceedings of the 16th Int. Symp. on Fault-Tolerant Computing*, pages 84–89, 1986.

[9] A. Goyal, S.S. Lavenberg, and K.S. Trivedi. Probabilistic modeling of computer system availability. *Annals of Operations Research*, 8(1):285–306, 1987.

[10] A. Goyal, P. Shahabuddin, P. Heidelberger, V.F. Nicola, and P.W. Glynn. A unified framework for simulating Markovian models of highly dependable systems. *IEEE Transactions on Computers*, 41(1):36–51, 1992.

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

[12] B.R. Haverkort. *Performance of computer communication systems: a model-based approach*. John Wiley & Sons, Inc. New York, NY, USA, 1998.

[13] P. Heidelberger. Fast simulation of rare events in queueing and reliability models. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 5(1):43–85, 1995.

[14] S. Juneja and P. Shahabuddin. Fast simulation of markov chains with small transition probabilities. *Management Science*, pages 547–562, 2001.

[15] S. Juneja and P. Shahabuddin. Splitting-based importance-sampling algorithm for fast simulation of markov reliability models with general repair-policies. *Reliability, IEEE Transactions on*, 50(3):235–245, 2001.

[16] A.M. Law and W.D. Kelton. *Simulation modeling and analysis*. McGraw-Hill New York, 1991.

[17] S. Luke. Fast Mersenne Twister. http://www.cs.gmu.edu/∼sean/research/.

[18] M.K. Nakayama and P. Shahabuddin. Quick simulation methods for estimating the unreliability of regenerative models of large, highly reliable systems. *Probability in the Engineering and Informational Sciences*, 18(03):339–368, 2004.

[19] V.F. Nicola, M.K. Nakayama, P. Heidelberger, and A. Goyal. Fast simulation of highly dependable systems with general failure and repair processes. *IEEE Transactions on Computers*, 42(12):1440–1452, 1993.

[20] D. Reijsbergen, P.T. de Boer, W. Scheinhardt, and B. Haverkort. Rare event simulation for highly dependable systems with fast repairs. In *Quantitative Evaluation of Systems (QEST), 2010 Seventh International Conference on the*, pages 251–260. IEEE, 2010.

[21] W.H. Sanders and L.M. Malhis. Dependability evaluation using composed SAN-based reward models. *Journal of parallel and distributed computing*, 15(3):238–254, 1992.

[22] P. Shahabuddin. Importance sampling for the simulation of highly reliable Markovian systems. *Management Science*, pages 333–352, 1994.

[23] SAE AADL working group. *Architecture Analysis and Design Language (AADL)*. SAE standards AS5506, Nov 2004, http://www.sae.org/technical/standards/AS5506.

[24] H.L.S. Younes, M. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(3):216–228, 2006.

[25] H.L.S. Younes and R.G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proceedings of the 14th International Conference on Computer Aided Verification,*, pages 223–235, 2002.