Bounding the Impact of Unbounded Attacks in Stabilization

Swan Dubois, Toshimitsu Masuzawa, and Sébastien Tixeuil

Abstract—Self-stabilization is a versatile approach to fault-tolerance since it permits a distributed system to recover from any transient fault that arbitrarily corrupts the contents of all memories in the system. Byzantine tolerance is an attractive feature of distributed systems that permits to cope with arbitrary malicious behaviors. Combining these two properties proved difficult: it is impossible to contain the spatial impact of Byzantine nodes in a self-stabilizing context for global tasks such as tree orientation and tree construction.

We present and illustrate a new concept of Byzantine containment in stabilization. Our property, called *Strong Stabilization* enables to contain the impact of Byzantine nodes if they actually perform too many Byzantine actions. We derive impossibility results for strong stabilization and present strongly stabilizing protocols for tree orientation and tree construction that are optimal with respect to the number of Byzantine nodes that can be tolerated in a self-stabilizing context.

Index Terms—Byzantine fault, Distributed algorithm, Fault tolerance, Stabilization, Spanning tree construction

1 Introduction

The advent of ubiquitous large-scale distributed systems advocates that tolerance to various kinds of faults and hazards must be included from the very early design of such systems. *Self-stabilization* [3], [5], [14] is a versatile technique that permits forward recovery from any kind of *transient* faults, while *Byzantine Fault-tolerance* [9] is traditionally used to mask the effect of a limited number of *malicious* faults. Making distributed systems tolerant to both transient and malicious faults is appealing yet proved difficult [6], [2], [12] as impossibility results are expected in many cases.

Two main paths have been followed to study the impact of Byzantine faults in the context of self-stabilization.The first one is *Byzantine fault masking*. In completely connected synchronous systems, one of the most studied problems in the context of self-stabilization with Byzantine faults is that of *clock synchronization*. In [1], [6], probabilistic self-stabilizing protocols were proposed for up to one third of Byzantine processors, while in [4], [8] deterministic solutions tolerate up to one fourth and one third of Byzantine processors, respectively. The second one is *Byzantine containment*. For *local* tasks (*i.e.* tasks whose correctness can be checked locally, such as vertex coloring, link coloring, or dining philosophers), the notion of *strict* stabilization was proposed [12], [13], [11]. Strict stabilization guarantees that there exists a *containment radius* outside which the effect of permanent faults is masked. In [12], the authors show that this Byzantine containment scheme is possible only for *local* tasks. As many problems are not local, it turns out that it is impossible to provide strict stabilization for those.

Our Contribution. In this paper, we investigate the possibility of Byzantine containment in a self-stabilizing setting for tasks that are global (*i.e.* for with there exists a causality chain of size r, where r depends on n the size of the network), and focus on two global problems, namely tree orien-

- *S. Dubois is with the LIP6, UPMC Sorbonne Universités & INRIA, France (E-mail: swan.dubois@lip6.fr).*
- *T. Masuzawa is with the Osaka University, Japan (E-mail: masuzawa@ist.osaka-u.ac.jp)¿*
- *S. Tixeuil is with the LIP6, UPMC Sorbonne Universités & Institut Universitaire de France, France (E-mail: sebastien.tixeuil@lip6.fr).*

A preliminary version of this work appears in the proceedings of the 8th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'06), see [10]. This work is supported in part by ANR projects SHAMAN, ALADDIN and SPADES and by Global COE (Centers of Excellence) Program of MEXT and Grant-in-Aid for Scientific Research (B)22300009 of JSPS.

tation and tree construction. As strict stabilization is impossible with such global tasks, we weaken the containment constraint by limiting the number of times that correct processes can be disturbed by Byzantine ones. Recall that strict stabilization requires that processes beyond the containment radius eventually achieve their desired behavior and are never disturbed by Byzantine processes afterwards. We relax this requirement in the following sense: we allow these correct processes beyond the containment radius to be disturbed by Byzantine processes, but only a limited number of times, even if Byzantine nodes take an infinite number of actions.

The main contribution of this paper is to present new possibility results for containing the influence of unbounded Byzantine behaviors. In more details, we define the notion of *strong stabilization* as the novel form of the containment and introduce *disruption times* to quantify the quality of the containment. The notion of strong stabilization is weaker than the strict stabilization but is stronger than the classical notion of self-stabilization (*i.e.* every strongly stabilizing protocol is self-stabilizing, but not necessarily strictly stabilizing). While strict stabilization aims at tolerating an unbounded number of Byzantine processes, we explicitly refer the number of Byzantine processes to be tolerated. A self-stabilizing protocol is (t, c, f)-strongly stabilizing if the subsystem consisting of processes more than c hops away from any Byzantine process is disturbed at most t times in a distributed system with at most f Byzantine processes. Here c denotes the containment radius and t denotes the disruption time.

To demonstrate the possibility and effectiveness of our notion of strong stabilization, we consider *tree construction* and *tree orientation*. It is shown in [12] that there exists no strictly stabilizing protocol with a constant containment radius for these problems. The impossibility result can be extended even when the number of Byzantine processes is upper bounded (by one). In this paper, we provide a $(f\Delta^d, 0, f)$-strongly stabilizing protocol for rooted tree construction, provided that correct processes remain connected, where n (respectively f) is the number of processes (respectively Byzantine processes) and d is the diameter of the subsystem consisting of all correct processes. The containment radius of 0 is obviously optimal. We show that the problem of tree orientation has no constant bound for the containment radius in a tree with two Byzantine processes even when we allow processes beyond the containment radius to be disturbed a finite number of times. Then we consider the case of a single Byzantine process and present a $(\Delta, 0, 1)$-strongly stabilizing protocol for tree orientation, where Δ is the maximum degree of processes. The containment radius of 0 is also optimal. Notice that each process does not need to know the number f of Byzantine processes and that f can be $n-1$ at the worst case. In other words, the algorithm is adaptive in the sense that the disruption times depend on the actual number of Byzantine processes. Both algorithms are also optimal with respect to the number of tolerated Byzantine nodes.

Due to space limitations, this paper contains only the formal definition of strong stabilization and a short presentation of results related to tree construction. The interested reader could find in the associated supplementary file (i) a discussion on relationship between strong stabilization and pseudo stabilization (Section 1), (ii) complete proofs and time complexities of the tree construction (Section 2), and (iii) the complete study of tree orientation (Section 3).

2 PRELIMINARIES

2.1 Distributed System

A *distributed system* $S = (P, L)$ consists of a set $P = \{v_1, v_2, \ldots, v_n\}$ of processes and a set L of bidirectional communication links (simply called links). A link is an unordered pair of distinct processes. A distributed system S can be regarded as a graph whose vertex set is P and whose link set is L, so we use graph terminology to describe a distributed system S.

Processes u and v are called *neighbors* if $(u, v) \in L$. The set of neighbors of a process v is denoted by N_v, and its cardinality (the *degree* of v) is denoted by $\Delta_v (= |N_v|)$. The degree Δ of a distributed system $S = (P, L)$ is defined as $\Delta = \max\{\Delta_v \mid v \in P\}$. We do not assume existence of a unique identifier for each process (that is, the system is anonymous). Instead we assume each process can distinguish its neighbors from each other by locally arranging

them in some arbitrary order: the k-th neighbor of a process v is denoted by $N_v(k)$ $(1 \leq k \leq \Delta_v)$.

Processes can communicate with their neighbors through *link registers*. For each pair of neighboring processes u and v, there are two link registers $r_{u,v}$ and $r_{v,u}$. Message transmission from u to v is realized as follows: u writes a message to link register $r_{u,v}$ and then v reads it from $r_{u,v}$. The link register $r_{u,v}$ is called an *output register* of u and is called an *input register* of v. The set of all output (respesctively input) registers of u is denoted by Out_u (respectively In_u), *i.e.* $Out_u = \{r_{u,v} \mid v \in N_u\}$ and $In_u = \{r_{v,u} \mid v \in N_u\}$.

The variables that are maintained by processes denote process states. Similarly, the values of the variables stored in each link register denote the state of the registers. A process may take actions during the execution of the system. An action is simply a function that is executed in an atomic manner by the process. The actions executed by each process is described by a finite set of guarded actions of the form $\langle \text{guard} \rangle \longrightarrow \langle \text{statement} \rangle$. Each guard of process u is a boolean expression involving the variables of u and its input registers. Each statement of process u is an update of its state and its output/input registers.

A global state of a distributed system is called a *configuration* and is specified by a product of states of all processes and all link registers. We define C to be the set of all possible configurations of a distributed system S. For a process set $R \subseteq P$ and two configurations ρ and ρ', we denote $\rho \overset{R}{\mapsto} \rho'$ when ρ changes to ρ' by executing an action of each process in R simultaneously. Notice that ρ and ρ' can be different only in the states of processes in R and the states of their output registers. For completeness of execution semantics, we should clarify the configuration resulting from simultaneous actions of neighboring processes. The action of a process depends only on its state at ρ and the states of its input registers at ρ, and the result of the action reflects on the states of the process and its output registers at ρ'.

A *schedule* of a distributed system is an infinite sequence of process sets. Let $Q = R^1, R^2, \ldots$ be a schedule, where $R^i \subseteq P$ holds for each i $(i \geq 1)$. An infinite sequence of configurations $e = \rho_0, \rho_1, \ldots$ is called an *execution* from an initial configuration ρ_0 by a schedule Q, if e satisfies $\rho_{i-1} \overset{R^i}{\mapsto} \rho_i$ for each i $(i \geq 1)$. Process actions are executed atomically, and we also assume that a *distributed daemon* schedules the actions of processes, *i.e.* any subset of processes can simultaneously execute their actions.

The set of all possible executions from $\rho_0 \in C$ is denoted by E_{ρ_0}. The set of all possible executions is denoted by E, that is, $E = \bigcup_{\rho \in C} E_\rho$. We consider *asynchronous* distributed systems where we can make no assumption on schedules except that any schedule is *weakly fair*: every process is contained in infinite number of subsets appearing in any schedule.

In this paper, we consider (permanent) *Byzantine faults*: a Byzantine process (*i.e.* a Byzantine-faulty process) can make arbitrary behavior independently from its actions. If v is a Byzantine process, v can repeatedly change its variables and its out put registers arbitrarily.

In asynchronous distributed systems, time is usually measured by *asynchronous rounds* (simply called *rounds*). Let $e = \rho_0, \rho_1, \ldots$ be an execution by a schedule $Q = R^1, R^2, \ldots$. The first round of e is defined to be the minimum prefix of e, $e' = \rho_0, \rho_1, \ldots, \rho_k$, such that $\bigcup_{i=1}^{k} R^i = P'$ where P' is the set of correct processes of P. Round t $(t \geq 2)$ is defined recursively, by applying the above definition of the first round to $e'' = \rho_k, \rho_{k+1}, \ldots$. Intuitively, every correct process has a chance to update its state in every round.

2.2 Self-Stabilizing Protocol Resilient to Byzantine Faults

Problems considered in this paper are so-called *static problems*, *i.e.* they require the system to find static solutions. For example, the spanning-tree construction problem is a static problem, while the mutual exclusion problem is not. Some static problems can be defined by a *specification predicate* (shortly, specification), $spec(v)$, for each process v: a configuration is a desired one (with a solution) if every process satisfies $spec(v)$. A specification $spec(v)$ is a boolean expression on variables of P_v $(\subseteq P)$ where P_v is the set of processes whose variables appear in $spec(v)$. The variables appearing in the specification are called *output variables* (shortly, *O-variables*). In what follows, we consider a static problem defined by specification $spec(v)$.

A *self-stabilizing protocol* is a protocol that eventually reaches a *legitimate configuration*, where $spec(v)$ holds at every process v, regardless of the initial configuration. Once it reaches a legitimate configuration, every process v never changes its O-variables and always satisfies $spec(v)$. From this definition, a self-stabilizing protocol is expected to tolerate any number and any type of transient faults since it can eventually recover from any configuration affected by the transient faults. However, the recovery from any configuration is guaranteed only when every process correctly executes its action from the configuration, *i.e.*, we do not consider existence of permanently faulty processes.

When (permanent) Byzantine processes exist, Byzantine processes may not satisfy $spec(v)$. In addition, correct processes near the Byzantine processes can be influenced and may be unable to satisfy $spec(v)$. Nesterenko and Arora [12] define a *strictly stabilizing protocol* as a self-stabilizing protocol resilient to unbounded number of Byzantine processes.

Given an integer c, a *c-correct process* is a process defined as follows.

Definition 1 (c-correct process): A process is c-correct if it is correct (*i.e.* not Byzantine) and located at distance more than c from any Byzantine process.

Definition 2 ((c, f)-containment): A configuration ρ is (c, f)-*contained* for specification $spec$ if, given at most f Byzantine processes, in any execution starting from ρ, every c-correct process v always satisfies $spec(v)$ and never changes its O-variables.

The parameter c of Definition 2 refers to the *containment radius* defined in [12]. The parameter f refers explicitly to the number of Byzantine processes, while [12] dealt with unbounded number of Byzantine faults (that is $f \in \{0 \dots n\}$).

Definition 3 ((c, f)-strict stabilization): A protocol is (c, f)-*strictly stabilizing* for specification $spec$ if, given at most f Byzantine processes, any execution $e = \rho_0, \rho_1, \dots$ contains a configuration ρ_i that is (c, f)-contained for $spec$.

An important limitation of the model of [12] is the notion of r-*restrictive* specifications. Intuitively, a specification is r-restrictive if it prevents combinations of states that belong to two processes u and v that are at least r hops away. An important consequence related to Byzantine tolerance is that the containment radius of protocols solving those specifications is at least r. For some problems, such as the spanning tree construction we consider in this paper, r can not be bounded to a constant. We can show that there exists no $(o(n), 1)$-strictly stabilizing protocol for the spanning tree construction.

To circumvent the impossibility result, we define a weaker notion than the strict stabilization. Here, the requirement to the containment radius is relaxed, *i.e.* there may exist processes outside the containment radius that invalidate the specification predicate, due to Byzantine actions. However, the impact of Byzantine triggered action is limited in times: the set of Byzantine processes may only impact the subsystem consisting of processes outside the containment radius a bounded number of times, even if Byzantine processes execute an infinite number of actions.

From the states of c-correct processes, *c-legitimate configurations* and *c-stable configurations* are defined as follows.

Definition 4 (c-legitimate configuration): A configuration ρ is c-legitimate for *spec* if every c-correct process v satisfies $spec(v)$.

Definition 5 (c-stable configuration): A configuration ρ is c-stable if every c-correct process never changes the values of its O-variables as long as Byzantine processes make no action.

Roughly speaking, the aim of self-stabilization is to guarantee that a distributed system eventually reaches a c-legitimate and c-stable configuration. However, a self-stabilizing system can be disturbed by Byzantine processes after reaching a c-legitimate and c-stable configuration. The *c-disruption* represents the period where c-correct processes are disturbed by Byzantine processes and is defined as follows

Definition 6 (c-disruption): A portion of execution $e = \rho_0, \rho_1, \dots, \rho_t$ $(t > 1)$ is a c-disruption if and only if the following holds: (i) e is finite, (ii) e contains at least one action of a c-correct process for changing the value of an O-variable, (iii) ρ_0 is c-legitimate for *spec* and c-stable, and (iv) ρ_t is the first configuration after ρ_0 such that ρ_t is c-legitimate for *spec* and c-stable.

Now we can define a self-stabilizing protocol such that Byzantine processes may only impact the subsystem consisting of processes outside the containment radius a bounded number of times, even if Byzantine processes execute an infinite number

of actions.

Definition 7 ((t,k,c,f)-time contained configuration): A configuration ρ_0 is (t,k,c,f)-time contained for *spec* if given at most f Byzantine processes, the following properties are satisfied: (i) ρ_0 is c-legitimate for *spec* and c-stable, (ii) every execution starting from ρ_0 contains a c-legitimate configuration for *spec* after which the values of all the O-variables of c-correct processes remain unchanged (even when Byzantine processes make actions repeatedly and forever), (iii) every execution starting from ρ_0 contains at most t c-disruptions, and (iv) every execution starting from ρ_0 contains at most k actions of changing the values of O-variables for each c-correct process.

Definition 8 ((t,c,f)-strongly stabilizing protocol): A protocol A is (t,c,f)-strongly stabilizing if and only if starting from any arbitrary configuration, every execution involving at most f Byzantine processes contains a (t,k,c,f)-time contained configuration that is reached after at most l rounds. Parameters l and k are respectively the (t,c,f)-stabilization time and the (t,c,f)-process-disruption time of A.

Note that a (t,k,c,f)-time contained configuration is a (c,f)-contained configuration when $t = k = 0$, and thus, (t,k,c,f)-time contained configuration is a generalization (relaxation) of a (c,f)-contained configuration. Thus, a strongly stabilizing protocol is weaker than a strictly stabilizing one (as processes outside the containment radius may take incorrect actions due to Byzantine influence). However, a strongly stabilizing protocol is stronger than a classical self-stabilizing one (that may never meet their specification in the presence of Byzantine processes).

The parameters t, k and c are introduced to quantify the strength of fault containment, we do not require each process to know the values of the parameters. Actually, the protocols proposed in this paper assume no knowledge on the parameters.

3 Strongly-Stabilizing Spanning Tree Construction

3.1 Problem Definition

In this section, we consider only distributed systems in which a given process r is distinguished as the root of the tree.

For *spanning tree construction*, each process v has an O-variable $prnt_v$ to designate a neighbor as its parent. Since processes have no identifiers, $prnt_v$ actually stores k ($\in \{1,\ 2, \ldots,\ \Delta_v\}$) to designate its k-th neighbor as its parent. No neighbor is designated as the parent of v when $prnt_v = 0$ holds. For simplicity, we use $prnt_v = k$ ($\in \{1,\ 2, \ldots,\ \Delta_v\}$) and $prnt_v = u$ (where u is the k-th neighbor of $v \in N_v(k)$) interchangeably, and $prnt_v = 0$ and $prnt_v = \perp$ interchangeably.

The goal of spanning tree construction is to set $prnt_v$ of every process v to form a rooted spanning tree, where $prnt_r = 0$ should hold for the root process r.

We consider Byzantine processes that can behave arbitrarily. The faulty processes can behave as if they were any internal processes of the spanning tree, or even as if they were the root processes. The first restriction we make on Byzantine processes is that we assume the root process r can start from an arbitrary state, but behaves correctly according to a protocol. Another restriction on Byzantine processes is that we assume that all the correct processes form a connected subsystem; Byzantine processes never partition the system.

It is impossible, for example, to distinguish the (real) root r from the faulty processes behaving as the root, we have to allow that a spanning forest (consisting of multiple trees) is constructed, where each tree is rooted with a root, correct or faulty one.

We define the specification predicate $spec(v)$ of the tree construction as follows. $spec(v)$ is the predicate $(prnt_v = 0) \wedge (level_v = 0)$ if v is the root r and $(prnt_v \in \{1, \ldots,\ \Delta_v\}) \wedge ((level_v = level_{prnt_v} + 1) \vee (prnt_v$ is Byzantine$))$ otherwise.

Notice that $spec(v)$ requires that a spanning tree is constructed at any 0-legitimate configuration, when no Byzantine process exists.

Figure 1 shows an example of 0-legitimate configuration with Byzantine processes. The arrow attached to each process points the neighbor designated as its parent.

3.2 Protocol ss-ST

In many self-stabilizing tree construction protocols (see the survey of [7]), each process checks locally the consistence of its $level$ variable with respect to the one of its neighbors. When it detects an

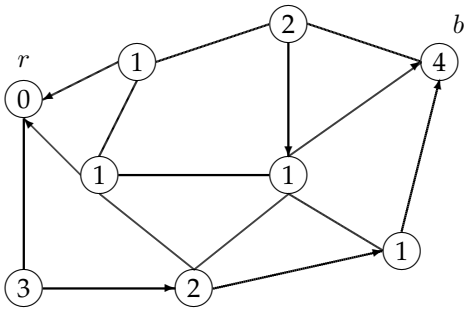

Fig. 1. A legitimate configuration for spanning tree construction (numbers denote the level of processes). r is the (real) root and b is a Byzantine process which acts as a (fake) root.

inconsistency, it changes its $prnt$ variable in order to choose a "better" neighbor. The notion of "better" neighbor is based on the global desired property on the tree (*e.g.* shortest path tree, minimun spanning tree...).

When the system may contain Byzantine processes, they may disturb their neighbors by providing alternatively "better" and "worse" states. The key idea of protocol ss-ST to circumvent this kind of perturbation is the following: when a correct process detects a local inconsistency, it does not choose a "better" neighbor but it chooses another neighbor according to a round robin order (along the set of its neighbor).

Figure 2 presents our strongly-stabilizing spanning tree construction protocol ss-ST that can tolerate any number of Byzantine processes other than the root process (providing that the subset of correct processes remains connected). These assumptions are necessary since a Byzantine root or a set of Byzantine processes that disconnects the set of correct processes may disturb all the tree infinitely often. Then, it is impossible to provide a (t, k, f)-strongly stabilizing protocol for any finite integer t.

The protocol is composed of three rules. Only the root can execute the first one (GA0). This rule sets the root in a legitimate state if it is not the case. Non-root processes may execute the two other rules (GA1 and GA2). The rule GA1 is executed when the state of a process is not legitimate. Its execution leads the process to choose a new parent and to compute its local state in function of this new parent. The last rule (GA2) is enabled when a process is in a legitimate state but there exists an inconsistence between its variables and its shared registers. The execution of this rule leads the process to compute the consistent values for all its shared registers.

3.3 Proof of Strong Stabilization of ss-ST

In this section, we provide a sketch of the proof of the strong stabilization of ss-ST.

We cannot make any assumption on the initial values of register variables. But, we can observe that if an output register of a correct process has inconsistent values with the process variables then this process is enabled by a rule of ss-ST. By fairness assumption, any such process takes a step in a finite time.

Once a correct process v executes one of its action, variables of its output registers have values consistent with the process variables: r-$prnt_{v,prnt_v} = true$, r-$prnt_{v,w} = false$ $(w \in N_v - \{prnt_v\})$, and r-$level_{v,w} = level_v$ $(w \in N_v)$ hold.

Consequently, we can assume in the following that all the variables of output registers of every correct process have consistent values with the process variables.

We denote by $\mathcal{LC}$ the following set of configurations: $\mathcal{LC} = \Big\{\rho \in C \Big| (prnt_r = 0) \wedge (level_r = 0) \wedge \big(\forall v \in V - (B \cup \{r\}), (prnt_v \in \{1, \ldots, \Delta_v\}) \wedge (level_v = level_{prnt_v} + 1)\big)\Big\}$.

We interest now on properties of configurations of $\mathcal{LC}$. We can observe that any configuration of $\mathcal{LC}$ is 0-legitimate and that any correct process is not enabled in such a configuration. We can deduce:

Lemma 1: Any configuration of $\mathcal{LC}$ is 0-legitimate and 0-stable.

We can observe that there exists some 0-legitimate configurations which not belong to $\mathcal{LC}$ (for example the one of Figure 2).

To prove the convergence of ss-ST to $\mathcal{LC}$ starting from any configuration, we prove that any execution reaches a configuration in which every correct process is disabled. We show that the root r executes its rule at most once. Then, any of its correct neighbor v executes its rule at most Δ_v before pointing to r. After that, v is never enabled. By repeating the reasoning, we obtain:

Lemma 2: Given at most $n - 1$ Byzantine processes, for any initial configuration ρ_0 and any

constants of process v

$\Delta_v =$ the degree of v;

$N_v =$ the set of neighbors of v;

variables of process v

$prnt_v \in \{0, 1, 2, \ldots, \Delta_v\}$: integer; // $prnt_v = 0$ if v has no parent,

// $prnt_v = k \in \{1, 2, \ldots, \Delta_v\}$ if $N_v[k]$ is the parent of v.

$level_v$: integer; // distance from the root.

variables in shared register $r_{v,u}$

r-prnt$_{v,u}$: boolean; // *r-prnt*$_{v,u}$ =*true* iff u is a parent of v.

r-level$_{v,u}$: integer; // the value of $level_v$

predicates

$pred_0 : prnt_v \neq 0$ or $level_v \neq 0$ or $\exists w \in N_v, [(\textit{r-prnt}_{v,w}, \textit{r-level}_{v,w}) \neq (\textit{false}, 0)]$

$pred_1 : prnt_v \notin \{1,\ 2, \ldots,\ \Delta_v\}$ or $level_v \neq \textit{r-level}_{prnt_v,v} + 1$

$pred_2 : (\textit{r-prnt}_{v,prnt_v}, \textit{r-level}_{v,prnt_v}) \neq (\textit{true}, level_v)$

or $\exists w \in N_v - \{prnt_v\}, [(\textit{r-prnt}_{v,w}, \textit{r-level}_{v,w}) \neq (\textit{false}, level_v)]$

atomic action of the root $v = r$ // represented in form of guarded action

GA0 : $pred_0 \longrightarrow prnt_v := 0;\ level_v := 0;$ for each $w \in N_v$ do $(\textit{r-prnt}_{v,w}, \textit{r-level}_{v,w}) := (\textit{false}, 0);$

atomic actions of $v \neq r$ // represented in form of guarded actions

GA1 : $pred_1 \longrightarrow\ prnt_v := next_v(prnt_v)$ where $next_v(k) = (k \bmod \Delta_v) + 1;$

$level_v := \textit{r-level}_{prnt_v,v} + 1;\ (\textit{r-prnt}_{v,prnt_v}, \textit{r-level}_{v,prnt_v}) := (\textit{true}, level_v);$

for each $w \in N_v - \{prnt_v\}$ do $(\textit{r-prnt}_{v,w}, \textit{r-level}_{v,w}) := (\textit{false}, level_v);$

GA2 : $\neg pred_1$ and $pred_2 \longrightarrow\ (\textit{r-prnt}_{v,prnt_v}, \textit{r-level}_{v,prnt_v}) := (\textit{true}, level_v);$

for each $w \in N_v - \{prnt_v\}$ do $(\textit{r-prnt}_{v,w}, \textit{r-level}_{v,w}) := (\textit{false}, level_v);$

Fig. 2. Protocol ss-ST (actions of process v)

execution $e = \rho_0, \rho_1, \ldots$ starting from ρ_0, there exists a configuration ρ_i such that $\rho_i \in \mathcal{LC}$.

It remains to prove that any execution starting from an arbitrary configuration of $\mathcal{LC}$ contains only a finite number of 0-disruptions. First, we prove by induction on the distance δ between the root and a correct process v in the subgraph of correct processes that v executes its action at most Δ^δ times. We say that a Byzantine process b deceive a correct neighbor v in the step $\rho \mapsto \rho'$ if the state of b makes the guard of an action of v true in ρ and if v executes this action in this step. As a 0-disruption can be caused only by an action of a Byzantine process from a legitimate configuration, we can bound the number of 0-disruptions by counting the total number of times that correct processes are deceived of neighboring Byzantine processes. If a 0-correct v is deceived by a Byzantine neighbor b, it takes necessarily Δ_v actions before being deceiving again by b (recall that we use a round-robin policy for $prnt_v$). As any 0-correct process v takes at most Δ^d actions in any execution, v can be deceived by a given Byzantine neighbor at most Δ^{d-1} times. A Byzantine process can have at most Δ neighboring correct processes and thus can deceive correct processes at most $\Delta \times \Delta^{d-1} = \Delta^d$ times. We have at most f Byzantine processes, so the total number of times that correct processes are deceived by neighboring Byzantine processes is $f\Delta^d$. Hence, the number of 0-disruption in e is bounded by $f\Delta^D$. Now, assume by contradiction that there exists an infinite 0-disruption $d = \rho_i, \ldots$ in e. This implies that for all $j \geq i$, ρ_j is not in $\mathcal{LC}$, which contradicts Lemma 2. Then, we can state:

Lemma 3: Any configuration in $\mathcal{LC}$ is a $(f\Delta^d, \Delta^d, 0, f)$-time contained configuration of the spanning tree construction, where f is the number of Byzantine processes and d is the diameter of the subsystem consisting of all the correct processes.

Hence the following theorem.

Theorem 1 (Strong-stabilization): Protocol ss-ST is

a $(f\Delta^d, 0, f)$-strong stabilizing protocol for the spanning tree construction, where f is the number of Byzantine processes and d is the diameter of the subsystem consisting of all the correct processes.

4 Concluding Remarks

We introduced the notion of strong stabilization, a property that permits self-stabilizing protocols to contain Byzantine behaviors for tasks where strict stabilization is impossible. In strong stabilization, only the first Byzantine actions that are performed by a Byzantine process may disturb the system. If the Byzantine node does not execute Byzantine actions, but only correct actions, its existence remains unnoticed by the correct processes. So, by behaving properly, the Byzantine node may have the system disturbed arbitrarily far in the execution. By contrast, if the Byzantine node executes many Byzantine actions at the beginning of the execution, there exists a time after which those Byzantine actions have no impact on the system. As a result, the faster an attacker spends its Byzantine actions, the faster the system become resilient to subsequent Byzantine actions. An interesting trade-off appears: the more actually Byzantine actions are performed, the faster the stabilization of our protocols is (since the number of steps performed by correct processes in response to Byzantine disruption is independent from the number of Byzantine actions). Our work raises several important open questions.

First, is there a trade-off between the number of perturbations Byzantine nodes can cause and the containment radius ? In this paper, we strove to obtain optimal containment radius in strong stabilization, but it is likely that some problems do not allow strong stabilization with containment radius 0. It is then important to characterize the difference in containment radius when the task to be solved is "harder" than *e.g.* tree construction.

Second, is there a trade-off between the total number of perturbations Byzantine nodes can cause and the number of Byzantine nodes, that is, is a single Byzantine node more effective to harm the system than a team of Byzantine nodes, considering the same total number of Byzantine actions ? A first step in this direction was recently taken by [15], where Byzantine actions are assumed to be upper bounded, for the (global) problem of leader election. Their result hints that only Byzantine actions are relevant, independently of the number of processes that perform them. It is thus interesting to see if the result still holds in the case of potentially infinite number of Byzantine actions.

References

[1] Michael Ben-Or, Danny Dolev, and Ezra N. Hoch. Fast self-stabilizing byzantine tolerant digital clock synchronization. In *ACM Symposium on Principles of Distributed Computing (PODC 2008)*, pages 385–394, 2008.

[2] Ariel Daliot and Danny Dolev. Self-stabilization of byzantine protocols. In *Self-Stabilizing Systems (SSS 2005)*, pages 48–67, 2005.

[3] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communication of ACM*, 17(11):643–644, 1974.

[4] Danny Dolev and Ezra N. Hoch. On self-stabilizing synchronous actions despite byzantine attacks. In *International Symposium on Distributed Computing (DISC 2007)*, pages 193–207, 2007.

[5] Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

[6] Shlomi Dolev and Jennifer L. Welch. Self-stabilizing clock synchronization in the presence of byzantine faults. *Journal of ACM*, 51(5):780–799, 2004.

[7] Felix C. Gärtner. A survey of self-stabilizing spanning-tree construction algorithms. Technical report ic/2003/38, EPFL, 2003.

[8] Ezra N. Hoch, Danny Dolev, and Ariel Daliot. Self-stabilizing byzantine digital clock synchronization. In *Self-Stabilizing Systems (SSS 2006)*, pages 350–362, 2006.

[9] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[10] Toshimitsu Masuzawa and Sébastien Tixeuil. Bounding the impact of unbounded attacks in stabilization. In *Self-Stabilizing Systems (SSS 2006)*, pages 440–453, 2006.

[11] Toshimitsu Masuzawa and Sébastien Tixeuil. Stabilizing link-coloration of arbitrary networks with unbounded byzantine faults. *International Journal of Principles and Applications of Information Science and Technology (PAIST)*, 1(1):1–13, 2007.

[12] Mikhail Nesterenko and Anish Arora. Tolerance to unbounded byzantine faults. In *Symposium on Reliable Distributed Systems (SRDS 2002)*, page 22, 2002.

[13] Yusuke Sakurai, Fukuhito Ooshita, and Toshimitsu Masuzawa. A self-stabilizing link-coloring protocol resilient to byzantine faults in tree networks. In *International Conference on Principles of Distributed Systems (OPODIS 2004)*, pages 283–298, 2005.

[14] Sébastien Tixeuil. *Algorithms and Theory of Computation Handbook, Second Edition*, chapter Self-stabilizing Algorithms, pages 26.1–26.45. Chapman & Hall/CRC Applied Algorithms and Data Structures. CRC Press, Taylor & Francis Group, 2009.

[15] Yukiko Yamauchi, Toshimitsu Masuzawa, and Doina Bein. Adaptive containment of time-bounded byzantine faults. In *Self-Stabilizing Systems (SSS 2010)*, pages 126–140, 2010.

AUTHOR'S BIOGRAPHIES

Swan Dubois is currently a Ph.D student in computer science at the University Pierre and Marie Curie - Paris 6 and at the INRIA (France). Its research interests are fault-tolerance in distributed systems (especially self-stabilization) and graph theory.

Toshimitsu Masuzawa received the B.E., M.E. and D.E. degrees in computer science from Osaka University in 1982, 1984 and 1987. He had worked at Osaka University during 1987–1994, and was an associate professor of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST) during 1994–2000. He is now a professor of Graduate School of Information Science and Technology, Osaka University. He was also a visiting associate professor of Department of Computer Science, Cornell University between 1993-1994. His research interests include distributed algorithms, parallel algorithms and graph theory. He is a member of ACM, IEEE, IEICE and IPSJ.

Sébastien Tixeuil is a full professor at the University Pierre & Marie Curie - Paris 6 (France) and Institut Universitaire de France, where he leads the NPA research group. He received his Ph.D. from University of Paris Sud-XI in 2000. His research interests include fault and attack tolerance in dynamic networks and systems.