Stability of Distributed Algorithms in the face of Incessant Faults

R. E. Lee DeVille and Sayan Mitra

University of Illinois at Urbana Champaign
{rdeville,mitras}@illinois.edu

Abstract. For large distributed systems built from inexpensive components, one expects to see incessant failures. This paper proposes two models for such faults and analyzes two well-known self-stabilizing algorithms under these fault models. For a small number of processes, the properties of interest are verified automatically using probabilistic model-checking tools. For a large number of processes, these properties are characterized using asymptotic bounds from a direct Markov chain analysis and approximated by numerical simulations.

1 Introduction

Self-stabilization guarantees automatic fault-tolerance: after a fault, the system may deviate from its desirable behavior for a finite amount of time, but it eventually reaches a desirable (or *legal*) state automatically. This property is particularly attractive in large distributed systems where manual failure management is impractical. Self-stabilizing algorithms for mutual exclusion, leader election, spanning tree construction and other distributed computation tasks have been extensively studied (see for example [1,2,3]). These algorithms guarantee that *once failures cease* a legal state is reached in a finite number of steps. Often the number of steps (and therefore the amount of time) required for recovery from even a single failure grows at least linearly with the size of the system, and it is assumed that no further failures occur during this period. For these algorithms to be valid, the failure probability times the number of components must be significantly less than one.

Distributed systems built from off-the shelf components and deployed in harsh environments [4,5,6] will experience frequent component failures. As a result, these systems may not experience failure-free periods which are long enough to recover completely to a legal state. In this paper, we initiate a systematic investigation of distributed algorithms in the face of such *incessant failures*. Random transmission and link failures have been studied recently in [7,8]. Also, fault-tolerance with respect to a locally bounded number of faults (as opposed to globally bounded) has been studied in [9]. To the best of our knowledge, however, our work is the first attempt at investigating distributed systems under random failures with bounded failure rate.

In the face of incessant faults, the system will not stabilize, i.e. enter a legal state and remain there. However, not all illegal states are equivalent; some are worse than others. For a given distributed system with state space X, define $Q : X \to \mathbb{R}$, which determines "how far" each state is from legal: legal states are those for which $Q = 0$, and high Q states are ones which are particularly bad. We show below how to use a Markov chain to model a distributed system under incessant faults. The Markov chain (under weak assumptions) will have a unique invariant measure π which leads to natural global measures of performance (e.g. the mean of Q, $Prob(Q > k)$ for some k, etc.). If one knows π, then one can compute all of these, but in general, it is difficult to compute π explicitly.

Two observations emerge from the results in this paper. First, the measure π depends significantly on the types of allowed faults, i.e. changing the fault model leads to a significantly different "typical state" during the evolution of a system. Secondly, since a post-fault state depends on the pre-fault state, which itself depends on the output of the algorithm after a random number of steps, etc., *one can no longer assume that the stabilization process and the faults are decoupled.* That is, when the faults occur on a timescale comparable to (or shorter than) the stabilization time, the system and the faults interact in complicated ways. This leads to some surprising and non-intuitive phenomena and requires a novel approach to the analysis of such systems.

In this paper, we take a first step towards development of this general theory. Specifically, we do the following:

We define two incessant fault models, the *update (U)* fault and the *sleep-update (S/U)* fault. Roughly, an update fault only occurs when a process attempts to update its state; a sleep-update fault can occur at any time. Each type of fault transforms a given stochastic automaton model of a fault-free distributed algorithm into a new stochastic automaton with additional probabilistic transitions. In particular, for synchronous distributed algorithms, the resulting fault-transformed system is a Markov chain. Next, we analyze Dijkstra's self-stabilizing token ring algorithm (TR) [10] and a randomized graph coloring algorithm (GC) from [11] under incessant faults. First, for a small number ($N \approx 7$) of participating processes, we verify quantitative properties of these algorithms using probabilistic model checkers [12,13]. This analysis is automatic and provides exact values for quantitative properties, however, owing to the state space explosion it does not scale well with N. Second, we use techniques from probability theory to analyze the underlying Markov processes of TR. The insight gained from the Markov chain analysis allows us to understand how TR tends to fail under the U and S/U fault models, and this naturally leads to a modified version, namely TR2, which, as we show, performs significantly better under incessant faults. Finally, we perform simulations of both algorithms for large N to give an idea how several performance metrics depend on parameters.

The results of this paper suggest performing an analogous analysis of many well-known self-stabilizing algorithms for routing, maximal independent set, leader election [3], etc. We expect that the specific phenomena observed here will be representative of a wide range of models, and analysis along these lines will be very useful in developing new algorithms which are robust to incessant faults. Moreover, the scaling problems we encounter with standard model checkers suggests the exploration of parallelizable statistical tools such as [14,15], and further emphasizes the need for new tools and techniques.

2 Background and Fault Models

There are several formalisms for compositionally specifying randomized distributed algorithms (see, for example [16,17,18,19] and the references therein). We describe fault-prone algorithms as *stochastic automata*, which are simplified versions of the Probabilistic I/O Automata of [20].

We denote the set of probability distributions over a set S by $\mathcal{P}(S)$. In describing the states of processors in a distributed system, we find it convenient to use a variable structure. Let X be a set of variables. Each variable $x \in X$, is associated with a *type*, denoted by *type(x)*, which is the set of values that x may take. A *valuation* of the variables in X is a function that associates each $x \in X$ with a value in *type*(x). A particular valuation for a set of variables X is written as $\mathbf{x}$, and the value of an individual variable x is denoted by $\mathbf{x}.x$. The set of all valuations of X is denoted by $Val(X)$. For a valuation $\mathbf{x} \in Val(X)$, the restriction of $\mathbf{x}$ to $Y \subseteq X$ is denoted by $\mathbf{x} \lceil Y$. For a probability distribution $\mu \in \mathcal{P}(Val(X))$ the marginal distribution of μ on $Y \subseteq X$ is denoted by $\mu \lceil Y$.

2.1 Stochastic Automata

Definition 1. *A* Stochastic Automaton (SA) $\mathcal{A}$ *is a 4-tuple* $(X, U, \bar{\mu}, \rightarrow)$*, where (1)* X *is a set of* state variables*,* $Val(X)$ *is called the set of* states*, (2)* U *is a set of* input variables*, (3)* $\bar{\mu} \subseteq \mathcal{P}(Val(X))$ *is an* initial distribution *on states, and (4)* $\rightarrow \subseteq Val(U) \times Val(X) \times \mathcal{P}(Val(X))$ *is a set of* probabilistic transitions*, satisfying:* **(D)** *for each* $\mathbf{u} \in Val(U), \mathbf{x} \in Val(X)$*, there exists* $\mu \in \mathcal{P}(Val(X))$ *such that* $(\mathbf{u}, \mathbf{x}, \mu) \in \rightarrow$*.*

The input variables provide a mechanism for modeling inter-automata communication. For example, the state variable of one automaton may act as inputs to others. In a synchronous distributed system, this variable-based communication can be implemented by message passing. In this paper, we assume existence of unique initial distribution of SA for the sake of convenience; the metrics we analyze in Sections 3 and 4 are in fact independent of the initial distributions.

Notation. If $(\mathbf{u}, \mathbf{x}, \mu) \in \rightarrow$, we write $(\mathbf{u}, \mathbf{x}) \rightarrow \mu$. For a SA $\mathcal{A}$ we denote its components by $U_{\mathcal{A}}, X_{\mathcal{A}}, \bar{\mu}_{\mathcal{A}}$ and $\rightarrow_{\mathcal{A}}$, respectively, and for a SA $\mathcal{A}_1$ its components are denoted by $U_1, X_1, \bar{\mu}_1$ and $\rightarrow_1$.

Informally, each execution or run of $\mathcal{A}$ is a (possibly infinite) sequence $(\mathbf{u}_0, \mathbf{x}_0)$, $(\mathbf{u}_1, \mathbf{x}_1), \ldots$. Each pair in the sequence corresponds to a probabilistic transition or a step. Throughout this paper, we use the "time" and "number of steps" synonymously. As in [20], we can define the probability measures over sets of executions of $\mathcal{A}$ by resolving the nondeterminism with a scheduler, however, for this paper we work with a restricted class of SAs which simplifies the formal framework.

$\mathcal{A}$ is said to be *finite* if $Val(X_{\mathcal{A}})$ is finite, *closed* if $U_{\mathcal{A}} = \emptyset$, *pure* if for every $\mathbf{u} \in Val(U), \mathbf{x} \in Val(X)$ there exists a *unique* μ satisfying condition **(D)**, and *deterministic* if for every $(\mathbf{u}, \mathbf{x}) \rightarrow_{\mathcal{A}} \mu$, μ is a Dirac delta distribution. A particular probabilistic transition $\mathbf{u}, \mathbf{x} \rightarrow \mu$ is said to be *active* if $\mu \neq \delta_{\mathbf{x}}$, otherwise it is *passive*. (Throughout this paper $\delta_{\mathbf{x}}$ is the Dirac measure at $\mathbf{x}$.) A state $\mathbf{x} \in Val(X)$ is said to be *stable* if for all $\mathbf{u} \in Val(U)$ there are no active transitions from $(\mathbf{u}, \mathbf{x})$. Clearly, a closed, pure SA $\mathcal{A}$ is equivalent to a discrete time Markov chain (DTMC) with state space $Val(X_{\mathcal{A}})$. Suppose $P_{\mathcal{A}}$ is the transition matrix of this equivalent Markov chain.

Example 1. The pseudocode in Figure 1 specify ordinary (*Left*) and special (*Right*) processes participating in Dijkstra's token ring [10]. The natural numbers N and K are parameters. TR_i, $0 < i < N$, is a deterministic, finite-state SA with the following components: (1) set of state variables $X_i = \{x_i\}$, where x_i is a variable of type $\{0, \ldots, K-1\}$, (2) set of input variables $U_i = \{x_{i-1}\}$, where x_{i-1} is a variable of type $\{0, \ldots, K-1\}$, (3) $\bar{\mu}$ is the uniform distribution over $Val(X_i)$, (4) for each $\mathbf{u} \in Val(U_i), \mathbf{x} \in Val(X_i), (\mathbf{u}, \mathbf{x}) \rightarrow \mu$ iff (i) $\mathbf{u}.x_{i-1} = \mathbf{x}.x_i$ and $\mu = \delta_{\mathbf{x}}$, or (ii) $\mathbf{u}.x_{i-1} \neq \mathbf{x}.x_i$ and $\mu = \delta_{\mathbf{x}'}$, where $\mathbf{x}'$ is the valuation that assigns $\mathbf{u}.x_{i-1}$ to x_i.

```
automaton TR($N$,$K$:$\mathbb{N}$,const $i \neq 0$)
variables
  state $x_i$: $\{0,\ldots,K-1\}$ := uni[$\{0,\ldots,K-1\}$]
  input $x_{i-1}$: $\{0,\ldots,K-1\}$

transitions
  pre $x_i \neq x_{i-1}$
  eff $x_i := x_{i-1}$
```

```
automaton TR($N$,$K$:$\mathbb{N}$,const $i = 0$)
variables
  state $x_0$: $\{0,\ldots,K-1\}$
        := uni[$\{0,\ldots,K-1\}$]
  input $x_{N-1}$: $\{0,\ldots,K-1\}$

transitions
  pre $x_0 = x_{N-1}$
  eff $x_0 := x_0 + 1 \mod K$
```

Fig. 1. SA models for token ring. *Left:* processes $i = 1 \ldots N-1$, *Right* process $i = 0$.

The parallel composition operation on SA is used for building models of distributed systems where several processes execute concurrently and communicate through shared input variables. Roughly, the composed SA is obtained by taking the union of the variables of the component automata and merging the transitions.

Definition 2. *Two SA are* compatible *if they have disjoint set of state variables. Given a pair of compatible SA $\mathcal{A}_1$ and $\mathcal{A}_2$ their* composition, *denoted by $\mathcal{A}_1 \| \mathcal{A}_2$, is defined as the structure $(U, X, \bar{\mu}, \rightarrow)$, where (1) $X = X_1 \cup X_2$, (2) $U = (U_1 \cup U_2) \setminus X$, (3) $\bar{\mu} = \bar{\mu}_1 \times \bar{\mu}_2$, and (4) $\rightarrow$ is defined as follows: for each $\mathbf{u} \in Val(U), \mathbf{x} \in Val(X), \mu \in \mathcal{P}(Val(X))$, $(\mathbf{u}, \mathbf{x}) \rightarrow \mu$ iff for each $i \in \{1, 2\}$ $((\mathbf{u}, \mathbf{x}) \lceil U_i, (\mathbf{u}, \mathbf{x}) \lceil X_i) \rightarrow_i \mu_i \lceil X_i$.*

It is easy to check that (finite, pure, deterministic) SA are closed under composition. The composition operation is inductively extended to multiple SA in the obvious way. A particular probabilistic transition $\mathbf{u}, \mathbf{x} \to \mu$ of the composed stochastic automaton $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$ is said to be *active with respect to* $\mathcal{A}_1$ if $\mu \lceil X_1 \neq \delta_{\mathbf{x} \lceil X_1}$.

Example 2. The overall token-ring system $\mathsf{TR}(N, K)$ is specified as composition of $\mathsf{TR}(N, K, 0) \| \ \mathsf{TR}(N, K, 1) \| \ \ldots \ \mathsf{TR}(N, K, N-1)$. We define the following functions which will be useful later:

$$token(\mathbf{x}, i) = (i = 0 \ \wedge \ \mathbf{x}.x_0 = \mathbf{x}.x_{N-1}) \vee (i \neq 0 \ \wedge \ \mathbf{x}.x_i \neq \mathbf{x}.x_{i-1})$$
$$hastoken(\mathbf{x}) = \{i \mid token(\mathbf{x}, i)\}, \ \ legal(\mathbf{x}) = (|hastokens(\mathbf{x})| = 1).$$

Here $\mathbf{x}$ is a valuation of all the variables of TR and $i \in \{0, \ldots, N-1\}$.

2.2 Incessant Fault Models

We introduce two models for random incessant faults. These faults introduce additional probabilistic transitions in a SA $\mathcal{A}$ which capture the effect of the faults on state variables of $\mathcal{A}$. Update faults capture (possibly transient) faults in the memory, disk drives, network interface cards, while sleep-update faults capture the effects of stochastic disturbances such as transient hardware faults, power surges, cosmic rays, and corruption of messages. First, we define how incessant faults transform the SA models for individual processes. In the *update(U)* fault model, whenever a SA $\mathcal{A}$ performs a computation and sets new values to its state variables, with some probability a fault occurs and the variable is set to an arbitrary value.

Definition 3. *Given a SA $\mathcal{A}$, the* U-faulty *version of $\mathcal{A}$ with rate $\epsilon \in (0, 1]$ is a SA $\mathcal{B} = (X_\mathcal{B}, U_\mathcal{B}, \bar{\mu}_\mathcal{B}, \to_\mathcal{B})$, where $U_\mathcal{B} = U_\mathcal{A}$, $X_\mathcal{B} = X_\mathcal{A}$, $\bar{\mu}_\mathcal{B} = \bar{\mu}_\mathcal{A}$, and $\to_\mathcal{B}$ is defined as follows: for every $(\mathbf{u}, \mathbf{x}) \to_\mathcal{A} \mu$ where $\mu \neq \delta_\mathbf{x}$, $(\mathbf{u}, \mathbf{x}) \to_\mathcal{B} \mu'$, where for every $\mathbf{x}' \in Val(X)$, $\mu'(\mathbf{x}')$ is defined as $(1-\epsilon)\mu(\mathbf{x}') + \frac{\epsilon}{|Val(X)|}$.*

It is clear from the above that if $\mathcal{A}$ is pure (also closed) then so is $\mathcal{B}$. Also, once $\mathcal{B}$ reaches a stable state, update faults do not occur. The above is the definition of U-faults for a single process; the faulty model for a complete distributed system is obtained by composing the transformed SA for the individual processes.

Sleep-update faults may affect the state of the system even after a stable state is reached: at every step, each state variable may be reset to an arbitrary value with some small probability.

Definition 4. *Given a SA $\mathcal{A}$, the* S/U-faulty *version of $\mathcal{A}$ with rate $\epsilon \in (0, 1]$ is a SA $\mathcal{B} = (X_\mathcal{B}, U_\mathcal{B}, \bar{\mu}_\mathcal{B}, \to_\mathcal{B})$, where $U_\mathcal{B} = U_\mathcal{A}$, $X_\mathcal{B} = X_\mathcal{A}$, $\bar{\mu}_\mathcal{B} = \bar{\mu}_\mathcal{A}$, and $\to_\mathcal{B}$ is defined as follows: for every $(\mathbf{u}, \mathbf{x}) \to_\mathcal{A} \mu$, $(\mathbf{u}, \mathbf{x}) \to_\mathcal{B} \mu'$ where for every $\mathbf{x}' \in Val(X)$, $\mu'(\mathbf{x}') = (1-\epsilon)\mu(\mathbf{x}') + \frac{\epsilon}{|Val(X)|}$.*

Example 3. The pseudocode in Figure 2 specifies the version of TR_i under S/U-faults. The parameter ϵ serves as the rate for incessant faults. STR_i, $i \neq 0$, is a SA with set of state variables, input variables, and initial distribution identical to that of TR_i of Figure 1. The specification of the version with U-faults, denoted by UTR_i, is identical to STR_i, except that the second set of probabilistic transitions is absent. The code for the transitions specify the probabilistic transitions.

```
automaton STR($N$,$K$:$\mathbb{N}$,const $i \neq 0$, $\epsilon : (0,1)$)
variables
  state $x_i$: $\{0,\ldots,K-1\}$
        := uni$[\{0,\ldots,K-1\}]$
  input $x_{i-1}$: $\{0,\ldots,K-1\}$

transitions
  pre $x_i = x_{i-1}$
  eff $x_i := x_{i-1}$ with prob $(1-\epsilon)$ +
          $k : \{0,\ldots,K-1\}$ with prob $\epsilon/K$;

  pre $x_i = x_{i-1}$
  eff $x_i := x_i$ with prob $(1-\epsilon)$ +
          $k : \{0,\ldots,K-1\}$ with prob $\epsilon/K$;
```

Fig. 2. TR with incessant sleep-update (S/U) faults.

3 Token Ring

In this section, we analyze S/U-faulty and U-faulty version of Dijkstra's token ring algorithm $\mathsf{TR}(N,K)$ (see Figure 2). First, we use probabilistic model-checking to exactly verify quantitative properties of the faulty systems. These techniques are automatic and they provide exact answers to a rich class of quantitative questions; however, they do not scale to systems with large number of processes. Our second approach analyzes the Markov chain corresponding to these systems directly; this allows us to determine bounds on the probabilities of various events. This analysis enables us to prove properties of the faulty system with arbitrarily large number of processes assuming that the fault rates are not too large; specifically, we assume that faults are rare enough so that it is unlikely to see more than one fault during any single transition, but common enough that many faults may occur during a self-stabilization process.

Analysis for small N. Probabilistic model checking tools (including PRISM [12] and MRMC [13]) can be used for verifying quantitative properties of Stochastic Automata, Markov Chains, and Markov Decision Processes. Given (a) a description of the model and (b) a property, a model checker returns true or false depending on whether the property is satisfied in all states of the system or not. For probabilistic model checking, properties may include boolean predicates as well as quantitative statements about the probability of certain states and executions (paths). These properties are described using probabilistic extensions of temporal logics such as Probabilistic Computational Tree Logic (PCTL) [12], Continuous Stochastic Logic (CSL) [21], and QuaTEx [14].

For the token ring system under U-faults and S/U-faults we are interested in the following quantitative properties:

Steady State(SS). For each $k \in \{1,\ldots,N\}$, define $SS(k)$ as the probability of the system being in any state $\mathbf{x}$ with $|hastokens(\mathbf{x})| = k$, where this probability is taken with respect to the invariant measure of the Markov chain.

Expected Stabilization Time (EST). For a state $\mathbf{y}$, define $L(\mathbf{y})$ to be the expected number of steps required to reach a state $\mathbf{x}$ satisfying $legal(\mathbf{x})$. The EST of an algorithm under a certain error model is then defined as $max_{\mathbf{y}} L(\mathbf{y})$. (N.B.: Under S/U-faults the legal states are no longer absorbing, so this quantity is more pertinent for U-faults.)

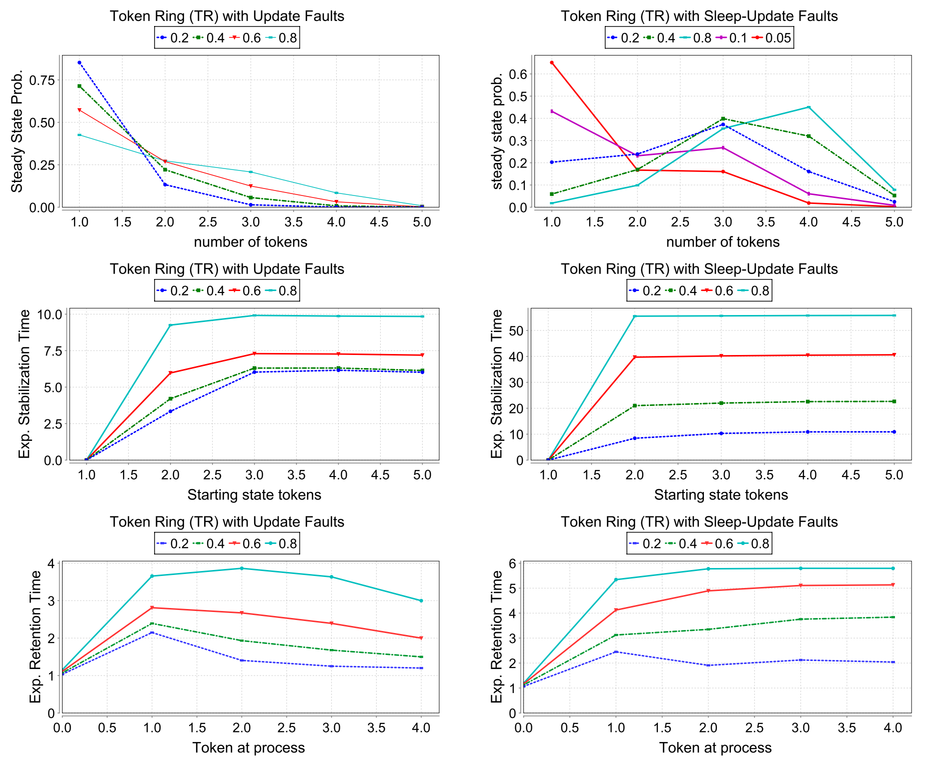

Fig. 3. *Top:* Steady state probabilities (SS) for TR under U-faults (*Left*) and S/U-faults (*Right*). *Bottom:* EST for TR under U-faults (*Left*) and S/U-faults (*Right*).

Expected Retention Time (ERT). Define $R(i)$ as the expected time to reach a state $\mathbf{y}$ with $\neg token(\mathbf{y}, i)$, given that we start at a state $\mathbf{x}$ with $token(\mathbf{x}, i)$, and define the ERT as $\max_i R(i)$.

The EST metric is useful in applications where just reaching a stable state allows a higher level application to make progress. For example, in a wireless system where graph coloring (see Section 4) is used for channel frequency assignments, reaching a stable coloring means that a node can send a packet successfully (at least for one round). We computed SS, EST, and ERT for $N = 4 \ldots 7$[1] with different fault rates $\epsilon = 0.2, 0.4, 0.6$ and 0.8. Typical results are shown in Figure 3. In the case of U-faults (UTR), as $\epsilon \to 0$ the probability of observing $k > 1$ tokens drops off and the probability of observing a single token approaches 1. In the case of S/U-faults (STR), the probability of observing $k > 1$ tokens also drops to 0 but at a slower rate. These observations comport with general results on limits of regularly-perturbed Markov chains [22].

[1] Modelchecking larger systems proved to be impractical with PRISM.

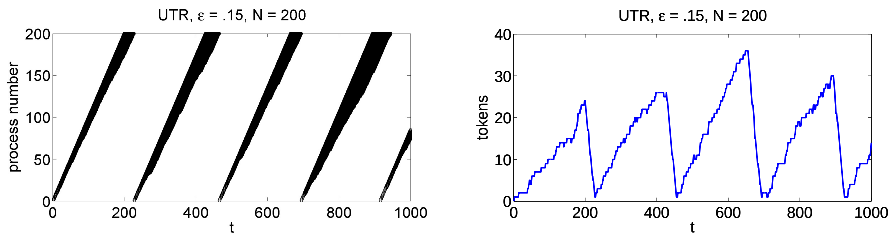

Fig. 4. Simulation of UTR for $N = 200, K = 201, \epsilon = 0.15$. *Left:* a raster plot of the locations of the tokens versus time. *Right:* Total number of tokens versus time; the number of tokens in each cycle grow to about $\epsilon N = 30$; this process is roughly periodic with period N.

For computing the expected stabilization times (EST) we assign a reward of 1 unit to every transition of the system and then we check for the following property in PRISM: $\mathbf{R}_{=?}[\mathbf{F}\ \mathit{legal}\ \{|\mathit{hastokens}| = k\}]$. Here $\mathbf{F}$ is the eventual operator and $\mathbf{R}$ is the expected reward operator in temporal logic. Both the EST and ERT increase with fault rates, and this increment is much more pronounced in STR.

Analysis for large N. For large N, we consider the Markov chain corresponding to UTR(N, K, ϵ) and analyze it directly; our observable Q will be the number of tokens.

We denote $t \in \mathbb{N}$ as the round number, T_t as the (random) number of tokens at round t, and L_t as the (random) subset of $\{0, \ldots, N-1\}$ giving the locations of the tokens at round t. Whenever a process changes value, it goes to the correct value with probability $1 - \epsilon(K-1)/K$ (there is a ϵ/K probability of a fault accidentally giving the right answer) and to any given incorrect state with probability $\delta := \epsilon/K$).

First consider the case of a single token in the system: $T_t = 1, L_t = \{n\}, n \neq 0$. Then $x_i = a$ for $i = 0, \ldots, n-1$ and $x_i = b$ for $i = n, \ldots, N-1$ for some $a \neq b$. The only process which will change is the nth, and without a fault the new value would be a; however, with faults we have $P(x_n \mapsto a) = 1 - \epsilon + \delta, P(x_n \mapsto b) = \delta, P(x_n \mapsto c) = (K-2)\delta$, where $c \neq a, c \neq b$. Changing to b is an error, but does not change the number of tokens, so the probability of having two tokens after the update is $(K-2)\delta$. Thus, if $T_t = 1 \wedge L_t = \{n\}, n \neq 0$, then $P(T_{t+1} = 1) = 1 - (K-2)\delta, P(T_{t+1} = 2) = (K-2)\delta$. On the other hand, if $L_t = \{0\}$, then the state of the system is $x_i = a$ for all i. The correct transition would be for $x_0 \mapsto a+1$, but even if there is an incorrect transition $x_0 \mapsto b \neq a+1$, the process still has one correct token.

Now consider the case of multiple tokens. We first assume that there are multiple tokens in a row, but away from the 0 process: $T_t = q, L_t = \{n, n+1, \ldots, n+q-1\}, 0 \notin L_t$. We show the result of one update, where we assume correct execution, and we denote by stars those states for which an error is possible:

process:	$n-1$	n	$n+1$	$n+2$	$\dots$	$n+q-2$	$n+q-1$	$n+q$
before:	x_{n-1}	x_n	x_{n+1}	x_{n+2}	$\dots$	x_{n+q-2}	x_{n+q-1}	x_{n+q-1}
after:	x_{n-1}	x^*_{n-1}	x^*_n	x^*_{n+1}	$\dots$	x^*_{n+q-3}	x^*_{n+q-2}	x_{n+q-1}

Consider the $(n+1)$st process; it should have a token under correct execution. But even if there is an update fault, unless the new value is one of the two values x_{n-1}, x_{n+1}, then it will still have a token. If the fault occurs at the $(n+1)$st process, then the probability of decreasing the number of tokens is either 2δ or δ (it is the latter if $x_{n-1} = x_{n+1}$). This holds true for all of the processes $n+1, \dots, n+q-1$. The process cannot decrease the number of tokens if the update fault occurs at process n. Thus the probability of decreasing the number of tokens by one is, to leading order, bounded above by $(2q-1)\delta$. If the new value of process n is anything other than x_{n-1} or x_n, then this increases the number of tokens, so the probability of an increase in tokens is $(K-2)\delta$. Therefore, whenever $T_t = q, L_t = \{n, n+1, \dots, n+q-1\}$, we have $P(T_{t+1} = q+1) = \delta(K-2)+O(\delta^2)$, and $P(T_{t+1} = q-1) \leq \delta(2q-1) + O(\delta^2)$.

Finally, consider the case where there are multiple tokens, but the set of tokens is not contiguous. Each contiguous block of tokens will act as if there are no other tokens in the system, since the entire algorithm is local. Thus, if we have l blocks of lengths q_l, with $\sum q_l = q$, then the probability of decreasing the number of tokens is then bounded above by $\delta \sum_l (2q_l - 1) + O(\delta^2) \leq \epsilon(2q-1)/(K-1) + O(\epsilon^2)$, while the probability of increasing the number of tokens is $\delta \sum_l (K-2) + O(\delta^2) = \epsilon l(K-2)/(K-1) + O(\epsilon^2)$. In short, more blocks means more likelihood of gaining a token, since tokens are created on the boundaries of blocks of tokens.

Consider a run of tokens next to the 0 process, i.e. $L_t = \{N-q, \dots, N-1\}$. If none of the values $x_{N-q}, \dots, x_{N-1}$ are equal to x_0, then each of these values updates to its predecessor, but the 0 process would stay fixed (notice the 0 process is sleeping throughout). This ends with exactly one token at 0 after q updates. If, however, one of these tokens is equal to x_0, this creates an erroneous token; as these q updates progress, we create an erroneous token each time there is a coincidence between states $N-1$ and 0. Since the incorrect states are chosen randomly, we expect about q/K of these coincidences.

We are now prepared to describe the typical "life cycle" of UTR. Start with a single token at process 0. The earliest a token can cycle around and reach 0 again is after N computational steps, so we want to compute the number of tokens we would expect to have after N steps, or T_N. Using the bounds on tokens increasing or decreasing, define $\widetilde{T}_t$ as the stochastic process with $\widetilde{T}_0 = 1$ and

$$\begin{aligned} P(\widetilde{T}_{t+1} = \widetilde{T}_t + 1) &= \epsilon(K-2)/(K-1), \\ P(\widetilde{T}_{t+1} = \widetilde{T}_t - 1) &= \epsilon(2q-1)/(K-1), \end{aligned} \tag{1}$$

and $\widetilde{T}_{t+1} = \widetilde{T}_t$ otherwise. By Chernoff's Theorem [23, Theorem 9.3], we have that $\mathcal{P}(\widetilde{T}_t < T_t) \sim e^{\rho t}$ for some $\rho < 0$. Rescaling and passing to the limit [24, Theorem 5.3] that, with probability exponentially close (in N) to one,

$$\widetilde{T}_N = \frac{N}{2}(1 - e^{-2\epsilon}) + o(N) \approx \epsilon N.$$

Thus if the system starts with a single token at the 0 process, the process will eventually generate a run of tokens of length $q \approx \epsilon N$, but this run will be effectively cleared up when it reaches process 0 — in fact, the mean number of tokens which survive after this run of tokens is absorbed by zero is about $q/K \approx \epsilon N/K < \epsilon$.

The simulations shown in Figure 4 corroborate the above analysis. These results are obtained by encoding a virtual token ring in Matlab. In every time-step, we first compute the correct transition. We then determine which processes could have a fault in this step (for the U model, the processes which updated; for the S/U model, all processes). For each processes which could have a fault, we chose an random number in [0,1]; if it was less than ϵ, then that process has an error, i.e. it is set to a randomly chosen value in $\{0, ..., K-1\}$.

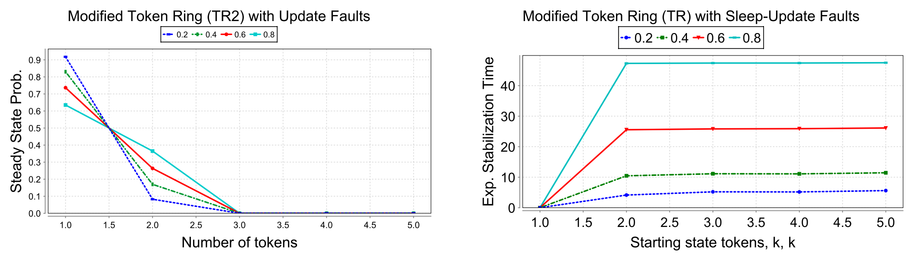

Fig. 5. *Left:* Steady state distribution of TR2 under update faults. *Right:* EST of TR2 under sleep-update faults for $N = 5$.

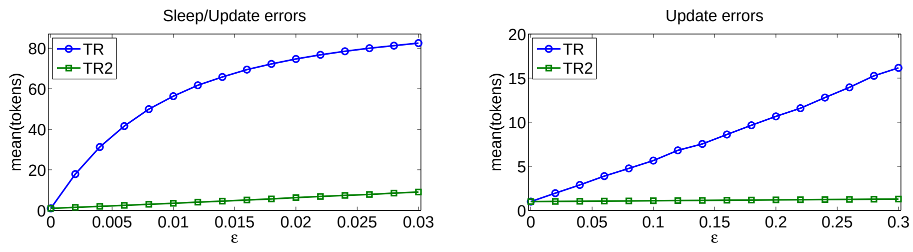

Fig. 6. Mean number of tokens in the steady state of TR and TR2 in both error models with $N = 100$. The mean is taken over 10^5 updates after the system is started in a correct state.

Modified Token Ring algorithm TR2. The TR algorithm does not perform well in the presence of incessant faults, even if they are only U-faults. One insight gained in the probabilistic arguments above is that, in the U-fault model, multiple tokens tend to come in runs. A simple way to correct this is to change the algorithm so that no process attempts to enter a state in which both it, and its predecessor, have a token. One such method is a "two lookbacks" version of TR, where each process looks at the inputs from two predecessors; we call this algorithm TR2. Basically, the algorithm looks to the two previous processors, and updates to its predecessor only if the previous two agree, otherwise it sleeps (see Figure 7). In Figure 5 we display the performance of TR2 with 5 processes

```
automaton TR2($N$,$K$:$\mathbb{N}$,const $i > 1$)
variables
  state $x_i$: $\{0,\ldots,K-1\}$ := uni[$\{0,\ldots,K-1\}$]
  input $x_{i-1}, x_{i-2}$: $\{0,\ldots,K\}$
transitions
  pre $x_i \neq x_{i-1} \wedge x_{i-1} = x_{i-2}$
  eff $x_i := x_{i-1}$
```

Fig. 7. Modified token ring algorithm.

using PRISM, and in Figure 6 for 100 processes using simulations. TR2 works significantly better than TR for both error models and in all parameter regimes.

4 Graph Coloring

We analyze S/U-faulty and U-faulty versions of the randomized graph coloring algorithm from [11]. The pseudocode in Figure 8 describes the self-stabilizing graph coloring algorithm. Fix an undirected graph G with N vertices. For each vertex $i \in \{0,\ldots,N-1\}$ in the graph, the set of neighbors (adjacent vertices) of i is denoted by NB_i, $K = \max_i |NB_i|$ is the maximum degree of G, and define the *palette* P of colors as the set $\{0,\ldots,K\}$. GC_i, $0 \leq i \leq N-1$, is a SA with the following components: (1) A set of state variables $X_i = \{x_i\}$, where x_i is of type P, and the value of x_i is the color of vertex i. (2) A set of input variables $U_i = \{x_j | j \in NB_i\}$, where each x_j is a variable of type P. NC_i is a derived variable; its value is the set of colors (values) of the neighbors of i. (3) An initial distribution $\bar{\mu}$ that is uniform over $Val(X_i)$, and (4) a set of probabilistic transitions we now define. We say that there is a collision at vertex i if the color of i is in NC_i. If there is a collision at i, process i picks a color in $P \setminus NC_i$ uniformly at random. Define

$$conflict(\mathbf{x}, i) = \exists\ j \in NB_i, \mathbf{x}.x_j = \mathbf{x}.x_i$$
$$hasconflict(\mathbf{x}) = \{i | conflict(\mathbf{x}, i)\}, \quad legal(\mathbf{x}) = (|hasconflict(\mathbf{x})| = 0).$$

For the graph coloring algorithm under U-faults and S/U-faults we are interested in the following quantitative properties:

Steady State(SS) : $SS(k)$ is the probability of the system being in a state $\mathbf{x}$ satisfying $|hasconflicts(\mathbf{x})| = k$, for some $k \in \{0,\ldots,N\}$.

```
automaton GC($N$:ℕ, $P$:set[ℕ], const $i \neq 0$)        derived
variables                                                $NC_i$: Set[$P$] := $\{x_j | j \in NB_i\}$
  state $x_i$: $P$ := uni[$P$]
  input $x_j$: $P$ where $j \in NB_i$                     transitions
                                                           pre $x_i \in NC_i$
                                                           eff $x_i$ := uni[$P \setminus NC_i$]
```

Fig. 8. Self-stabilizing graph coloring algorithm *autoGC*.

Expected Stabilization Time (EST) : Starting from an arbitrary state with k conflicts, the maximum expected time to reach a legal state.

Analysis for Small N. Just as was done for the token ring system, our analysis for UGC and SGC for a small number of processes employs the PRISM and MRMC model checkers. We compute SS and EST for randomly generated graphs with $N = 4 \dots 7$ with different error rates $\epsilon = 0.2, 0.4, 0.6$ and 0.8. Typical results are shown in Figures 9. For U-faults (UGC), the states with no conflicts have

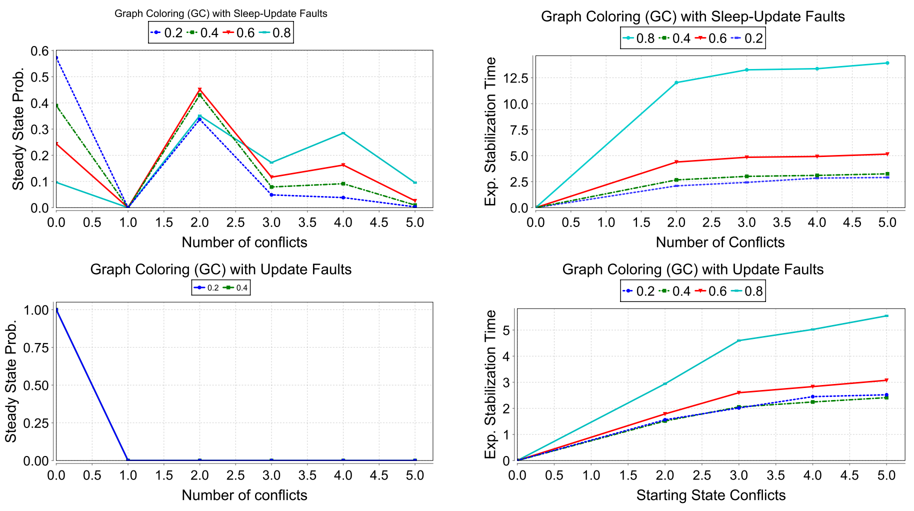

Fig. 9. GC(5, 4) under sleep-update (*Top*) faults SGC and update faults (*Bottom*) UGC. *Left:* Steady state distribution. *Right:* Expected Stabilization Times (EST).

a steady-state probability of 1. This is because under U-faults, once a legal configuration is reached, the system undergoes no further transitions, i.e. the legal states are absorbing states of UGC. In the case of S/U-faults (SGC), the steady-state probability of observing $k > 0$ conflicts is positive but it drops to 0 as the error rate goes to 0. For example, we observe that with an error rate of $\epsilon = 0.2$, there is a 5% probability of observing 4 conflicts in the long run. This type of quantitative results will be useful for analyzing performance of higher-level applications that use graph coloring as a service, e.g., assignment of channels in a multi-channel wireless network. Both for UGC and SGC, the expected time to stabilize (EST) to a legal state decreases as the error rate decreases. As expected the value of the EST for SGC is higher than that of UGC.

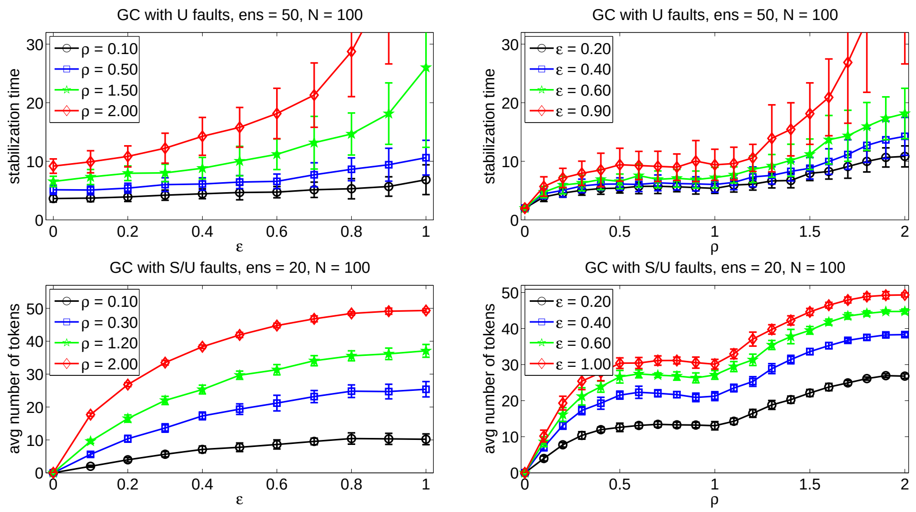

Fig. 10. Numerical simulation for UGC (*Top*) showing the time to reach a legal state and and SGC (*Bottom*) showing the mean number of collisions. In each case, we plot the same data versus both ϵ and ρ.

Analysis for Large N. We present numerical simulations for UGC and SGC in Figure 10. The obtain the undirected random graphs we choose N vertices uniformly at random in the unit disk, fix a *communication radius* $\rho \in [0, 2]$, and add edges between vertices that are within distance ρ of one another. Each data point corresponds to choosing an ensemble of *ens* graphs with a given ρ, each of which is simulated with fault rate ϵ; the point plotted is the ensemble mean and the error bars are the ensemble standard deviation. For UGC we plot the EST; for SGC, we instead plot the average number of collisions over a long run. In either case, a higher number is a signature of the poor performance. Of course, both of these metrics worsen when ϵ is increased, but there is a plateau (perhaps even a nonmonotonicity) for $\rho \in (1/2, 1)$. Surprisingly, in this range increasing the communication radius does not adversely affect performance.

References

1. Dolev, S.: Self-stabilization. MIT Press, Cambridge, MA, USA (2000)
2. Schneider, M.: Self-stabilization. ACM Comput. Surv. **25** (1993) 45–67

3. Herman, T.: A comprehensive bibliography on self-stabilization (2002) A Working Paper in the Chicago Journal of Theoretical Computer Science. http://www.cs.uiowa.edu/ftp/selfstab/bibliography/.
4. Steiner, C.: Bot in the delivery:KIVA systems. Forbes Magazine (2009) http://www.forbes.com/forbes/2009/0316/040_bot_time_saves_nine.html.
5. of Energy, U.: (Smart Grid) http://www.oe.energy.gov/smartgrid.htm.
6. Akyildiz, L.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Communications Magazine **40** (2002) 102–114
7. Pelc, A., Peleg, D.: Feasibility and complexity of broadcasting with random transmission failures. Theoretical Computer Science **370** (2007) 279–292
8. Kar, S., Moura, J.: Distributed average consensus in sensor networks with random link failures. In: Acoustics, Speech and Signal Processing. Volume 2., IEEE (2007) II–1013–II–1016
9. Nesterenko, M., Arora, A.: Local tolerance to unbounded byzantine faults. In: In IEEE SRDS. (2002) 22–31
10. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM **17** (1974) 643–644
11. Ghosh, S., Karaata, M.H.: A self-stabilizing algorithm for coloring planar graphs. Distrib. Comput. **7** (1993) 55–59
12. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Proc. 12th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06). Volume 3920 of LNCS., Springer (2006) 441–444
13. Katoen, J.P., Kemna, T., Zapreev, I., Jansen, D.N.: Bisimulation minimisation mostly speeds up probabilistic model checking. In: Tools and Algorithms for the Construction and Analysis of Systems, TACAS'07. Volume 4424 of LNCS., Springer (2007) 87–101
14. Agha, G., Meseguer, J., Sen, K.: PMaude: Rewrite-based specification language for probabilistic object systems. In: 3rd Workshop on Quantitative Aspects of Programming Languages (QALP'05). (2005)
15. Younes, H.: Ymer: A statistical model checker. In: Proc. 17th Intl. Conf. on Computer Aided Verification (CAV'05). Volume 3576 of LNCS., Springer (2005) 429–433
16. Hermanns, H.: Interactive Markov Chains : The Quest for Quantified Quality. Springer Berlin / Heidelberg (2002)
17. Segala, R.: A compositional trace-based semantics for probabilistic automata. In: Proc. of the 6th Intl. Conf. on Concurrency Theory (CONCUR '95). Volume 962 of LNCS., Philadelphia, PA, USA (1995) 234–248
18. Wu, S.H., Smolka, S., Stark, E.: Composition and behaviors of probabilistic I/O automata. Theoretical Computer Science **176** (1997) 1–38
19. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford University, CA (1997) Technical Report STAN-CS-TR-98-1601.
20. Canetti, R., Cheung, L., Kaynar, D., Liskov, M., Lynch, N., Pereira, O., Segala, R.: Task-structured Probabilistic I/O Automata. In: Proc. of the 8th Intl. Workshop on Discrete Event Systems – WODES'2006. (2006) IEEE catalog number 06EX1259.
21. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time markov chains. ACM Trans. Comput. Logic **1** (2000) 162–170
22. Young, P.: The evolution of conventions. Econometrica **61** (1993) 57–84
23. Billingsley, P.: Probability and Measure. John Wiley & Sons, New York (1995)
24. Shwartz, A., Weiss, A.: Large deviations for performance analysis. Chapman & Hall, London (1995)