# The Degree of Masking Fault Tolerance vs. Temporal Redundancy

Nils Müllner and Oliver Theel
*Department of Computer Science*
*Carl von Ossietzky Universität Oldenburg*
*26111 Oldenburg, Germany*
*Email: {nils.muellner|oliver.theel}@informatik.uni-oldenburg.de*
*Phone: +49-441-798-2364*
*Fax: +49-441-798-2756*

***Abstract*—Self-stabilizing systems, intended to run for a long time, commonly have to cope with transient faults during their mission. We model the behavior of a distributed self-stabilizing system under such a fault model as a Markov chain. Adding fault detection to a self-correcting non-masking fault tolerant system is required to progress from non-masking systems towards their masking fault tolerant functional equivalents. We introduce a novel measure, called limiting window availability (LWA) and apply it on self-stabilizing systems in order to quantify the probability of (masked) stabilization against the time that is needed for stabilization. We show how to calculate LWA based on Markov chains: first, by a straightforward Markov chain modeling and second, by using a suitable abstraction resulting in a space-reduced Markov chain. The proposed abstraction can in particular be applied to spot fault tolerance bottlenecks in the system design.**

***Keywords*-Distributed Algorithms, Fault Tolerance, Masking, Non-masking, Self-Stabilization**

## I. Introduction

We distinguish four classes of fault tolerance: 1) intolerant, 2) failsafe, 3) non-masking, and 4) masking fault tolerant systems (cf. Table I) as proposed in [4, p.8].

| | safe | not safe |
|---|---|---|
| live | masking | non-masking |
| not live | failsafe | intolerant |

Table I
Fault Tolerance Classes of Systems Based on Their Liveness and Safety Property Specifications [4, p.8]

Live systems generally utilize correctors and safe systems employ detectors [1]. Detectors on the one hand enable the system to tell whether it is – wrt. a subsystem the detector supervises – currently working according to its specification or not. Hence, the failsafe property allows a system to trigger a safe standstill, preventing it from working unsafe by compromising liveness and potentially causing trouble.

Correctors on the other hand give a system the ability to repair certain errors (i.e., re-establish safety). Like detectors, correctors may be explicitly implemented. But in some systems, the corrector functionality is somehow "inherently" given like in most self-stabilizing systems. Those non-masking systems can repair themselves without explicitly knowing that they violate their safety property (and without knowing that they currently repair errors). Masking fault tolerant systems employ both, detectors as well as correctors, either implicitly or explicitly or in any combination thereof.

In this paper, we focus on long running (live) systems that are already capable of correcting themselves in order to re-establish the safety property. Until now, the methods introduced in literature enhanced an intolerant system towards non-masking fault tolerance by adding correctors. Subsequently, the addition of detectors allowed those systems to become masking fault tolerant [1], [5], [6], [7]. Yet, the fault tolerance specification was given wrt. a certain fault model and the systems were required to completely mask all the mishaps specified by the fault model. These approaches, though, did not analyze *probabilistic* masking of otherwise non-masking fault tolerant systems. They only stated the conditions under which a system was (perfectly) masking or not.

In the approach presented here, we also amend live (non-masking) systems to become maskingly fault tolerant up to a certain probability (or degree). We focus on the addition of a detection layer called *fault masker* [10]. But instead of specifying, which faults can be masked under certain preconditions, we introduce a method that allows for the *exact calculation* of the probability distribution with which faults are masked in relation to the efforts spent for error correction. The approach can straightforwardly be used in settings where subsystems supervised by the fault masker can fail and can be repaired or exchanged.

We restrict ourselves to only use time, i.e. temporal redundancy, as form of redundancy for error correction in order to – as purely as possible – pin-point temporal redundancy vs. the "degree of masking." Although also possible, we do not exploit forms of spatial redundancy to be used by the corrector, as, for example, adopted in error correcting codes (ECC)[1]. Clearly, both, temporal and spatial redundancy, are resources that can be used to increase the "degree of masking" of an otherwise non-masking system.

We illustrate our approach by an example: a self-stabilizing and hence non-masking distributed system. In order to evolve from non-masking towards masking, we require a detector. We deploy a detector module called *fault masker* [10] on the (self-stabilizing) system so that we can measure to which extent the system masks faults. As we increase the amount of time that a system is allowed

[1]Even self-stabilization requires "a little" spatial redundancy (e.g., space for additional code) to actually utilize time for achieving fault tolerance. Here, we distinguish between functional space redundancy as used for additional code, and information redundancy, as used for example by ECC in terms of dynamically assignable additional parity bits.

to utilize for correction of errors, we compute a measure that expresses the cost/benefit relation between time granted for correction (i.e., stabilization) and probability of correct system functioning (i.e., system availability). We call this measure *limiting window availability* (*LWA*). It is an extension to the *instantaneous window availability* measure presented in [10].

*LWA* relates the (feasible) amount of time a system is granted to deliver correct system service to its environment to the *probability mass increase* for doing so: having more time to deliver correct system service allows for the successful masking of otherwise observable error or failure scenarios with a higher probability. Calculating this time/probability relation allows for finding those favorable parameter setting that feature the least amount of time required for a maximal probability increase. Furthermore, *LWA* allows to compare different system implementations and select the most suited ones.

The contribution of our work is 1) the definition of limiting window availability, 2) the introduction of a method for calculating limiting window availability by means of Markov chains, and 3) an abstraction-based method for achieving state space reduction of those Markov chains.

## II. Related Work

In this section, we review relevant notions of self-stabilization and associated aspects. Then, we discuss previous work, related to the practical as well as to the theoretical background for using Markov chains for modeling system behavior of distributed systems as required in the scope of this paper.

### A. Self-Stabilization

We use self-stabilizing systems examplarily for non-masking systems. They are suitable for motivating our approach as they primarily use time as redundancy resource.

This allows us to focus on the relation between the redundancy resource time and fault tolerance. In [3], self-stabilization is defined by means of closure and convergence properties:

**Definition 1 (*Self-Stabilization [3]*)**

*A system is self-stabilizing wrt. a safety predicate $\mathcal{P}$ iff:*

1) *Starting from any state, it is guaranteed that the system will eventually reach a state that satisfies the safety predicate $\mathcal{P}$ (*convergence property*), provided that no fault happens.*
2) *Given that the system satisfies the safety predicate, it is guaranteed to stay in a state that satisfies the safety predicate $\mathcal{P}$ (*closure property*), provided that no fault happens.*

Common to the fault models mostly used in the scope of self-stabilizing systems are the following assumptions: Code and constants cannot be corrupted by faults but any program variable can. Faults are assumed to be transient, i.e., non-permanent in nature. Faults can happen anytime and the number of faults though, is not restricted. Clearly, when the temporal separation between faults is large enough wrt. a certain self-stabilizing system, it can stabilize. We adopt the above fault model with the exception that we restrict faults 1) to only happen when a process executes a computational step and 2) to only impact (but arbitrarily alter) the variables of this particular process.

There exist several notions of self-stabilization, some of which are determined by the choice of the scheduler out of several scheduler classes. A scheduler "drives system execution" by selecting sets of processes allowed to take the next computational step. For our analysis, since we model self-stabilizing systems in terms of Markov chains and computational steps as state transitions, we adopt a probabilistic scheduler. A probabilistic scheduler selects any process with a particular, non-zero probability. As a consequence, the proper stabilization notion of our systems under consideration is *probabilistic self-stabilization* as categorized by Devismes et al. [2] in contrast to *deterministic self-stabilization* assumed in the earlier discussions. Probabilistic self-stabilization guarantees system convergence with probability 1, only.

### B. Towards Limiting Window Availability

In [9], a Monte Carlo simulation of self-stabilizing distributed algorithms was introduced that calculated a measure relating the amount of time spent for stabilization to the gain of probability that a fault could be masked (i.e., the increase in system availability). The analytic framework of this relation, presented in [10], was supported by the notion of *instantaneous window availability* as a suitable measure. The term *instantaneous* implicitly assumes that the self-stabilizing system is granted *a sufficient but still finite number of execution steps* to settle.

Yet, we extend this notion by transiting from a finite number of execution steps as settling phase towards the system's steady state distribution (which holds in the limit). This extension allows us to more easily analyze a substract system avoiding quite complicated analytic terms or plain simulation. Computing the limiting window availability (in contrast to instantaneous window availability) uses the steady state probability distribution of the system under consideration over the state space as initial system state distribution for the analysis. From there, *LWA* allows to "observes" the convergence behavior towards the states satisfying the system's safety predicate *time-stepwise*.

### C. Towards Masking Fault Tolerance

The gradual composition of a fault intolerant algorithm via a non-masking functional equivalent towards a fully masking tolerant solution (with respect to a given fault model) has been discussed by Arora and Kulkarni [1], [5], [6], [7]. We apply their method in our example: a non-stabilizing broadcast algorithm that is amended with a self-stabilizing spanning tree algorithm to provide a self-stabilizing broadcast algorithm (cf. the example algorithm in Figures 2 and 3). Furthermore, Arora and Kulkarni based their approach on the distinction between intolerant, non-masking fault-tolerant, and masking fault-tolerant systems. We propose a bridge over the gap between non-masking systems and their masking functional equivalents, using time redundancy.

### D. Markov Chains

Over the past decades, in the areas of reliability engineering, be it software engineering or integrated circuit engineering, three model types have proven to be practical in general. Reliability block diagrams (RBD), fault trees, and Markov chains are among the options mostly used to quantify a system's fault tolerance. As self-stabilizing systems are inherently prone to fault propagation, the system components are not independent of each other. Therefore, RBDs do not suffice here. Fault trees comply with the requirements neither, as they cannot cope with failures and repairs (nor errors and correction). Thus, we model the reachable system states as states of a Markov chain and calculate the transitions in between. The system behavior model results in an ergodic Markov chain.

The reduction of Markov chains has been discussed broadly. Here, we review a selection of relevant papers. In [12] Markov chains representing the behavior of production systems are reduced by identifying and excluding unreachable states. Qureshi et al. [11] reduce Markov chains that model the behavior of wireless local area networks and wireless sensor networks to cope with the exponential state space explosion problem. This reduction is solely based on the Hamiltonian circuit property,[2] a requirement we do not see as granted in our setting. In [14] a bounding reduction technique for finite discrete-time Markov chains is introduced. Although this work is valuable for the decomposition of Markov chains in order to cope with state space explosion, the reduction we propose is an *exact abstraction* and not a decomposition. Another valuable approach to tackle state space explosion has been presented in [8], but it also focuses on decomposition but not on exact abstraction.

## III. Definitions

After the basic system specifications in Section III-A, we state the definition of *limiting availability* in Section III-B and define *limiting window availability* ($LWA$) and related notions required in the remainder of the paper in Section III-C.

### A. Basic System Definitions

A system $\Pi$ comprises $n$ processes $\pi_i$ with $\Pi = \{\pi_1, \ldots, \pi_n\}$. Each process $\pi_i$ has a register $reg_i$ to store information in. The *configuration* $c_t$ of a system at time $t$ is the particular instantiation of values of $reg_i, i = 1, \ldots, n$, at time $t$. We assume a global safety predicate $\mathcal{P} \equiv \mathcal{P}_1 \wedge \ldots \wedge \mathcal{P}_n$ that comprises $n$ local predicates. Each local predicate $\mathcal{P}_i$ specifies the set of legal configurations of the register of its corresponding process $\pi_i$. We say the system is *operational* or in a *legal state* (or *legal configuration*) at time $t$, if the current system configuration satisfies the global predicate, denoted by $c_t \models \mathcal{P}$. We abstract the state of a process that satisfies its predicate with 0 (labeled $true$ in an algorithm) and the dissatisfaction of its predicate with 2 (labeled $false$ in an algorithm). In the next section, we present a self-stabilizing broadcast algorithm that is based on a three-value system. The third value which represents *don't know* is abbreviated by 1 (and by "$dk$" in an algorithm).

[2]The Hamiltonian circuit property states that there exists a path which visits every vertex exactly once.

In Section IV-C, we group configurations into so-called *compounds* $\mathbb{0}, \ldots, \mathbb{n}$. In our example, a compound contains the same number of processes which dissatisfy their local predicates as given by the compound's name numerical value. Hence, compound $\mathbb{0}$ contains those configurations that satisfy the global predicate. Compound $\mathbb{n}$ comprises all configurations where each process dissatisfies its local predicate. Consequently, the number of possible compounds is equal to the number of processes in the system plus one. We denote an execution step as a transition from $c_t$ to $c_{t+1}$ by $\overrightarrow{c_t, c_{t+1}}$.

As indicated above, we use a probabilistic scheduler that selects prior to each computation step one process to execute. We consider the probability $p > 0$ that the executing process $\pi_i$ works according to its specification (i.e., correctly). $q = 1-p$ is the probability that the executing process suffers from a fault, thus, makes a *fault step* and stores $false$ in its register $reg_i$. We also refer to process $\pi_i$ with $\pi(reg_i)$ as the process containing register $reg_i$.

### B. Limiting Availability

Limiting availability can be used to analyze systems that are supposed to run infinitely long. We denote the (instantaneous) availability of a system (which is the probability that the system works according to its specification) at time $t$ with $A_t$.

**Definition 2 (*Limiting Availability*)**

Limiting availability *is the the probability that a system works according to its specification at time instant* $t$ *as* $t$ *approaches infinity:* $A_{lim} := \lim_{t \to \infty} A_t$.

### C. Limiting Window Availability, its Vector, and its Gradient

A self-stabilizing system, even if not available at a certain point in time, might become available some time later. Suppose the system is not available, then how long does it take until it is operational again? We rephrase this question as: What is the probability increase per time step that we wait for the system to work according to its specification again? To give a quantitative answer to this question, we require a proper measure.

**Definition 3 (*Limiting Window Availability*)**

*Assume that at time* $t = 0$*, an initial system state distribution holds that corresponds to the steady state distribution of a system. Then,* Limiting Window Availability of window size $i$ (of this system), *denoted by* $l_i$, $i \geq 0$, *is the probability that the system has at least once reached a state satisfying* $\mathcal{P}$ *within the following* $i$ *time steps:*

$$l_i := \sum_{j=0}^{i} p(\forall k, 0 \leq k < i : c_k \not\models \mathcal{P} \wedge c_i \models \mathcal{P}),$$

$i$ *is called* window size.

Note, that $l_0 = A_{lim}$.

**Definition 4 (*Limiting Window Availability Vector*)**

*The* limiting window availability vector of size $i$ (of a system), *denoted by* $LWA$*, is an* $i$*-dimensional vector of probabilities. The element in the* $i^{th}$ *position is the limiting*

*window availability of window size $i-1$ of that system,*
$LWA := \langle l_0, l_1, \ldots, l_{i-1} \rangle$.

The mission goal is to wait for the system to work according to its specification again. Informally speaking, once the system has reached a legal state, we do not care anymore whether the system is compromised or not during its ongoing operation for the calculation of the $LWA$. Hence, the sequence of $l_i \in LWA$ is monotonically not decreasing.

**Definition 5 (*Limiting Window Availability Gradient*)**

*The* Limiting Window Availability Gradient, *(or synonymously LWA Differential), denoted by $LWA_{grad}$, of a given limiting window availability vector of size $i$ is an $(i-1)$-dimensional vector of probabilities with*
$LWA_{grad} := \langle l_1 - l_0, \ldots, l_{i-1} - l_{i-2} \rangle$.

The elements of $LWA_{grad}$ represent the increase of probability per time step that a system will have satisfied its predicate. Note, that the gradient is undefined for LWA vectors of size 1.

We are particularly interested in the maximal elements of $LWA_{grad}$, since they present time points with the largest increase in system availability. Such a time point determines the window size that is most reasonable to grant the system for stabilization. As it turned out, systems might exhibit multiple local maxima.

### D. Fault Masker

We already introduced the *fault masker* [10], an additional software layer over a non-masking system. It provides the ability to exploit time in order to increase the degree of masking fault tolerance of the system underneath it. The fault masker, when called from an upper software layer, monitors the system for two purposes. First, it is able to detect if the system is not responding according to its specification. Secondly, in such a case, the fault masker 1) waits a time step (corresponding to an execution step), 2) retries the inquiry and 3) checks whether the new system's response is correct. After a certain number of unsuccessful "retries" (corresponding to the $LWA$ window size $m$), it reports the system failure to the upper layer.

## IV. Calculation of Limiting Window Availability

We present our approach in four steps. Each step begins with the description of the general method followed by a short example. First, we specify a system and the problem specification according to Section IV-A. Second, we construct a Markov chain that models the system behavior in Section IV-B. Third, we reduce the Markov chain losslessly in Section IV-C. Fourth, we use the reduced Markov chain to calculate the limiting window availability of window size 20 for a given system in Section IV-D.

### A. Problem Specification

The specification of a system consists of three major parts: 1) the specification of a system structure (processes and communication channels), 2) a self-stabilizing algorithm, and 3) a fault model. As we compute our measures and since a self-stabilizing system is employed, we do not need to care about an initial state. Furthermore, we use serial execution semantics (i.e., in each time step, exactly one process is elected for execution). As we have discussed earlier, we assume a probabilistic scheduler.

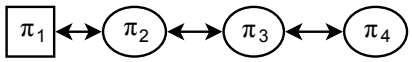


Figure 1. The Four Process Example System, $\pi_1$ being the Root Process

**Method** We specify a distributed system according to Section III-A. A self-stabilizing algorithm is executed on such a system. We use the fault model as described earlier.

**Example** The system comprises four processes $\Pi = \{\pi_1, \ldots, \pi_4\}$. In our example, the distributed algorithm requires one distinguished root process $\pi_1$. The processes are connected as shown in Figure 1. Figure 2 shows a self-stabilizing version of a simple broadcast algorithm for the root process. Non-root processes execute the algorithm shown in Figure 3.

```
1  const id := 0,
2  var reg,
3  repeat{
4     reg := true}
```

Figure 2. Broadcast Sub-Algorithm for the Root Process

```
1  const neighbors := {π_i, ...},
2  const id := min{id(π_i), ...} + 1,
3  var reg,
4  var set := {reg_i, π(reg_i) ∈ neighbors|∀i :id(π_i) =id−1}
5  repeat{
6    ¬((∃ reg_i : π(reg_i) ∈ set ∧ reg_i = false) xor
7       (∃ reg_i : π(reg_i) ∈ set ∧ reg_i = true)) →
8          reg:=dk
9  □ ∃ reg_i : π(reg_i) ∈ set ∧ reg_i = true  → reg := true
10 □ ∃ reg_i : π(reg_i) ∈ set ∧ reg_i = false → reg := false}
```

Figure 3. Broadcast Sub-Algorithm for Non-Root Processes

Each process is configured along a spanning tree algorithm. As result, each process knows its distance from the root process, stored in the constant $id$. The root process $\pi_1$, having $id = 0$, constantly writes a $true$ value into its register. Each non-root process takes those processes into account for the computation of its value, that belong to a process that is one step closer to the root process than itself. The registers $reg_i$ of those neighbors that are closer to the root process are referred to in the $set$ structure. Computing its value, a non-root process obtains one of three values. When the registers in *set* 1) contain both $true$ and $false$, or exclusively $dk$ values, the executing process stores $dk$ in its register. Otherwise, if 2) $true$ is read in a register of the *set* and $false$ is not, $true$ is stored. In case a register reads 3) $false$ but not $true$, then the executing process stores $false$ in its register. Obviously, fault propagation follows from root towards the leafs, and the structure of a spanning tree in the

initialization phase is mandatory for stabilization. We say that $\mathcal{P}_i \models true$ if $reg_i = true$.

**Theorem 1**

*The proposed broadcast algorithm is probabilistically self-stabilizing under a probabilistic scheduler.*

**Proof 1**

***Anchor:*** *In the absence of faults, the limiting probability that the root process executes is* $1$*. Then, the root process writes* $true$ *to its register and satisfies* $\mathcal{P}_1$*.*
***Step:*** *In the absence of faults, the limiting probability for each process* $\pi_i, 1 < i \leq n$*, to execute after all neighbors have already stabilized is* $1$*. Then, the process writes* $true$ *to its register and satisfies* $\mathcal{P}_i$*.*
***Closing:*** *In the absence of faults, only the correct value is stored, since the root process always executes line* $4$ *and non-root processes (eventually) always execute line* $8$ *in Figures 2 and 3 respectively. This proves* closure*. Thus, the algorithm is probabilistically self-stabilizing.*

When a transient fault perturbs the registers of the executing process, it fails to store the computed value (cf. lines $6-9$ in Figure 3) and instead, stores $false$. We abbreviate $true$ with $0$, $dk$ with $1$ and $false$ with $2$. We use the system shown in Figure 1 with $n = 4$. The system state at time $t$ is the quadruple $c_t = \langle reg_1, reg_2, reg_3, reg_4 \rangle$. Consequently, the transition probabilities between the states result in a matrix that comprises $54 \times 54$ entries. Fortunately, we can disregard those states that are unreachable: Note that the serially connected processes and the root process can only store either $true$ or $false$. Furthermore, due to the serial nature of the topology, each non-root process has exactly one successor. Thus, states in which at least one process stores a $dk$ value, are unreachable (as the $dk$ value is only required by the algorithm on non-serial topologies.) This, luckily, reduces the transition table of the example to a $16 \times 16$ matrix.

### B. Markov Model

**Method** We use the resulting $16$ relevant system states as vertices of a Markov chain. Then, we calculate the transition probabilities between each pair of states.

**Example** The probabilistic scheduler is assumed to potentially select every process prior to every computation step with an equal probability of $p_{exec}(\pi_i) = \frac{1}{n}$. We abbreviate $p_{exec}(\pi_i)$ with $e_i$ hereafter. Transient faults are assumed to occur with a probability of $q = 1 - p$. In the example, we set $q = 0.01$. A process that is working correctly will fail to compute the correct value for its register, if it is not provided with correct information by the neighbors' registers specified in $set$ (fault propagation, i.e., faults are strictly propagated from processes closer to the root process towards processes closer to the leaf processes). The probability that a process executes and is – while executing – corrupted by a fault is $e_i \cdot q = 0.25 \cdot 0.01 = 0.0025$ (analogously defined for the counter probability $p$ that no corruption occurs).

The transition probabilities can be calculated as shown in Tables II - IV. In the tables, we group the states into compounds $\mathbb{0}$ to $\mathbb{4}$ as described in Section III-A. Due to serial execution semantics, states from group $\mathbb{0}$ can only reach states from group $\mathbb{0}$ and $\mathbb{1}$ with one computation step, states from group $\mathbb{1}$ can only reach states from groups $\mathbb{0}$, $\mathbb{1}$ and $\mathbb{2}$, and so on. Table II shows all transitions from groups $\mathbb{0}$, $\mathbb{1}$ and $\mathbb{2}$ into groups $\mathbb{0}$ and $\mathbb{1}$. Table III shows all transitions from groups $\mathbb{1}$, $\mathbb{2}$ and $\mathbb{3}$ into group $\mathbb{2}$. Table IV, finally, shows the remaining transitions. We calculate the steady state probability distribution of this Markov chain as described in [13, p.351] and present the results in Table V. As we are interested in long-running system services, the steady state distribution as the initial distribution is required for the calculation of $LWA$. By definition, once the system works according to its specification, the mission goal is accomplished regardless whether the system is perturbed by faults afterwards or not. Hence, we design the state $\langle 0,0,0,0 \rangle$ as a sink that cannot be left once reached. For doing so, the transition probability $p(\overrightarrow{\langle 0,0,0,0 \rangle, \langle 0,0,0,0 \rangle})$ is set to 1. The probability mass belonging to state $\langle 0,0,0,0 \rangle$ after $i$ iterations then corresponds to the limiting window availability $l_i$. The $LWA$ vector with a maximal limiting availability window size of 5 is shown in Table VI.

| Compound | State | Steady State Probability |
|---|---|---|
| $\mathbb{0}$ | $\langle 0,0,0,0 \rangle$ | 0.936254913358677 |
| $\mathbb{1}$ | $\langle 0,0,0,2 \rangle$ | 0.020767040703947 |
| $\mathbb{1}$ | $\langle 0,0,2,0 \rangle$ | 0.006443085000445 |
| $\mathbb{1}$ | $\langle 0,2,0,0 \rangle$ | 0.005896554367512 |
| $\mathbb{1}$ | $\langle 2,0,0,0 \rangle$ | 0.004721801275921 |
| $\mathbb{2}$ | $\langle 0,0,2,2 \rangle$ | 0.011734460936930 |
| $\mathbb{2}$ | $\langle 0,2,0,2 \rangle$ | 0.000103249069863 |
| $\mathbb{2}$ | $\langle 0,2,2,0 \rangle$ | 0.003596242185866 |
| $\mathbb{2}$ | $\langle 2,0,0,2 \rangle$ | 0.000101514623954 |
| $\mathbb{2}$ | $\langle 2,0,2,0 \rangle$ | 0.000028052478081 |
| $\mathbb{2}$ | $\langle 2,2,0,0 \rangle$ | 0.002411422793886 |
| $\mathbb{3}$ | $\langle 0,2,2,2 \rangle$ | 0.005204454376759 |
| $\mathbb{3}$ | $\langle 2,0,2,2 \rangle$ | 0.000049131622044 |
| $\mathbb{3}$ | $\langle 2,2,0,2 \rangle$ | 0.000042503806239 |
| $\mathbb{3}$ | $\langle 2,2,2,0 \rangle$ | 0.001243938539611 |
| $\mathbb{4}$ | $\langle 2,2,2,2 \rangle$ | 0.001401634860264 |

Table V
STEADY STATE PROBABILITY DISTRIBUTION

| $l_0$ | $l_1$ | $l_2$ | $l_3$ | $l_4$ | $l_5$ |
|---|---|---|---|---|---|
| 0.93626 | 0.94562 | 0.95266 | 0.95824 | 0.96282 | 0.96668 |

Table VI
THE $LWA$ OF THE EXAMPLE SYSTEM WITH MAXIMAL WINDOW SIZE 5

Next, we reduce the Markov chain in order to investigate the relation between the compounds.

### C. Preparations for the Reduced Markov Chain

The purpose to reduce a Markov chain is to reduce the number of states (and transitions) and to be able to observe the relation between compounds of interest. There might be multiple objectives of interest. One example that we show in this paper is the probability distribution of the compounds as specified in Tables II - IV. In these tables, we have grouped the states according to the number of processes that fail to satisfy their specification. Hence, the number of each compound corresponds to the minimal number of execution

| | ↓from/to→ | $\mathbb{0}$ | $\mathbb{1}$ | | | |
|---|---|---|---|---|---|---|
| | | $\langle 0,0,0,0\rangle$ | $\langle 0,0,0,2\rangle$ | $\langle 0,0,2,0\rangle$ | $\langle 0,2,0,0\rangle$ | $\langle 2,0,0,0\rangle$ |
| $\mathbb{0}$ | $\langle 0,0,0,0\rangle$ | $p(e_1+e_2+e_3+e_4)$ | $qe_4$ | $qe_3$ | $qe_2$ | $qe_1$ |
| $\mathbb{1}$ | $\langle 0,0,0,2\rangle$ | $pe_4$ | $p(e_1+e_2+e_3)+qe_4$ | | | |
| | $\langle 0,0,2,0\rangle$ | $pe_3$ | | $p(e_1+e_2)+qe_3$ | | |
| | $\langle 0,2,0,0\rangle$ | $pe_2$ | | | $p(e_1+e_4)+qe_2$ | |
| | $\langle 2,0,0,0\rangle$ | $pe_1$ | | | | $p(e_3+e_4)+qe_1$ |
| $\mathbb{2}$ | $\langle 0,0,2,2\rangle$ | | $pe_3$ | | | |
| | $\langle 0,2,0,2\rangle$ | | $pe_2$ | | $pe_4$ | |
| | $\langle 0,2,2,0\rangle$ | | | $pe_2$ | | |
| | $\langle 2,0,0,2\rangle$ | | $pe_1$ | | | $pe_4$ |
| | $\langle 2,0,2,0\rangle$ | | | $pe_1$ | | $pe_3$ |
| | $\langle 2,2,0,0\rangle$ | | | | $pe_1$ | |

Table II
TRANSITIONS GROUPED BY NUMBER OF OPERATIONAL PROCESSES 1/3

| | ↓from/to→ | $\mathbb{2}$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | $\langle 0,0,2,2\rangle$ | $\langle 0,2,0,2\rangle$ | $\langle 0,2,2,0\rangle$ | $\langle 2,0,0,2\rangle$ | $\langle 2,0,2,0\rangle$ | $\langle 2,2,0,0\rangle$ |
| $\mathbb{1}$ | $\langle 0,0,0,2\rangle$ | $qe_3$ | $qe_2$ | | $qe_1$ | | |
| | $\langle 0,0,2,0\rangle$ | $e_4$ | | $qe_2$ | | $qe_1$ | |
| | $\langle 0,2,0,0\rangle$ | | $qe_4$ | $e_3$ | | | $qe_1$ |
| | $\langle 2,0,0,0\rangle$ | | | | $qe_4$ | $qe_3$ | $e_2$ |
| $\mathbb{2}$ | $\langle 0,0,2,2\rangle$ | $p(e_1+e_2)+qe_3+e_4$ | | | | | |
| | $\langle 0,2,0,2\rangle$ | | $pe_1+q(e_2+e_4)$ | | | | |
| | $\langle 0,2,2,0\rangle$ | | | $pe_1+qe_2+e_3$ | | | |
| | $\langle 2,0,0,2\rangle$ | | | | $q(e_1+e_4)+pe_3$ | | |
| | $\langle 2,0,2,0\rangle$ | | | | | $q(e_1+e_3)$ | |
| | $\langle 2,2,0,0\rangle$ | | | | | | $qe_1+e_2+pe_4$ |
| $\mathbb{3}$ | $\langle 0,2,2,2\rangle$ | $pe_2$ | | | | | |
| | $\langle 2,0,2,2\rangle$ | $pe_1$ | | | $pe_3$ | | |
| | $\langle 2,2,0,2\rangle$ | | $pe_1$ | | | | $pe_4$ |
| | $\langle 2,2,2,0\rangle$ | | $pe_1$ | | | | |

Table III
TRANSITIONS GROUPED BY NUMBER OF OPERATIONAL PROCESSES 2/3

| | ↓from/to→ | $\mathbb{3}$ | | | | $\mathbb{4}$ |
|---|---|---|---|---|---|---|
| | | $\langle 0,2,2,2\rangle$ | $\langle 2,0,2,2\rangle$ | $\langle 2,2,0,2\rangle$ | $\langle 2,2,2,0\rangle$ | $\langle 2,2,2,2\rangle$ |
| $\mathbb{2}$ | $\langle 0,0,2,2\rangle$ | $qe_2$ | $qe_1$ | | | |
| | $\langle 0,2,0,2\rangle$ | $e_3$ | | $qe_1$ | | |
| | $\langle 0,2,2,0\rangle$ | $e_4$ | | | $qe_1$ | |
| | $\langle 2,0,0,2\rangle$ | | $qe_3$ | $e_2$ | | |
| | $\langle 2,0,2,0\rangle$ | | $e_4$ | | $e_2$ | |
| | $\langle 2,2,0,0\rangle$ | | | $qe_4$ | $e_3$ | |
| $\mathbb{3}$ | $\langle 0,2,2,2\rangle$ | $pe_1+qe_2+e_3+e_4$ | | | | $qe_1$ |
| | $\langle 2,0,2,2\rangle$ | | $q(e_1+e_3)+e_4$ | | | $e_2$ |
| | $\langle 2,2,0,2\rangle$ | | | $q(e_1+e_4)+e_2$ | | $e_3$ |
| | $\langle 2,2,2,0\rangle$ | | | | $qe_1+e_2+e_3$ | $e_4$ |
| $\mathbb{4}$ | $\langle 2,2,2,2\rangle$ | $pe_1$ | | | | $qe_1+e_2+e_3+e_4$ |

Table IV
TRANSITIONS GROUPED BY NUMBER OF OPERATIONAL PROCESSES 3/3

steps the system must take to stabilize. Another goal could be to group those states together where a specific node fails. Doing so would allow to find dependability bottlenecks in the system design.

**Method** Reducing the Markov chain consists of three steps: 1) calculation of the steady state probability for each state (cf. Table V), 2) building subsets of states (*compounds*) as desired, and 3) calculating the transition probabilities between the subsets.

**Example** We label the Markov chain presented in Tables II - IV as $M$. After calculating the steady state probability distribution of $M$ we build a new Markov chain $\mathbb{M}$ in which a state represents a compound (of states) of $M$. In our example, we identify the compounds $\mathbb{0}$, $\mathbb{1}$, $\mathbb{2}$, $\mathbb{3}$ and $\mathbb{4}$. Third, we calculate the transition probabilities between the states of $\mathbb{M}$, that is, between the compounds.

*Steady State Probability Distribution:* Table V shows the steady state probability distribution derived from $M$.

*Vertex Abstraction:* We divide the states from $M$ into compounds. We build a new Markov chain $\mathbb{M}$ in which each compound is represented by a single state. The motivation is to check, whether probability mass is "withheld" in a compound. If so, using additional time steps for stabilization is reasonable. The Markov chain $\mathbb{M}$, in our case, consists of $n+1$ states. The initial probability of a compound state in $\mathbb{M}$ is set to the sum of the steady state probabilities of the states it comprises.

*Transition Abstraction:* Similar to building compound states, we group all those transitions together that have in common i) a source in the same compound and ii) a target

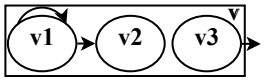


Figure 4. Example Reduction: Compound $\mathbb{v}$ of $v_1$, $v_2$ and $v_3$

in the same compound. An important part is to correctly consider the weighting of the transitions into the transition abstraction.

To differentiate between source and target state of a transition, we label the source state $v$ and the target state $w$. The reduced transitions are calculated using the following formula:

$$p(\overrightarrow{\mathbb{v},\mathbb{w}}) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} p(\overrightarrow{v_i,w_j}) \cdot p(v_i)}{\sum_{i=0}^{n} p(v_i)} \quad (1)$$

with $\mathbb{v},\mathbb{w} \in \{\mathbb{0},\mathbb{1},\mathbb{2},\mathbb{3},\mathbb{4}\}$ in our example.

Here, the number of transitions with the source state in compound $\mathbb{v}$ and the target state in compound $\mathbb{w}$ is $n$. The number of transitions with a target state in compound $\mathbb{w}$ is $m$. The transition probability from one state of compound $\mathbb{v}$ to one state in compound $\mathbb{w}$ is denoted by $p(\overrightarrow{v_i,w_j})$. Its particular weight (i.e., the probability mass in the particular state) is denoted by $p(v_i)$. Tables II - IV show the source states $\mathbb{v}$ as rows and the target states $\mathbb{w}$ as columns, both grouped by their particular number of operational processes.

When folding transitions, three cases can occur: 1) a transition from a state within a compound to itself, 2) a transition from a state within a compound to another state of the compound, and 3) a transition from one state within a compound to another process outside the compound as depicted in Figure 4. Our example does not exhibit compounds of all three cases that possibly can occur in the reduction. As we employ only serial execution semantics, the system state can change at most in one digit per execution step. The transition $\overrightarrow{v_1,v_2}$ can hence not occur as two states that have the same number of functional processes must at least differ in two digits. To show all possibilities, we briefly discuss the example depicted in Figure 4. We reduce vertices $v_1$, $v_2$ and $v_3$ into one compound $\mathbb{v}$ and calculate the transition probability $\overrightarrow{\mathbb{v},\mathbb{v}}$ exemplarily:

$$p(\overrightarrow{\mathbb{v},\mathbb{v}}) = \frac{p(\overrightarrow{v1,v1}) \cdot p(v1) + p(\overrightarrow{v1,v2}) \cdot p(v1)}{p(v1) + p(v2) + p(v3)} \quad (2)$$

We return to the reduction of $M$ to $\mathbb{M}$ and exemplarily calculate the transition probability $p(\overrightarrow{\mathbb{1},\mathbb{1}})$. The transition probabilities we need are given in Table II. The steady state probabilities are given in Table V. Then, $p(\overrightarrow{\mathbb{1},\mathbb{1}})$ calculates as follows:

$$\begin{aligned} p(\overrightarrow{\mathbb{1},\mathbb{1}}) = \; & \tfrac{0.745\cdot 0.0208+0.4975\cdot 0.0064}{0.0208+0.0064+0.0059+0.0047} + \\ & \tfrac{0.4975\cdot 0.0059+0.4975\cdot 0{,}0047}{0.0208+0.0064+0.0059+0.0047} \\ & = 0.6333722949242 \end{aligned} \quad (3)$$

*Reduced Markov Chain:* As described, we require five states labeled $\mathbb{0}$ to $\mathbb{4}$. Then, we calculate their initial state probabilities as sums of the respective probabilities of the states the compounds comprise (shown in Table VII), and the transition probabilities (shown in Table VIII). Note, that the information which is abstracted away only concerns transition probabilities *within* the compounds. As a consequence, the abstracted Markov chain $\mathbb{M}$ delivers *exact results* of the probability distribution *between* the compounds over time.

| Vertex | Initial Probability |
|---|---|
| $\mathbb{0}$ | 0.936254913358678 |
| $\mathbb{1}$ | 0.037828481347825 |
| $\mathbb{2}$ | 0.017974942088580 |
| $\mathbb{3}$ | 0.006540028344653 |
| $\mathbb{4}$ | 0.001401634860264 |

Table VII
INITIAL PROBABILITY DISTRIBUTION FOR $\mathbb{M}$

| ↓from/to→ | $\mathbb{0}$ | $\mathbb{1}$ | $\mathbb{2}$ | $\mathbb{3}$ | $\mathbb{4}$ |
|---|---|---|---|---|---|
| $\mathbb{0}$ | 1 | 0 | 0 | 0 | 0 |
| $\mathbb{1}$ | 0.2475 | 0.6334 | 0.1191 | 0 | 0 |
| $\mathbb{2}$ | 0 | 0.2507 | 0.6580 | 0.0913 | 0 |
| $\mathbb{3}$ | 0 | 0 | 0.2510 | 0.6960 | 0.0530 |
| $\mathbb{4}$ | 0 | 0 | 0 | 0.2475 | 0.7525 |

Table VIII
TRANSITION PROBABILITIES OF THE REDUCED MARKOV CHAIN

### D. The Limiting Window Availability Gradient

In the previous section, we have shown how to calculate $LWA$ by an example using Markov chain $M$. The results are shown in the column labeled $\mathbb{0}$ (representing a functioning system) of Table IX for window sizes of 0 to 20. The $LWA$ results for the reduced example $\mathbb{M}$ are also given in Table IX: here, the different columns show the probabilities on a compound-basis. After calculating the $LWA$ and the

| | $\mathbb{0}$ | $\mathbb{1}$ | $\mathbb{2}$ | $\mathbb{3}$ | $\mathbb{4}$ |
|---|---|---|---|---|---|
| 0 | 0.93626 | 0.03783 | 0.01798 | 0.00654 | 0.00140 |
| 1 | 0.94562 | 0.02847 | 0.01798 | 0.00654 | 0.00140 |
| 2 | 0.95266 | 0.02254 | 0.01686 | 0.00654 | 0.00140 |
| 3 | 0.95824 | 0.01850 | 0.01542 | 0.00644 | 0.00140 |
| 4 | 0.96282 | 0.01558 | 0.01397 | 0.00624 | 0.00140 |
| 5 | 0.96668 | 0.01337 | 0.01261 | 0.00596 | 0.00138 |
| 6 | 0.96999 | 0.01163 | 0.01139 | 0.00564 | 0.00136 |
| 7 | 0.97286 | 0.01022 | 0.01029 | 0.00530 | 0.00132 |
| 8 | 0.97539 | 0.00905 | 0.00932 | 0.00496 | 0.00127 |
| 9 | 0.97764 | 0.00807 | 0.00846 | 0.00462 | 0.00122 |
| 10 | 0.97963 | 0.00723 | 0.00768 | 0.00429 | 0.00116 |
| 11 | 0.98142 | 0.00651 | 0.00699 | 0.00397 | 0.00110 |
| 12 | 0.98303 | 0.00588 | 0.00637 | 0.00368 | 0.00104 |
| 13 | 0.98449 | 0.00532 | 0.00582 | 0.00340 | 0.00098 |
| 14 | 0.98580 | 0.00483 | 0.00531 | 0.00314 | 0.00092 |
| 15 | 0.98700 | 0.00439 | 0.00486 | 0.00290 | 0.00086 |
| 16 | 0.98808 | 0.00400 | 0.00445 | 0.00267 | 0.00080 |
| 17 | 0.98907 | 0.00365 | 0.00407 | 0.00246 | 0.00074 |
| 18 | 0.98998 | 0.00333 | 0.00373 | 0.00227 | 0.00069 |
| 19 | 0.99080 | 0.00305 | 0.00342 | 0.00209 | 0.00064 |
| 20 | 0.99156 | 0.00279 | 0.00314 | 0.00193 | 0.00059 |

Table IX
PROBABILITY DISTRIBUTION DEVELOPMENT OF $\mathbb{M}$ ON A COMPOUND-BASIS WITH INCREASING WINDOW SIZE

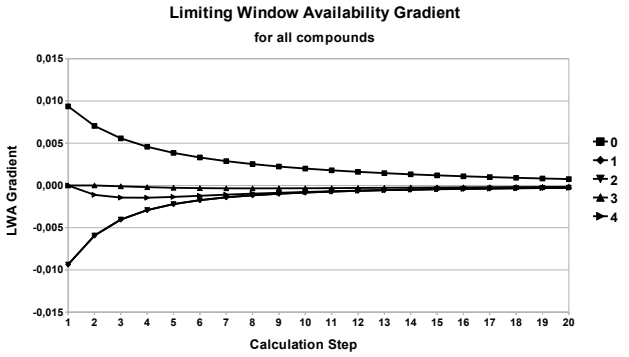


Figure 5. $LWA_{grad}$ of the Example

computation of compounds in our interest, we calculate the $LWA_{grad}$. Results are presented in Figure 5 (discrete results are connected by straight lines for better visibility).

### E. Discussion

An interesting characteristic of the example we used is given by the columns $\mathbb{1}$ to $\mathbb{4}$ in Table IX: note that the probability mass in these states is constant for $o$ steps where $o$ is the label of the compound (the last column shows one step to many due to rounding). This means, that the probability mass income equals the probability mass outgo for the first $o$ steps. This fact relies on the steady state probability distribution which is used as initial distribution, as well as on the transition probabilities. We call the effect that compounds do not loose probability mass over the first steps *probability mass retention*. When calculating larger and non-serial systems (with trees and cycles), this effect is even stronger. The probability mass in states other than $\mathbb{0}$ can temporarily even increase. The evolution of the probability mass within the compounds is a valuable indicator whether stabilization can be expected *soon* or whether the probability mass "is stuck in a compound that is *far from stabilization*." Such an analysis represents a valuable observation in the discovery of bottlenecks in the system design.

## V. Conclusion

We presented the notion of limiting window availability. We showed how to use Markov models for calculating $LWA$ and quantified the trade-off between time and an increased degree of masking fault tolerance using an example. We showed how to group states to compounds and observed the effect of probability mass retention.

Small systems, like the example we used, have a strictly monotonic $LWA_{grad}$. Larger systems do not necessarily comply with this behavior [9]. We presented an abstraction method capable of computing the precise $LWA$ of fault tolerant systems.

In the future, we will use this abstraction for the decomposition of systems (i.e., the decomposition of the Markov chains modeling the system's behavior). This will allow us to also compute the $LWA$ for larger systems, as we currently are unable to, due to the state space explosion of their respective Markov chains.

## Acknowledgments

N. Müllner was supported by the German Research Foundation (DFG) under grant DRK 1076/1 TrustSoft. O. Theel was supported by the DFG under grant SFB/TR 14 AVACS.

## References

[1] A. Arora and S. S. Kulkarni. Designing Masking Fault-Tolerance via Nonmasking Fault-Tolerance. *IEEE Transactions on Software Engineering*, 24(6):435–450, 1998.

[2] S. Devismes, S. Tixeuil, and M. Yamashita. Weak vs. Self vs. Probabilistic Stabilization. In *ICDCS '08: Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*, pages 681–688, Washington, DC, USA, 2008. IEEE Computer Society.

[3] S. Dolev. *Self-Stabilization*. MIT Press, Cambridge, MA, USA, 2000.

[4] F. C. Gärtner. Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments. *ACM Computing Surveys*, 31(1):1–26, 1999.

[5] S. Kulkarni and A. Arora. Automating the Addition of Fault-Tolerance. In *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFTS'2000)*, 2000.

[6] S. S. Kulkarni. *Component based design of fault-tolerance*. PhD thesis, 1999.

[7] S. S. Kulkarni and A. Arora. Compositional Design of Multitolerant Repetitive Byzantine Agreement. *Lecture Notes in Computer Science*, 1346:169–183, 1997.

[8] L. Leskelä. Computational methods for stochastic relations and markovian couplings. In *VALUETOOLS '09: Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[9] N. Müllner, A. Dhama, and O. Theel. Derivation of Fault Tolerance Measures of Self-Stabilizing Algorithms by Simulation. In *ANSS '08: Proceedings of the 41st Annual Symposium on Simulation*, pages 183–192. IEEE Computer Society Press, Apr. 2008.

[10] N. Müllner, A. Dhama, and O. Theel. Deriving a Good Trade-off Between System Availability and Time Redundancy. In *Proceedings of the Symposia and Workshops on Ubiquitious, Automatic and Trusted Computing*, number E3737, pages 61–67. IEEE Computer Society Press, July 2009.

[11] H. K. Qureshi, K. Shahzad, S. A. Khayam, M. Rajarajan, and V. Rakocevic. *Complexity reduction of Markov channel models for wireless networks using graph theory*. 2008.

[12] D. Racoceanu, N. Zerhouni, and N. Addouche. Modular modeling and analysis of a distributed production system with distant specialized maintenance. In *Proc. of the 2002 IEEE International Conference on Robotics and Automation, on CD ROM*, pages 4046–4052, 2002.

[13] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Second edition, 2002.

[14] L. Truffet. Reduction techniques for discrete-time markov chains on totally ordered state space using stochastic comparisons. *Journal of Applied Probability (J. Appl. Probab.)*, 37:795–806, 2000.