Computing the Fault-Containment Time of Self-Stabilizing Algorithms Using Markov Chains and Lumping

Volker Turau(✉)

Institute of Telematics, Hamburg University of Technology, Hamburg, Germany
turau@tuhh.de

Abstract. The analysis of self-stabilizing algorithms is in the vast majority of all cases limited to the worst case stabilization time starting from an arbitrary configuration. Considering the fact that these algorithms are intended to provide fault tolerance in the long run this is not the most relevant metric. From a practical point of view the worst case time to recover in case of a single fault is much more crucial. This paper presents techniques to derive upper bounds for the mean time to recover from a single fault for self-stabilizing algorithms Markov chains in combination with lumping. To illustrate the applicability of the techniques they are applied to a self-stabilizing coloring algorithm.

1 Introduction

Fault tolerance aims at making distributed systems more reliable by enabling them to continue the provision of services in the presence of faults. The strongest form is *masking fault tolerance*, where a system continues to operate after faults without any observable impairment of functionality, i.e. safety is always guaranteed. In contrast *non-masking fault tolerance* does not ensure safety at all times. Users may experience incorrect system behavior, but eventually the system will fully recover. The potential of this concept lies in the fact that it can be used in cases where masking fault tolerance is too costly or even impossible to implement [11]. *Self-stabilizing algorithms* are a category of distributed algorithms that provide non-masking fault tolerance. They guarantee that systems eventually recover from transient faults of any scale such as perturbations of the state in memory or communication message corruption [6]. A critical issue is the length of the time span until full recovery. Examples are known where a memory corruption at a single process caused a vast disruption in large parts of the system and triggered a cascade of corrections to reestablish safety. Thus, an important issue is the containment of the effect of transient faults.

A *fault-containing* system has the ability to contain the effects of transient faults in space and time. The goal is to keep the extent of disruption during recovery proportional to the extent of the faults. An extreme case of fault-containment with respect to space is given when the effect of faults is bounded to the set of faulty nodes. Azar et al. call this *error confinement* [1]. More relaxed forms of

P. Spirakis and P. Tsigas (Eds.): SSS 2017, LNCS 10616, pp. 62–77, 2017.
https://doi.org/10.1007/978-3-319-69084-1_5

fault-containment are known as time-adaptive self-stabilization [19], scalable self-stabilization [14], strong stabilization [8], and 1-adaptive self-stabilization [3].

A configuration is called *k-faulty*, if in a legitimate configuration exactly k processes are hit by a fault (a configuration is called *legitimate* if it conforms with the specification). A large body of research focuses on fault-containing for 1-faulty configurations. Several metrics have been introduced to quantify the containment behavior in the 1-faulty case [13,18]. A distributed algorithm $\mathcal{A}$ has *contamination radius* r if only nodes within the r-hop neighborhood of the faulty node change their state during recovery from a 1-faulty configuration. The *containment time* of $\mathcal{A}$ denotes the worst-case number of rounds any execution of $\mathcal{A}$ starting at a 1-faulty configuration needs to reach a legitimate configuration. In technical terms this corresponds to the *worst case time to recover* in case of a single fault. For randomized algorithms the expected number of rounds to reach a legitimate configuration corresponds to the *mean time to recover* (MTT).

Over the last two decades a large number of self-stabilizing algorithms have been published. Surprisingly the analysis of the vast majority of these algorithms is confined to the worst case stabilization time starting from an arbitrary configuration. Considering the fact that these algorithms are intended to provide fault tolerance in the long run this is not the most relevant metric at all. From a practical point of view the worst case time to recover from a 1-faulty configuration is much more crucial. This statement is justified considering the fact that the probability for a 1-faulty configuration is much larger then that for k-faulty configuration with large values of k. The reason is that a distributed system consists of independently operating computers where transient faults such as memory faults are independent events. Considering this fact it comes as a surprise that only in a few cases fault-containment metrics have been considered [12,25]. One reason may be that there are many techniques available to determine the worst case stabilization time of an algorithm, e.g., potential functions and convergence stairs, but there is no systematic approach to determine the containment metrics.

This paper discusses two techniques to analyze the containment time of randomized self-stabilizing algorithms with respect to memory and message corruption. The execution of the algorithm is modeled as a stochastic process. Let X be the random variable that represents the number of rounds the system requires to reach a legitimate configuration when starting in a 1-faulty configuration. Then the MTT of the algorithm is equal to $E[X]$; thus, we are interested in upper bounds for $E[X]$. In some cases it will be possible to derive an explicit expression for $E[X]$. An alternative is to use an absorbing Markov chain to derive an equation for $E[X]$. This equation may be solvable with a software package based on symbolic mathematics. However, the state space explosion problem will preclude success for many real world problems. An important optimization technique for the reduction of the complexity of Markov chains is *lumping* [17]. Lumping is a method based on the aggregation of states that exhibit the same behavior. It leads to a smaller Markov chain that retains the same performance characteristics as the original one.

The contribution of this paper is a discourse about computing containment metrics of self-stabilizing algorithms in the 1-faulty case. We present and apply techniques based on Markov chains to compute upper bounds for these metrics. In particular we demonstrate how lumping can be applied to reduce the complexity of the Markov chains. To demonstrate the usability of the techniques we apply them to a self-stabilizing coloring algorithm as a case study. We derive an absolute bound for the expected containment time and show that the variance is bounded by a surprisingly small constant independent of the network's size. We believe that the techniques can also be applied to other algorithms. The proofs of the technical lemmata can be found in the technical report [24].

2 Related Work

There exist several techniques to analyze self-stabilizing algorithms: potential functions, convergence stairs, Markov chains, etc. Markov chains are particularly useful for randomized algorithms [9]. Their main drawback is that in order to set up the transition matrix the adjacency matrix of the graph must be known. This restricts the applicability of this method to *small* or highly *symmetric* instances. Lee DeVille and Mitra apply model checking tools to Markov chains for cases of networks of small size ($n \leq 7$) to determine the expected stabilization time [5]. An example for highly symmetric networks are ring topologies, see for example [10,26]. Fribourg et al. model randomized distributed algorithms as Markov chains using the technique of coupling to compute upper bounds for the stabilization times [10]. Yamashita uses Markov chains to model self-stabilizing probabilistic algorithms and to prove stabilization [26]. Mitton et al. consider a randomized self-stabilizing $\Delta+1$-coloring algorithm and model this algorithm in terms of urns/balls using a Markov chain to get a bound for the stabilization time [22]. They evaluated the Markov chain for networks up to 1000 nodes analytically and by simulations. Crouzen et al. model faulty distributed algorithms as Markov decision processes to incorporate the effects of random faults when using a non-deterministic scheduler [4]. They used the PRISM model-checker to compute long-run average availabilities.

3 System Model

This paper uses the synchronous model of distributed computing as defined in the standard literature [6,13,23]. A distributed system is represented as an undirected graph $G(V, E)$ where V is the set of *nodes* and $E \subseteq V \times V$ is the set of *edges*. Let $n = |V|$ and $\Delta(G)$ denote the maximal degree of G. The topology is assumed to be fixed. If two nodes are connected by an edge, they are called *neighbors*. The set of neighbors of node v is denoted by $N(v) \subseteq V$ and $N[v] = N(v) \cup \{v\}$. Each node stores a set of variables. The values of all variables constitute the *local state* of a node. Let σ denote the set of possible local states of a node. The *configuration* of a system is the tuple of all local states of all nodes. $\Sigma = \sigma^n$ denotes the set of global states. A configuration is

called *legitimate* if it conforms with the specification. The set of all legitimate configurations is denoted by $\mathcal{L}$.

Nodes communicate either via locally shared memory (Sect. 4) or by exchanging messages (Sect. 6). In the shared memory model each node executes a protocol consisting of a list of rules of the form *guard* $\longrightarrow$ *statement*. The guard is a Boolean expression over the node's variables and its neighbors. The statement consists of a series of commands. A node is called *enabled* if one of its guards evaluates to true. The execution of a statement is called a *move*.

Execution of the statements is performed in a synchronous style, i.e., all enabled nodes execute their code in every round. In the message passing model a node performs three steps per round: receiving messages from neighbors, executing code, and sending messages to neighbors. An *execution* $e = \langle c_0, c_1, c_2, \ldots \rangle$, $c_i \in \Sigma$ is a sequence of configurations, where c_0 is called the *initial configuration* and c_i is the configuration after the i-th step. In other words, if the current configuration is c_{i-1} and all enabled nodes make a move, then this yields c_i.

The containment behavior of a self-stabilizing algorithm is characterized by the contamination radius and the containment time. In this paper we are interested in the most common fault situation: 1-faulty configurations. Such configurations arise when a single node v is hit by a memory corruption or a single message sent by v is corrupted. Denote by R_v the subgraph of the communication graph G induced by the nodes that are engaged in the recovery process from a 1-faulty configuration triggered by a fault at v. The contamination radius is equal to $\max\{dist(v, w) \mid w \in R_v\}$.

The stabilization time $st(n)$ is an obvious upper bound for the containment time. This can be narrowed down to $O(st(\Delta^r))$, if the contamination radius r is known. There are two situations in which it is possible to obtain better bounds: Either the structure of R_v is considerably simpler than that of G or the faulty configuration is close to a legitimate configuration (e.g., only v is not legitimate).

4 Contamination Radius

If an algorithm using the shared memory model has contamination radius r and no other fault occurs then this fault will not spread beyond the r-hop neighborhood of the faulty node v. In this case $R_v \subseteq G_v^r$, where G_v^r is the subgraph induced by nodes w with $dist(v, w) \leq r$. As an example consider the well known self-stabilizing algorithm $\mathcal{A}_1$ to compute a maximal independent set (see Algorithm 1).

Lemma 1. *Algorithm $\mathcal{A}_1$ has contamination radius two.*

Proof. Let v be a node hit by a memory corruption. First suppose the state of v changes from IN to OUT. Let $u \in N(v)$ then $u.state = OUT$. If u has an neighbor $w \neq v$ with $w.state = IN$ then u will not change its state during recovery. Otherwise, if all neighbors of u except v had state OUT node u may change state during recovery. But since these neighbors of u have a neighbor with

state IN they will not change their state. Thus, in this case only the neighbors of v may change state during recovery.

Next suppose that $v.state$ changes from OUT to IN. Then v and those neighbors of v with state IN can change to OUT. Then arguing as in the first case only nodes within distance two of v may change their state during recovery. □

Algorithm 1. Self-stabilizing algorithm $\mathcal{A}_1$ to compute a MIS.

if $state = IN \wedge \exists w \in N(v)$ *s.t.* $w.state = IN$ **then**
  $state := OUT$
if $state = OUT \wedge \forall w \in N(v)$ $w.state = OUT$ **then**
  if *random bit from 0,1 = 1* **then**
    $state := IN$

In the following we consider another example: $\Delta + 1$-coloring. Most distributed algorithms for this problem follow the same pattern. A node that realizes that it has selected the same color as one of its neighbors chooses a new color from a finite color palette. This palette does not include the current colors of the node's neighbors. To be executed under the synchronous scheduler these algorithms are either randomized or use identifiers for symmetry breaking. Variations of this idea are followed in [7,15,22]. As an example consider algorithm $\mathcal{A}_2$ from [15] (see Algorithm 2). Due to its choice of a new color from the palette algorithm $\mathcal{A}_2$ has contamination radius at least $\Delta(G)$ (see Fig. 1).

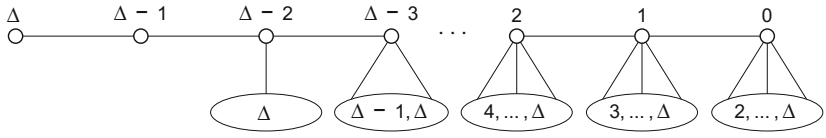

Fig. 1. The numbers indicate the nodes' colors. If the left-most node is hit by a fault and changes its color to $\Delta - 1$, then all nodes on the horizontal line may change color.

Algorithm 2. Self-stabilizing $\Delta + 1$-coloring algorithm $\mathcal{A}_2$ from [15].

if $c \neq \max(\{0, \ldots, \Delta\} \backslash \{w.c \mid w \in N(v)\})$ **then**
  if *random bit from 0,1 = 1* **then**
    $c := \max(\{0, \ldots, \Delta\} \backslash \{w.c \mid w \in N(v)\})$

A minor modification of algorithm $\mathcal{A}_2$ dramatically changes matters. Algorithm $\mathcal{A}_3$ (see Algorithm 3) has containment radius 1 (see Lemma 2) and R_v is a star graph with center v. Note that neighbors of v that change their color during recovery form an independent set.

Algorithm 3. Self-stabilizing $\Delta + 1$-coloring algorithm $\mathcal{A}_3$.

if $\exists w \in N(v)$ *s.t.* $c = w.c$ **then**
  if *random bit from 0,1 = 1* **then**
    $c :=$ choose $\{0, \ldots, \Delta\} \backslash \{w.c \mid w \in N(v)\}$

Lemma 2. *Algorithm $\mathcal{A}_3$ has contamination radius one.*

Proof. Let v be a node hit by a memory corruption changing its color to a color c already chosen by at least one neighbor of v. Let $N_{conf} = \{w \in N(v) \mid w.c = c\}$. In the next round the nodes in $N_{conf} \cup \{v\}$ will get a chance to choose a new color. The choices will only lead to conflicts between v and other nodes in N_{conf}. Thus, the fault will not spread beyond the set N_{conf}. With a positive probability the set N_{conf} will contain fewer nodes in each round. □

5 Containment Time

As the contamination radius the containment time strongly depends on the concrete structure of G. This can be illustrated with algorithm $\mathcal{A}_1$. Note that in this case R_v can contain any subgraph H with $\Delta(G)$ nodes. As an example let G consist of H and an additional node v connected to each node of H. A legitimate configuration is given if the state of v is IN and all other nodes have state OUT (Fig. 2 left). If v changes its state to OUT due to a fault then all nodes may change to state IN during the next round. Thus, there is little hope for a bound below the trivial bound. Similar arguments hold for the second 1-faulty configuration of $\mathcal{A}_1$ shown on the right of Fig. 2.

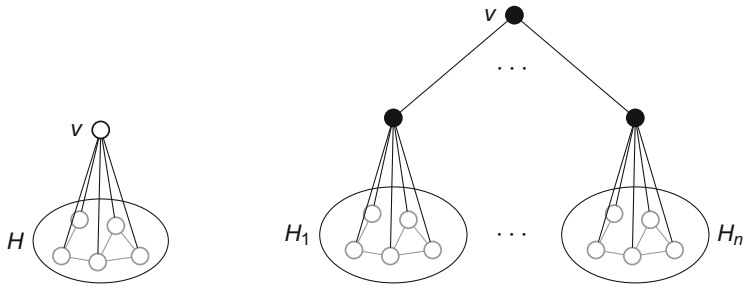

Fig. 2. 1-faulty configurations of $\mathcal{A}_1$ caused by a memory corruption at v. Nodes drawn in bold have state IN. The depicted graphs correspond to R_v.

We introduce two techniques to derive upper bounds for the expected containment time of a randomized synchronous self-stabilizing algorithm $\mathcal{A}$. Let X be the random variable that denotes the number of rounds until the system has reached a legitimate configuration when starting in a 1-faulty configuration c. The expected containment time equals the expected value $E[X]$. An analytical approach to compute an upper bound for $E[X]$ is to derive a bound for $g(i) = P\{X = i\}$ and use this to estimate $E[X] = \sum_{i=1}^{\infty} ig(i)$. This approach is often infeasible due to the high number of states. A remedy is the lumping technique explained in the following section.

5.1 Lumpable Markov Chains

The self-stabilizing algorithm $\mathcal{A}$ can be regarded as a transition systems of Σ. In each round the current configuration $c \in \Sigma$ is transformed into a new configuration $\mathcal{A}(c) \in \Sigma$. This process is described by the transition matrix P where p_{ij} gives the probability to move from configuration c_i to c_j in one round, i.e., $\mathcal{A}(c_i) = c_j$. To reduce the complexity we partition Σ into subsets $\Sigma_0, \ldots, \Sigma_l$ and consider these as the states of a Markov chain. A partitioning is called *lumpable* if the subsets Σ_i have the property that for each pair i, j the probability of a configuration $c \in \Sigma_i$ to be transformed in one round into a configuration of Σ_j is independent of the choice of $c \in \Sigma_i$ (Definition 6.3.1 [17]). This probability is then interpreted as the transition probability from Σ_i to Σ_j.

A state c_i of a Markov chain is called *absorbing* if $p_{ii} = 1$ and $p_{ij} = 0$ for $i \neq j$. For each self-stabilizing algorithm, the set of all absorbing states is equal to $\mathcal{L}$, the legitimate configurations. The number of rounds to reach a configuration in $\mathcal{L}$ starting from a given configuration $c_i \in \Sigma_i$ equals the number of steps before being absorbed in $\mathcal{L}$ when starting in state Σ_i. This equivalence allows us to use techniques from Markov chains to compute the stabilization time and thus, the containment time. Let Σ_0 consist of a single 1-faulty configuration and $\Sigma_l = \mathcal{L}$. Then $E[X]$ equals the expected number of rounds to reach Σ_l from Σ_0, where Σ_0 ranges over all 1-faulty configurations.

5.2 Example

To illustrate this approach we consider again algorithm $\mathcal{A}_3$. Let v be a node that changes in a legitimate state its color to c_f due to a memory fault. Let c_0 be the new configuration. This causes a conflict with those neighbors of v that had chosen c_f as their color. After the fault only nodes contained in R_v (a star graph) change their state. Once a neighbor has chosen a color different from c_f then it becomes passive (at least until the next transient fault). Let d be the number of neighbors of v that have color c_f in c_0. Denote by Σ_j the set of all configurations reachable from c where exactly $d - j$ neighbors of v are in conflict with v. Then $\Sigma_0 = \{c_0\}$ and $\Sigma_d \subseteq \mathcal{L}$. Let $c \in \Sigma_i$. Then $\mathcal{A}_3(c) \notin \Sigma_j$ for all $j < i$. This partitioning is not lumpable because the probability of a configuration $c \in \Sigma_i$ to be transformed in one round into a fixed configuration of Σ_j is not independent of the choice of $c \in \Sigma_i$. This issue can be resolved by using lower bounds of these probabilities. For $i < j$ let $p_{ij} \geq 0$ be a constant such that $P(\mathcal{A}_3(c) \in \Sigma_j) \geq p_{ij}$ for all $c \in \Sigma_i$. Furthermore, let $p_{ij} = 0$ for $j < i$ and for $i = 0, \ldots, d$

$$p_{ii} = 1 - \sum_{j=i+1}^{d} p_{ij}.$$

Then $p_{ii} \geq 0$ because $0 \leq \sum_{j=i}^{d} p_{ij} \leq \sum_{j=i}^{d} P(\mathcal{A}_3(c) \in \Sigma_j) = 1$ for each fixed $c \in \Sigma_i$. Thus, the matrix $P = (p_{ij})$ is a stochastic with $p_{dd} = 1$. P describe a new Markov chain C. The expected number of steps of C before being absorbed

by Σ_d when starting from state Σ_0 is an upper bound for $E[X]$, the expected containment time of $\mathcal{A}_3$.

To analyze C standard techniques can be applied. Let Q be the matrix obtained from P by removing the last row and the last column. Q describes the probability of transitioning from some transient state to another. The following properties are well known, e.g. Theorem 3.3.5 of [17]. Denote the $d \times d$ identity matrix by E_d. Then $N = (E_d - Q)^{-1}$ is the fundamental matrix of the Markov chain. The expected number of steps before being absorbed by Σ_d when starting from Σ_i is the i-th entry of vector $a = NI_d$ where I_d is a length-d column vector whose entries are all 1. The variance of these numbers of steps is given by the entries of $(2N - E_d)a - a_{sq}$ where a_{sq} is derived from a by squaring each entry.

In the rest of this paper the techniques are exemplary applied to a self-stabilizing $(\Delta + 1)$-coloring algorithm $\mathcal{A}_{col}$ using the message passing model. For the approach based on Markov chains a software package based on symbolic mathematics is used to compute $E[X]$ and $Var[X]$.

6 Algorithm $\mathcal{A}_{col}$

This section introduces coloring algorithm $\mathcal{A}_{col}$ (see Algorithm 4). Computing a $\Delta + 1$-coloring in expected $O(\log n)$ rounds with a randomized algorithm is long known [16,21]. Algorithm $\mathcal{A}_{col}$ follows the pattern sketched in Sect. 4. We derived it from a algorithm contained in [2] (Algorithm 19) by adding the self-stabilization property. The presented techniques can also be applied to other randomized coloring algorithms such as [7,15,22]. The main difference is that $\mathcal{A}_{col}$ assumes the message passing model, more precisely the synchronous $\mathcal{CONGEST}$ model as defined in [23]. Algorithm $\mathcal{A}_{col}$ stabilizes after $O(\log n)$ rounds with high probability whereas the above cited self-stabilizing algorithms all require a linear number of rounds. Since synchronous local algorithms can be converted to asynchronous self-stabilizing algorithms [20], there are self-stabilizing algorithms for $\Delta + 1$-coloring that are faster than $\mathcal{A}_{col}$. However, they entail a burden on memory resources, high traffic costs, and a long computational time.

At the start of each round of $\mathcal{A}_{col}$ each node broadcasts its current color to its neighbors. Based on the information received from its neighbors a node decides either to keep its color (final choice), to choose a new color or no color (value $\perp$). In particular with equal probability a node v draws uniformly at random a color from the set $\{0, 1, \ldots, \delta(v)\} \backslash tabu$ or indicates that it made no choice (see function `randomColor`). Here, *tabu* is the set of colors of neighbors of v that already made their final choice.

In the algorithm of [2] a node maintains a list with the colors of those neighbors that made their final choice. A fault changing this list is difficult to contain. Furthermore, in order to notice a memory corruption at a neighbor, each node must continuously send its state to all its neighbors and cannot stop to do so. This is the price of self-stabilization and well known [6]. These considerations lead to the design of Algorithm $\mathcal{A}_{col}$. Each node only maintains the chosen color

and whether its choice is final (variables `c` and `final`). $\mathcal{A}_{col}$ uses two additional variables `tabu` and `occupied`, but they are reset at the beginning of every round. In every round a node sends the values of these variables to all neighbors. To improve fault containment a node's final choice of a color is only withdrawn if it coincides with the final choice of a neighbor. To achieve a $\Delta + 1$-coloring a node makes a new choice if its color is larger than its degree. This situation can only originate from a fault.

Algorithm 4. Algorithm $\mathcal{A}_{col}$ as executed by a node v in each round.

```
Set<Color> tabu := ∅, occupied := ∅;
broadcast(c, final) to all neighbors w ∈ N(v);
for all neighbors w ∈ N(v) do
    receive(c_w, final_w) from node w;
    if c_w ≠ ⊥ then
        occupied := occupied ∪ {c_w};
        if final_w then tabu := tabu ∪ {c_w} ;
if c = ⊥ ∨ c > δ(v) then
    final := false;
else
    if final then
        if c ∈ tabu then final := false ;
    else
        if c ∉ occupied then final := true ;
if final = false then c := randomColor(v, tabu) ;

function Color randomColor(Node v, Set<Color> tabu)
    if random bit from 0,1 = 1 then return ⊥ ;
    return random color from {0,1,..., δ(v)}\tabu;
```

Next we prove correctness and ythe stabilization time of $\mathcal{A}_{col}$. A configuration is called a *legal coloring* if the values of variable c form a $\Delta + 1$-coloring. It is called *legitimate* if it is a legal coloring and $v.final = true$ for each node v.

Lemma 3. *A node v can change the value of variable $final$ from $true$ to $false$ only in the first round or when a fault occured just before the start of this round.*

Proof. Let $v.c = c_r$ at the beginning of the round. In order for v to set $v.final$ to $false$ one of the following conditions must be met at the start of the round: $c_r > \delta(v)$, $c_r = \perp$, or v has a neighbor w with $w.final = true$ and $w.c = c_r$.

The lemma is obviously true in the first case. Suppose that $c_r = \perp$ and $v.final = true$ at the round's start. If during the previous round the value of $v.final$ was set to $true$ then $v.c$ can not be $\perp$ at the start of this round. Hence, at the start of the previous round $final$ already had value $true$. But in this case $v.c$ was not changed in the previous round and thus, $c_r \neq \perp$, contradiction. Finally assume the last condition. Then v and w cannot have changed their value of c in the previous round, because then $final = true$ would be impossible at the start

of this round. Thus, v sent $(c_r, true)$ in the previous round. Hence, if $w.c = c_r$ at that time, w would have changed $w.final$ to $false$, again a contradiction. □

Lemma 4. *A node setting $final$ to true will not change its variables as long as no error occurs.*

Proof. Let v be a node that executes $final := true$. If v changes the value back to $false$ in a later round then by Lemma 3 a fault must have occured. Thus in an error-free execution node v will never change variable $final$ again. Since a node can only change variable c if $final = false$ the proof is complete. □

Lemma 5. *If at the end of a round during which no error occured each node v satisfies $v.final = true$ then the configuration is legitimate and remains legitimate as long as no error occurs.*

Proof. Note that no node changed its color during that round. If at the start of the round $v.final = true$ was already satisfied then none of v's neighbors also having $final = true$ had the same color as v. Next consider a neighbor w of v with $w.final = false$ at the start of the round. Since v sent $(v.c, true)$ at the start of this round, node w would have set $final$ to $false$ if it had chosen the same color as v. Contradiction. Finally consider that case that $v.final = false$ at the start of the round. Since v changed $final$ to $true$, none of its neighbors had chosen the same color as v. Thus, the configuration is legitimate. Obviously, this property can only be changed by a fault. □

The following theorem can be proved with the help of the last three lemmas.

Theorem 6 [24]**.** *Algorithm $\mathcal{A}_{col}$ is self-stabilizing and computes a $\Delta + 1$-coloring within $O(\log n)$ rounds with high probability (i.e. with probability at least $1 - n^c$ for any $c \geq 1$). $\mathcal{A}_{col}$ has contamination radius 1.*

7 Fault Containment Time of Algorithm $\mathcal{A}_{col}$

There is a significant difference from the shared memory model compared to the message passing model when analyzing the containment time. Firstly, a 1-faulty configuration also arises when a single message sent by a node v is corrupted. Secondly, this may cause v's neighbors to send messages they would not send in a legitimate configuration. Even though the state of nodes outside G_v^r does not change, these nodes may be forced to send messages. Thus, in general the analysis of the containment time cannot be performed by considering G_v^r only. This is only possible in cases when a fault at v does not force nodes at distance $r + 1$ to send messages they would not send had the fault not occurred.

In the following the fault containment behavior of $\mathcal{A}_{col}$ for 1-faulty configurations is analyzed. Two types of transient errors are considered:

1. A single broadcast message sent by v is corrupted. Note that the alternative of using $\delta(v)$ unicast messages instead a single broadcast has very good fault containment behavior but is slower due to the handling of acknowledgements.

2. Memory corruption at node v, i.e., the value of at least one of the two variables of v is corrupted.

The first case is analyzed analytically whereas for the second case Markov chains are used. The *independent degree* $\delta_i(v)$ of a node v is the size of a maximum independent set of $N(v)$. Let $\Delta_i(G) = \max\{\delta_i(v) \mid v \in V\}$.

7.1 Message Corruption

If a message broadcast by v contains a color c_f different from $v.c$ or the value $false$ for variable $final$ then the message $(c_f, false)$ has no effect on any $w \in N(v)$ regardless of the value of c_f, since $w.final = true$ for all $w \in N(v)$. Thus, this corrupted message has no effect at all. In order to compute the containment time for $\mathcal{A}_{col}$ we first compute the contamination radius.

Lemma 7. *The contamination radius of algorithm $\mathcal{A}_{col}$ after a single corruption of a broadcast message sent by node v is 1. At most $\delta_i(v)$ nodes change their state during recovery.*

Proof. It suffices to consider the case that v broadcasts message $(c_f, true)$ with $c_f \neq v.c$. Let $N_{conf}(v) = \{w \in N(v) \mid w.c = c_f\}$. The nodes in $N_{conf}(v)$ form an independent set, because they all have the same color. Thus $|N_{conf}(v)| \leq \delta_i(v)$.

Let $u \in V \backslash N[v]$. This node continues to send $(u.c, true)$ after the fault. Thus, a neighbor of u that changes its color will not change its color to $u.c$. This yields that no neighbor of u will ever send a message with $u.c$ as the first parameter. This is also true in case $u \in N(v) \backslash N_{conf}(v)$. Hence, no node outside $N_{conf}(v) \cup \{v\}$ will change its state, i.e. the contamination radius is 1.

Let $w \in N_{conf}(v)$. When the faulty message is received by w it sets $w.final$ to false. Before the faulty message was sent no neighbor of v had the same color as v. Thus, in the worst case a node $w \in N_{conf}(v)$ will choose $v.c$ as its new color and send $(v.c, false)$ to all neighbors. Since $v.final = true$ this will not force v to change its state. Thus, v keeps broadcasting $(v.c, true)$ and therefore no neighbor w of v will ever reach the state $w.c = v.c$ and $w.final = true$. Hence v will never change its state. □

With this result Theorem 6 implies that the containment time of this fault is $O(\log \delta_i(v))$ on expectation. The following theorem gives an absolute bound for the expected value of the containment time.

Theorem 8 [24]**.** *The expected value for the containment time of algorithm $\mathcal{A}_{col}$ after a corruption of a message broadcast by node v is at most $\frac{1}{\ln 2} H_{\delta_i(v)} + 1/2$ rounds (H_i denotes the i^{th} harmonic number) with a variance of at most*

$$\frac{1}{\ln^2 2} \sum_{i=1}^{\delta_i(v)} \frac{1}{i^2} + \frac{1}{4} \leq \frac{\pi^2}{6 \ln^2 2} + \frac{1}{4} \approx 3.6737.$$

7.2 Memory Corruption

This section demonstrates the use of Markov chains in combination with lumping to analyze the containment time. We consider the case that the memory of a single node v is hit by a fault. The analysis breaks down the stabilizing executions into several states and then computes the expected time for each of these phases. First we look at the case that the fault causes variable $v.final$ to change to $false$. If $v.c$ does not change, then a legitimate configuration is reached after one round. So assume $v.c$ also changes. Then the fault will not affect other nodes. This is because no $w \in N(v)$ will change its value of $w.c$ since $w.final = true$ and $v.final = false$. Thus, with probability at least $1/2$ node v will choose in the next round a color different from the colors of all neighbors and terminate one round later. Similar to X_d let random variable Z_d denote the number of rounds until a legal coloring is reached ($d = |N_{conf}(v)|$). It is easy to verify that $E[Z_d] = 3$ in this case.

The last case is that only variable $v.c$ is affected (i.e. $v.final$ remains $true$). The main difference to the case of a corrupted message is that this fault persists until $v.c$ has again a legitimate value. Let c_f be the corrupted value of $v.c$ and suppose that $N_{conf}(v) = \{w \in N(v) \mid w.c = c_f\} \neq \emptyset$. A node outside $S = N_{conf}(v) \cup \{v\}$ will not change its state (c.f. Lemma 7). Thus, the contamination radius is 1 and at most $\delta_i(v) + 1$ nodes change state. Let $d = |N_{conf}(v)|$. The subgraph G_S induced by S is a star graph with $d + 1$ nodes and center v.

Lemma 9. *To find a lower bound for $E[Z_d]$ we may assume that w can choose a color from $\{0, 1\} \backslash tabu$ with $tabu = \emptyset$ if $v.final = false$ and $tabu = \{v.c\}$ otherwise and v can choose a color from $\{0, 1, \ldots, d\} \backslash tabu$ with $tabu \subseteq \{0, 1\}$.*

Proof. When a node $u \in S$ chooses a color with function `randomColor` the color is randomly selected from $C_u = \{0, 1, \ldots, \delta(v)\} \backslash tabu$. Thus, if w and v choose colors in the same round, the probability that the chosen colors coincide is $|C_w \cap C_v|/|C_w||C_v|$. This value is maximal if $|C_w \cap C_v|$ is maximal and $|C_w||C_v|$ is minimal. This is achieved when $C_w \subseteq C_v$ and C_v is minimal (independent of the size of C_w) or vice versa. Thus, without loss of generality we can assume that $C_w \subseteq C_v$ and both sets are minimal. Thus, for $w \in N_{conf}(v)$ the nodes in $N(w) \backslash \{v\}$ already use all colors from $\{0, 1, \ldots, \delta(v)\}$ but 0 and 1 and all nodes in $N(v) \backslash N_{conf}(v)$ already use all colors from $\{0, 1, \ldots, \delta(v)\}$ but $0, 1, \ldots, d$. Hence, a node $w \in N_{conf}(v)$ can choose a color from $\{0, 1\} \backslash tabu$ with $tabu = \emptyset$ if $v.final = false$ and $tabu = \{v.c\}$ otherwise. Furthermore, v can choose a color from $\{0, 1, \ldots, d\} \backslash tabu$ with $tabu \subseteq \{0, 1\}$. In this case $tabu = \emptyset$ if $w.final = false$ for all $w \in N_{conf}(v)$. □

Thus, in order to bound the expected number of rounds to reach a legitimate state after a memory corruption we can assume that $G = G_S$ and $u.final = true$ and $u.c = 0$ (i.e. $c_f = 0$) for all $u \in S$. After one round $u.final = false$ for all $u \in S$. To compute the expected number of rounds to reach a legitimate state an execution of the algorithm for the graph G_s is modeled by a Markov chain $\mathcal{M}$ with the following states (I is the initial state) using the lumping technique:

I: Represents the faulty state with $u.c = 0$ and $u.final = true$ for all $u \in S$.
C^i: Node v and exactly $d - i$ non-center nodes will not be in a legitimate state after the following round ($0 \leq i \leq d$). In particular $v.final = false$ and $w.c = v.c \neq \perp$ or $v.c = w.c = \perp$ for exactly $d - i$ non-center nodes w.
P: Node v has not reached a legitimate state but will do so in the next round. In particular $v.final = false$ and $v.c \neq w.c$ for all non-center nodes w.
F: Node v is in a legitimate state, i.e. $v.final = true$ and $v.c \neq w.c$ for all non-center nodes w, but $w.c$ may be equal to $\perp$.

$\mathcal{M}$ is an absorbing chain with F being the single absorbing state. Note that when the system is in state F, then it is not necessarily in a legitimate state. This state reflects the set of configurations considered in the last section.

Lemma 10 [24]. *The transition probabilities of $\mathcal{M}$ are as follows:*

$I \longrightarrow P$: $\frac{d-1}{2d} + \frac{1}{d}\left(\frac{1}{2}\right)^{d+1}$
$I \longrightarrow C^0$: $\frac{d-1}{d}\left(\frac{1}{2}\right)^{d+1} + \frac{1}{2d}$
$I \longrightarrow C^j$: $\binom{d}{d-j}\left(\frac{1}{2}\right)^{d+1}$ $(0 < j \leq d)$
$C^i \longrightarrow C^j$: $\binom{d-i}{d-j}\left(\frac{1}{2}\right)^{d-i+1} + \frac{1}{d-i+1}\binom{d-i}{j-i}\left(\frac{1}{4}\right)^{d-i}(3^{d-j} - 2^{d-j})$ $(0 \leq i \leq j \leq d)$
$C^i \longrightarrow P$: $\frac{1}{d-i+1}\left(\frac{3}{4}\right)^{d-i} + \frac{d-i-1}{2(d-i+1)}$ $(0 \leq i < d)$
$C^d \longrightarrow P$: $1/2$
$P \longrightarrow F$: 1

We first calculate the expected number $E[A_d]$ of rounds to reach the absorbing state F. With Theorem 8 this will enable us to compute the expected number $E[Z_d]$ of rounds required to reach a legitimate system state. To build the transition matrix P of $\mathcal{M}$ the $d + 4$ states are ordered as $I, C^0, C^1, \ldots, C^d, P, F$. Let Q be the $(d + 3) \times (d + 3)$ upper left submatrix of P. For $s = -1, 0, 1, \ldots, d+1$ denote by Q_s the $(s + 2) \times (s + 2)$ lower right submatrix of Q, i.e. $Q = Q_{d+1}$. Denote by N_s the fundamental matrix of Q_s (notation as introduced in Sect. 5). Let 1_s be the column vector of length $(s + 2)$ whose entries are all 1 and $\epsilon_s = N_s 1_s$. For $s = 0, \ldots, d$, ϵ_s is the expected number of rounds to reach state F from state C^{d-s} and ϵ_{d+1} is the expected number of rounds to reach state F from I, i.e. $\epsilon_{d+1} = E[A_d]$ (Theorem 3.3.5, [17]). Identifying P with C^{d+1} we have $\epsilon_{-1} = 1$.

Lemma 11. *The expected number $E[A_d]$ of rounds to reach F from I is less than 5 and the variance is less than 3.6.*

Proof. Note that $\epsilon_{-1} = 1$ and $\epsilon_0 = \sum_{i=1}^{\infty} \frac{i}{2^i} + 1 = 3$. Q_s and N_s are upper triangle matrices. Let

$$E_i - Q_i = \begin{pmatrix} 1 - a_1 & -a_2 & \ldots & -a_{i+2} \\ 0 & & & \\ \vdots & & E_{i-1} - Q_{i-1} & \\ 0 & & & \end{pmatrix} \quad N_i = \begin{pmatrix} x_1 & x_2 & \ldots & x_{i+2} \\ 0 & & & \\ \vdots & & N_{i-1} & \\ 0 & & & \end{pmatrix}$$

$E_i = (E_i - Q_i)N_i$ gives rise to $(i+2)^2$ equations. Summing up the $i+2$ equations for the first row of E_i results in

$$\epsilon_i = (1-a_1)^{-1}\left(1 + \sum_{l=2}^{i+2} a_l \epsilon_{i+1-l}\right) \tag{1}$$

Hence

$$\epsilon_i = (1-a_1)^{-1}\left(1 + \sum_{l=2}^{i} a_l \epsilon_{i+1-l} + 3a_{i+1} + a_{i+2}\right)$$

for $i > 0$. By Lemma 19 of [24] $\epsilon_i \leq 4$ for $i = -1, 0, 1, \ldots, d$. Hence

$$E[A_d] = \epsilon_{d+1} = 1 + \sum_{l=2}^{d+3} a_l \epsilon_{d+2-l} \leq 1 + 4\sum_{l=2}^{d+3} a_l = 5, \text{ and}$$

$Var[A_d] = ((2N_{d+1} - E_{d+1})1_{d+1} - 1_{d+1}^2)[1] = 2\sum_{i=1}^{d+3} x_i \epsilon_{d+2-i} - \epsilon_{d+1} - \epsilon_{d+1}^2$. □

Lemma 12 [24]**.** *The expected value for the containment time after a memory corruption at node v is at most $\frac{1}{\ln 2} H_{\delta_i(v)} + 11/2$ with variance less than* 7.5.

Theorems 6 and 8, Lemmas 7 and 12 together prove the following Theorem.

Theorem 13. *$\mathcal{A}_{col}$ is a self-stabilizing algorithm for computing a $(\Delta+1)$-coloring in the synchronous model within $O(\log n)$ time with high probability. It uses messages of size $O(\log n)$ and requires $O(\log n)$ storage per node. With respect to memory and message corruption it has contamination radius* 1*. The expected containment time is at most $\frac{1}{\ln 2} H_{\Delta_i} + 11/2$ with variance less than* 7.5.

Corollary 14. *Algorithm $\mathcal{A}_{col}$ has expected containment time $O(1)$ for bounded-independence graphs. For unit disc graphs this time is at most* 8.8.

Proof. For these graphs $\Delta_i \in O(1)$, in particular $\Delta_i \leq 5$ for unit disc graphs. □

8 Conclusion

The analysis of self-stabilizing algorithms is often confined to the stabilization time starting from an arbitrary configuration. In practice the time to recover from a 1-faulty configuration is much more relevant. This paper presents techniques to analyze the containment time of randomized self-stabilizing algorithms for 1-faulty configurations. The execution of an algorithm is modeled as a Markov chain, its complexity is reduced with the lumping technique. The power of this technique is demonstrated by an application to a $\Delta+1$-coloring algorithm.

Acknowledgments. Research was funded by Deutsche Forschungsgemeinschaft DFG (TU 221/6-1).

References

1. Azar, Y., Kutten, S., Patt-Shamir, B.: Distributed error confinement. ACM Trans. Algorithms **6**(3), 48:1–48:23 (2010)
2. Barenboim, L., Elkin, M.: Distributed Graph Coloring: Fundamentals and Recent Developments. Morgan & Claypool Publishers, San Rafael (2013)
3. Beauquier, J., Delaet, S., Haddad, S.: Necessary and sufficient conditions for 1-adaptivity. In: 20th Internatioal Parallel and Distributed Processing Symposium, pp. 10–16 (2006)
4. Crouzen, P., Hahn, E., Hermanns, H., Dhama, A., Theel, O., Wimmer, R., Braitling, B., Becker, B.: Bounded fairness for probabilistic distributed algorithms. In: 11th International Conference Application of Concurrency to System Design, pp. 89–97, June 2011
5. Lee DeVille, R.E., Mitra, S.: Stability of distributed algorithms in the face of incessant faults. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 224–237. Springer, Heidelberg (2009). doi:10.1007/978-3-642-05118-0_16
6. Dolev, S.: Self-Stabilization. MIT Press, Cambridge (2000)
7. Dolev, S., Herman, T.: Superstabilizing protocols for dynamic distributed systems. Chicago J. Theor. Comput. Sci. **4**, 1–40 (1997)
8. Dubois, S., Masuzawa, T., Tixeuil, S.: Bounding the impact of unbounded attacks in stabilization. IEEE Trans. Parallel Distrib. Syst. **23**(3), 460–466 (2012)
9. Duflot, M., Fribourg, L., Picaronny, C.: Randomized finite-state distributed algorithms as Markov chains. In: Welch, J. (ed.) DISC 2001. LNCS, vol. 2180, pp. 240–254. Springer, Heidelberg (2001). doi:10.1007/3-540-45414-4_17
10. Fribourg, L., Messika, S., Picaronny, C.: Coupling and self-stabilization. Distrib. Comput. **18**(3), 221–232 (2006)
11. Gärtner, F.C.: Fundamentals of fault-tolerant distributed computing in asynchronous environments. ACM Comput. Surv. **31**(1), 1–26 (1999)
12. Ghosh, S., Gupta, A.: An exercise in fault-containment: self-stabilizing leader election. Inf. Process. Lett. **59**(5), 281–288 (1996)
13. Ghosh, S., Gupta, A., Herman, T., Pemmaraju, S.: Fault-containing self-stabilizing distributed protocols. Distrib. Comput. **20**(1), 53–73 (2007)
14. Ghosh, S., He, X.: Scalable self-stabilization. J. Parallel Distrib. Comput. **62**(5), 945–960 (2002)
15. Gradinariu, M., Tixeuil, S.: Self-stabilizing vertex coloring of arbitrary graphs. In: 4th International Conference on Principles of Distributed Systems, OPODIS 2000, pp. 55–70 (2000)
16. Johansson, Ö.: Simple distributed $\delta+1$-coloring of graphs. Inf. Process. Lett. **70**(5), 229–232 (1999)
17. Kemeny, J.G., Snell, J.L.: Finite Markov Chains. Springer, Heidelberg (1976)
18. Köhler, S., Turau, V.: Fault-containing self-stabilization in asynchronous systems with constant fault-gap. Distrib. Comput. **25**(3), 207–224 (2012)
19. Kutten, S., Patt-Shamir, B.: Adaptive stabilization of reactive protocols. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 396–407. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30538-5_33
20. Lenzen, C., Suomela, J., Wattenhofer, R.: Local algorithms: self-stabilization on speed. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 17–34. Springer, Heidelberg (2009). doi:10.1007/978-3-642-05118-0_2
21. Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM J. Comput. **15**(4), 1036–1055 (1986)

22. Mitton, N., Fleury, E., Guérin-Lassous, I., Séricola, B., Tixeuil, S.: On fast randomized colorings in sensor networks. In: Proceedings of ICPADS, pp. 31–38. IEEE (2006)
23. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM Society for Industrial and Applied Mathematics, Philadelphia (2000)
24. Turau, V.: Computing the fault-containment time of self-stabilizing algorithms using Markov chains. Technical report, Hamburg University of Techology (2017)
25. Turau, V., Hauck, B.: A fault-containing self-stabilizing (3–2/(delta+1))-approximation algorithm for vertex cover in anonymous networks. Theoret. Comput. Sci. **412**(33), 4361–4371 (2011)
26. Yamashita, M.: Probabilistic self-stabilization and random walks. In: 2013 International Conference on Computing, Networking and Communications (ICNC), pp. 1–7 (2011)