On Byzantine Containment Properties of the $min + 1$ Protocol

Swan Dubois*† Toshimitsu Masuzawa‡ 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.

We consider the well known problem of constructing a breadth-first spanning tree in this context. Combining these two properties proves difficult: we demonstrate that it is impossible to contain the impact of Byzantine nodes in a strictly or strongly stabilizing manner. We then adopt the weaker scheme of *topology-aware strict stabilization* and we present a similar weakening of strong stabilization. We prove that the classical $min + 1$ protocol has optimal Byzantine containment properties with respect to these criteria.

Keywords Byzantine fault, Distributed protocol, 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* [4, 6, 19] is a versatile technique that permits forward recovery from any kind of *transient* faults, while *Byzantine Fault-tolerance* [14] 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 [7, 2, 17] 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:

- *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, 7], probabilistic self-stabilizing protocols were proposed for up to one third of Byzantine processes, while in [5, 12] deterministic solutions tolerate up to one fourth and one third of Byzantine processes, respectively.

- *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 [17, 18, 16]. Strict stabilization guarantees that there exists a

*Université Pierre et Marie Curie & INRIA, France, swan.dubois@lip6.fr

†Contact author, Telephone: 33 1 44 27 87 67, Postal address: LIP6, Case 26/00-225, Campus Jussieu, 4 place Jussieu, 75252 Paris Cedex 5, France

‡Osaka University, Japan, masuzawa@ist.osaka-u.ac.jp

§Université Pierre et Marie Curie & INRIA, France, sebastien.tixeuil@lip6.fr

containment radius outside which the effect of permanent faults is masked, provided that the problem specification makes it possible to break the causality chain that is caused by the faults. 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.* there exists a causality chain of size r, where r depends on n the size of the network), and focus on a global problem, namely breadth-first spanning tree construction. A good survey on self-stabilizing solutions to this problem can be found in [11]. In particular, one of the simplest solution is known under the name of $min + 1$ protocol (see [13]). This name is due to the construction of the protocol itself. Each process has two variables: one pointer to its parent in the tree and one level in this tree. The protocol is reduced to the following rule: each process chooses as its parent the neighbor which has the smallest level (*min* part) and updates its level in consequence (+1 part). [13] proves that this protocol is self-stabilizing. In this paper, we propose a complete study of Byzantine containment properties of this protocol.

In a first time, we study space Byzantine containment properties of this protocol. As strict stabilization is impossible with such global tasks (see [17]), we use the weaker scheme of *topology-aware strict stabilization* (see [9]). In this scheme, we weaken the containment constraint by relaxing the notion of containment radius to containment area, that is Byzantine processes may disturb infinitely often a set of processes which depends on the topology of the system and on the location of Byzantine processes. We show that the $min+1$ protocol has optimal containment area with respect to topology-aware strict stabilization.

In a second time, we study time Byzantine containment properties of this protocol using the concept of *strong stabilization* (see [15, 8]). We first show that it is impossible to find a strongly stabilizing solution to the BFS tree construction problem. It is why we weaken the concept of strong stabilization using the notion of containment area to obtain *topology-aware strong stabilization*. We show then that the $min+1$ protocol has also optimal containment area with respect to topology-aware strong stabilization.

2 Distributed System

A *distributed system* $S = (P, L)$ consists of a set $P = \{v_1, v_2, \dots, v_n\}$ 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. We use the following notations: $n = |P|$ and $m = |L|$.

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. 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 $N_v(k)$ $(1 \leq k \leq \Delta_v)$.

In this paper, we consider distributed systems of arbitrary topology. We assume that a single process is distinguished as a *root*, and all the other processes are identical.

We adopt the *shared state model* as a communication model in this paper, where each process can directly read the states of its neighbors.

The variables that are maintained by processes denote process states. 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 neighbors.

A global state of a distributed system is called a *configuration* and is specified by a product of states of all processes. 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. 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 neighbors at ρ, and the result of the action reflects on the state of the process at ρ'.

We say that a process is *enabled* in a configuration ρ if the guard of at least one of its actions is evaluated as true in 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. We say that the daemon is *central* if it schedules action of only one process at any step.

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 *fair*: a process which is infinitely often enabled in an execution can not be never activated in this execution.

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 arbitrarily.

3 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* ([4]) 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 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.

3.1 Strict stabilization

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 [17] 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 [17]. The parameter f refers explicitly to the number of Byzantine processes, while [17] 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 [17] 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 breadth-first search (BFS) spanning tree construction we consider in this paper, r can not be bounded by a constant. In consequence, we can show that there exists no $(c, 1)$-strictly stabilizing protocol for the breadth-first search (BFS) spanning tree construction for any (finite) integer c.

3.2 Strong stabilization

To circumvent the impossibility result, [15] defines 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 processes outside the containment radius a bounded number of times, even if Byzantine processes execute an infinite number of actions.

In the following of this section, we recall the formal definition of strong stabilization adopted in [8]. 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 $e = \rho_0, \rho_1, \ldots, \rho_t$ ($t > 1$) is a c-disruption if and only if the following holds:*

1. *e is finite,*
2. *e contains at least one action of a c-correct process for changing the value of an O-variable,*
3. *ρ_0 is c-legitimate for* spec *and c-stable, and*
4. *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 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:*

1. *ρ_0 is c-legitimate for* spec *and c-stable,*
2. *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),*
3. *every execution starting from ρ_0 contains at most t c-disruptions, and*
4. *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 times 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.

4 Topology-aware Byzantine resilience

4.1 Topology-aware strict stabilization

In Section 3.1, we saw that there exist a number of impossibility results on strict stabilization due to the notion of r-restrictives specifications. To circumvent this impossibility result, we describe here a weaker notion than the strict stabilization: the *topology-aware strict stabilization* (denoted by TA strict stabilization for short) introduced by [9]. Here, the requirement to the containment radius is relaxed, *i.e.* the set of processes which may be disturbed by Byzantine ones is not reduced to the union of c-neighborhood of Byzantine processes but can be defined depending on the graph topology and Byzantine processes location.

In the following, we give formal definition of this new kind of Byzantine containment. From now, B denotes the set of Byzantine processes and S_B (which is function of B) denotes a subset of V (intuitively, this set gathers all processes which may be disturbed by Byzantine processes).

Definition 9 (S_B-correct node) *A node is* S_B-correct *if it is a correct node (*i.e. *not Byzantine) which not belongs to S_B.*

Definition 10 (S_B-legitimate configuration) *A configuration ρ is* S_B-legitimate *for spec if every S_B-correct node v is legitimate for spec (*i.e. *if spec(v) holds).*

Definition 11 ((S_B, f)-topology-aware containment) *A configuration ρ_0 is* (S_B, f)-topology-aware contained *for specification spec if, given at most f Byzantine processes, in any execution $e = \rho_0, \rho_1, \ldots$, every configuration is S_B-legitimate and every S_B-correct process never changes its O-variables.*

The parameter S_B of Definition 11 refers to the *containment area*. Any process which belongs to this set may be infinitely disturbed by Byzantine processes. The parameter f refers explicitly to the number of Byzantine processes.

Definition 12 ((S_B, f)-topology-aware strict stabilization) *A protocol is* (S_B, f)-topology-aware strictly stabilizing *for specification spec if, given at most f Byzantine processes, any execution $e = \rho_0, \rho_1, \ldots$ contains a configuration ρ_i that is (S_B, f)-topology-aware contained for spec.*

Note that, if B denotes the set of Byzantine processes and $S_B = \left\{ v \in V | \min_{b \in B} (d(v, b)) \leq c \right\}$, then a (S_B, f)-topology-aware strictly stabilizing protocol is a (c, f)-strictly stabilizing protocol. Then, the concept of topology-aware strict stabilization is a generalization of the strict stabilization. However, note that a TA strictly stabilizing protocol is stronger than a classical self-stabilizing protocol (that may never meet their specification in the presence of Byzantine processes).

The parameter S_B is introduced to quantify the strength of fault containment, we do not require each process to know the actual definition of the set. Actually, the protocol proposed in this paper assumes no knowledge on the parameter.

4.2 Topology-aware strong stabilization

Similarly to topology-aware strict stabilization, we can weaken the notion of strong stabilization using the notion of containment area. Then, we obtain the following definition:

Definition 13 (**S_B-stable configuration**) *A configuration ρ is S_B-stable if every S_B-correct process never changes the values of its O-variables as long as Byzantine processes make no action.*

Definition 14 (**S_B-TA-disruption**) *A portion of execution $e = \rho_0, \rho_1, \ldots, \rho_t$ $(t > 1)$ is a S_B-TA-disruption if and only if the followings hold:*

1. *e is finite,*
2. *e contains at least one action of a S_B-correct process for changing the value of an O-variable,*
3. *ρ_0 is S_B-legitimate for spec and S_B-stable, and*
4. *ρ_t is the first configuration after ρ_0 such that ρ_t is S_B-legitimate for spec and S_B-stable.*

Definition 15 (**(t, k, S_B, f)-TA time contained configuration**) *A configuration ρ_0 is (t, k, S_B, f)-TA time contained for* spec *if given at most f Byzantine processes, the following properties are satisfied:*

1. *ρ_0 is S_B-legitimate for* spec *and S_B-stable,*
2. *every execution starting from ρ_0 contains a S_B-legitimate configuration for* spec *after which the values of all the O-variables of S_B-correct processes remain unchanged (even when Byzantine processes make actions repeatedly and forever),*
3. *every execution starting from ρ_0 contains at most t S_B-TA-disruptions, and*
4. *every execution starting from ρ_0 contains at most k actions of changing the values of O-variables for each S_B-correct process.*

Definition 16 (**(t, S_B, f)-TA strongly stabilizing protocol**) *A protocol A is (t, S_B, f)-TA strongly stabilizing if and only if starting from any arbitrary configuration, every execution involving at most f Byzantine processes contains a (t, k, S_B, f)-TA-time contained configuration that is reached after at most l actions of each S_B-correct node. Moreover, S_B-legitimate configurations are closed by actions of A. Parameters l and k are respectively the (t, S_B, f)-stabilization time and the (t, S_B, f)-process-disruption time of A.*

5 BFS Spanning Tree Construction

In this section, we are interested in the problem of BFS spanning tree construction. That is, the system has a distinguished process called the root (and denoted by r) and we want to obtain a BFS spanning tree rooted to this root. We made the following hypothesis: the root r is never Byzantine.

To solve this problem, each process v has two O-variables: the first is $prnt_v \in N_v \cup \{\perp\}$ which is a pointer to the neighbor that is designated to be the parent of v in the BFS tree and the second is $level_v \in \{0, \ldots, D\}$ which stores the depth (the number of hops from the root) of v in this tree. Obviously, Byzantine process may disturb (at least) their neighbors. For example, a Byzantine process may act as the root. It is why the specification of the BFS tree construction we adopted states in fact that there exists a BFS spanning forest such that any root of this forest is either the real root of the system or a Byzantine process. More formally, we use the following specification of the problem.

Definition 17 (BFS path) *A path* $(v_0, \dots, v_k)$ $(k \geq 1)$ *of* S *is a* BFS path *if and only if:*

1. $prnt_{v_0} = \perp$, $level_{v_0} = 0$, *and* $v_0 \in B \cup \{r\}$,
2. $\forall i \in \{1, \dots, k\}, prnt_{v_i} = v_{i-1}$ *and* $level_{v_i} = level_{v_{i-1}} + 1$, *and*
3. $\forall i \in \{1, \dots, k\}, level_{v_{i-1}} = \min_{u \in N_{v_i}} \{level_u\}$.

We define the specification predicate $spec(v)$ of the BFS spanning tree construction as follows.

$$spec(v) : \begin{cases} prnt_v = \perp \text{ and } level_v = 0 \text{ if } v \text{ is the root } r \\ \text{there exists a BFS path } (v_0, \dots, v_k) \text{ such that } v_k = v \text{ otherwise} \end{cases}$$

In the case where any process is correct, note that *spec* implies the existence of a BFS spanning tree rooted to the real root. The well-known $min + 1$ protocol solves this problem in a self-stabilizing way (see [13]). In the following of this section, we assume that some processes may be Byzantine and we study the Byzantine containment properties of this protocol. We show that this self-stabilizing protocol has moreover optimal Byzantine containment properties.

In more details, we prove first that there exists neither strictly nor strongly stabilizing solution to the BFS spanning tree construction (see Theorems 1 and 2). Then, we demonstrate in Theorems 3 and 4 that the $min + 1$ protocol is both (S_B, f)-TA strictly and (t, S_B^*, f)-TA strongly stabilizing where $f \leq n - 1$, $t = 2m$, and

$$\begin{aligned} S_B &= \left\{ v \in V \,\middle|\, \min_{b \in B} (d(v,b)) \leq d(r,v) \right\} \\ S_B^* &= \left\{ v \in V \,\middle|\, \min_{b \in B} (d(v,b)) < d(r,v) \right\} \end{aligned}$$

Figure 1 provides an example of these containment areas. Finally, we show that these containment areas are in fact optimal (see Theorem 5 and 6).

5.1 Impossibility results

Theorem 1 *Even under the central daemon, there exists no* $(c, 1)$*-strictly stabilizing protocol for BFS spanning tree construction where* c *is any (finite) integer.*

Proof This result is a direct application of Theorem 4 of [17] (note that the specification of BFS tree construction is D-restrictive in the worst case where D is the diameter of the system). □

Theorem 2 *Even under the central daemon, there exists no* $(t, c, 1)$*-strongly stabilizing protocol for BFS spanning tree construction where* t *and* c *are any (finite) integers.*

Proof Let t and c be (finite) integers. Assume that there exists a $(t, c, 1)$-strongly stabilizing protocol $\mathcal{P}$ for BFS spanning tree construction under the central daemon. Let $S = (V, E)$ be the following system $V = \{p_0 = r, p_1, \dots, p_{2c+2}, p_{2c+3} = b\}$ and $E = \{\{p_i, p_{i+1}\}, i \in \{0, \dots, 2c+2\}\}$. Process p_0 is the real root and process b is a Byzantine one.

Assume that the initial configuration ρ_0 of S satisfies: $level_r = level_b = 0$, $prnt_r = prnt_b = \perp$ and other variables of b (if any) are identical to those of r (see Figure 2). Assume now that b takes exactly the same actions as r (if any) immediately after r (note that $d(r, b) > c$

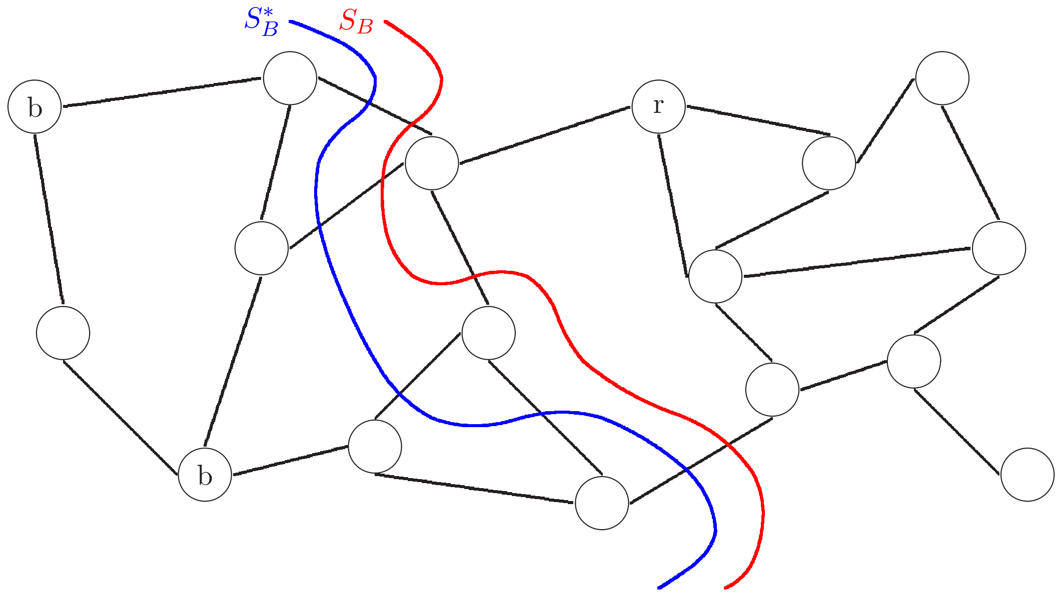

Figure 1: Example of containment areas for BFS spanning tree construction.

and hence $level_r = 0$ and $prnt_r = \perp$ still hold by closure and then $level_b = 0$ and $prnt_b = \perp$ still hold too). Then, by symmetry of the execution and by convergence of $\mathcal{P}$ to *spec*, we can deduce that the system reaches in a finite time a configuration ρ_1 (see Figure 2) in which: $\forall i \in \{1, \ldots, c+1\}, level_{p_i} = i$ and $prnt_{p_i} = p_{i-1}$ and $\forall i \in \{c+2, \ldots, 2c+2\}, level_{p_i} = 2c+3-i$ and $prnt_{p_i} = p_{i+1}$ (because this configuration is the only one in which all correct process v such that $d(v,b) > c$ satisfies $spec(v)$ when $level_r = level_b = 0$ and $prnt_r = prnt_b = \perp$). Note that ρ_1 is 0-legitimate and 0-stable and *a fortiori* c-legitimate and c-stable.

Assume now that the Byzantine process acts as a correct process and executes correctly $\mathcal{P}$. Then, by convergence of $\mathcal{P}$ in fault-free systems (remember that a $(t,c,1)$-strongly stabilizing protocol is a special case of self-stabilizing protocol), we can deduce that the system reaches in a finite time a configuration ρ_2 (see Figure 2) in which: $\forall i \in \{1, \ldots, 2c+3\}, level_{p_i} = i$ and $prnt_{p_i} = p_{i-1}$ (because this configuration is the only one in which every process v satisfies $spec(v)$). Note that the portion of execution between ρ_1 and ρ_2 contains at least one c-perturbation (p_{c+2} is a c-correct process and modifies at least once its O-variables) and that ρ_2 is 0-legitimate and 0-stable and *a fortiori* c-legitimate and c-stable.

Assume now that the Byzantine process b takes the following state: $level_b = 0$ and $prnt_b = \perp$. This step brings the system into configuration ρ_3 (see Figure 2). From this configuration, we can repeat the execution we constructed from ρ_0. By the same token, we obtain an execution of $\mathcal{P}$ which contains c-legitimate and c-stable configurations (see ρ_1) and an infinite number of c-perturbation which contradicts the $(t,c,1)$-strong stabilization of $\mathcal{P}$. □

5.2 Byzantine containment properties of the $min + 1$ protocol

In the $min + 1$ protocol, as in many self-stabilizing tree construction protocols, each process v checks locally the consistence of its $level_v$ variable with respect to the one of its neighbors. When it detects an inconsistency, it changes its $prnt_v$ variable in order to choose a "better" neighbor. The notion of "better" neighbor is based on the global desired property on the tree (here, the BFS requirement implies to choose one neighbor with the minimum level).

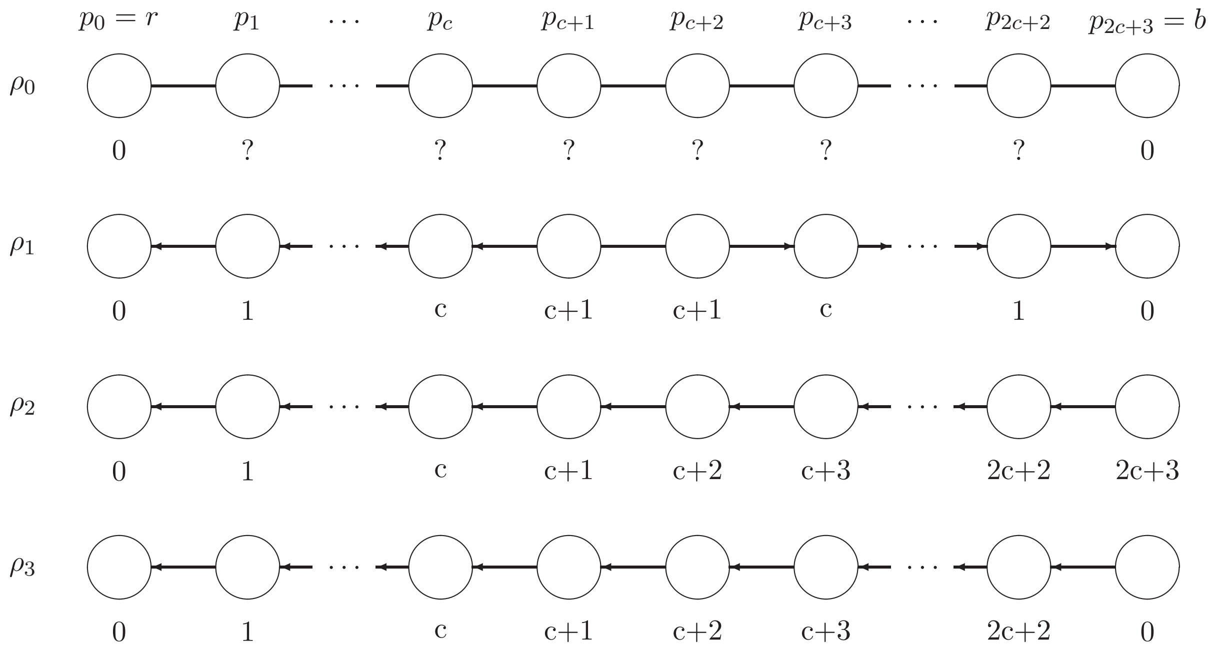

Figure 2: Configurations used in proof of Theorem 2.

When the system may contain Byzantine processes, they may disturb their neighbors by providing alternatively "better" and "worse" states.

The $min + 1$ protocol chooses an arbitrary one of the "better" neighbors (that is, neighbors with the minimal level). Actually this strategy allows us to achieve the (S_B, f)-TA strict stabilization but is not sufficient to achieve the (t, S_B^*, f)-TA strong stabilization. To achieve the (t, S_B^*, f)-TA strong stabilization, we must bring a slight modification to the protocol: we choose a "better" neighbor with a round robin order (among its neighbors with the minimal level).

Algorithm 5.1 presents our BFS spanning tree construction protocol $\mathcal{SSBFS}$ which is both (S_B, f)-TA strictly and (t, S_B^*, f)-TA strongly stabilizing (where $f \leq n-1$ and $t = 2m$) provided that the root is never Byzantine.

In the following of this section, we provide proofs of topology-aware strict and strong stabilization of $\mathcal{SSBFS}$. First at all, remember that the real root r can not be a Byzantine process by hypothesis. Note that the subsystems whose set of nodes are respectively $V \setminus S_B$ and $V \setminus S_B^*$ are connected by construction.

$(S_B, n-1)$-TA strict stabilization

Given a configuration $\rho \in C$ and an integer $d \in \{0, \dots, D\}$, let us define the following predicate:

$$I_d(\rho) \equiv \forall v \in V, level_v \geq min\left\{d, \min_{u \in B \cup \{r\}}\{d(v,u)\}\right\}$$

Lemma 1 *For any integer $d \in \{0, \dots, D\}$, the predicate I_d is closed.*

algorithm 5.1 $\mathcal{SSBFS}$: A TA strictly and TA strongly stabilizing protocol for BFS tree construction

Data:

N_v: totally ordered set of neighbors of v

Variables:

$prnt_v \in N_v \cup \{\bot\}$: pointer on the parent of v in the tree.

$level_v \in \mathbb{N}$: integer

Macro:

For any subset $A \subseteq N_v$, $choose(A)$ returns the first element of A which is bigger than $prnt_v$ (in a round-robin fashion).

Rules:

$\boldsymbol{(R_r)} :: (v = r) \wedge ((prnt_v \neq \bot) \vee (level_v \neq 0)) \longrightarrow prnt_v := \bot; level_v := 0$

$\boldsymbol{(R_v)} :: (v \neq r) \wedge \left((prnt_v = \bot) \vee (level_v \neq level_{prnt_v} + 1) \vee (level_{prnt_v} \neq \min\limits_{q \in N_v}\{level_q\}) \right) \longrightarrow$

$prnt_v := choose\left(\left\{ p \in N_v \,\middle|\, level_p = \min\limits_{q \in N_v}\{level_q\} \right\}\right); level_v := level_{prnt_v} + 1$

Proof Let d be an integer such that $d \in \{0, \dots, D\}$. Let $\rho \in C$ be a configuration such that $I_d(\rho) = true$ and $\rho' \in C$ be a configuration such that $\rho \overset{R}{\mapsto} \rho'$ is a step of $\mathcal{SSBFS}$.

If the root process $r \in R$ (respectively a Byzantine process $b \in R$), then we have $level_r = 0$ (respectively $level_b \geq 0$) in ρ' by construction of $\boldsymbol{(R_r)}$ (respectively by definition of $level_b$). Hence, $level_r \geq min\left\{d, \min\limits_{u \in B \cup \{r\}}\{d(r,u)\}\right\} = 0$ (respectively $level_b \geq min\left\{d, \min\limits_{u \in B \cup \{r\}}\{d(b,u)\}\right\} = 0$).

If a correct process $v \in R$ satisfies $v \neq r$, then there exists a neighbor p of v which satisfies the following property in ρ (since v is activated and $I_d(\rho) = true$):

$$level_p = \min_{q \in N_v}\{level_q\} \geq min\left\{d, \min_{u \in B \cup \{r\}}\{d(p,u)\}\right\}$$

Once, v is activated, we have: $level_v = level_p + 1$ in ρ'. Let be $\delta = \min\limits_{u \in B \cup \{r\}}\{d(v,u)\}$. Then, we have: $\min\limits_{u \in B \cup \{r\}}\{d(p,u)\} \geq \delta - 1$ (otherwise, we have a contradiction with the fact that $\delta = \min\limits_{u \in B \cup \{r\}}\{d(v,u)\}$ and that v and p are neighbors). Consequently, ρ' satisfies:

$$\begin{aligned} level_v = level_p + 1 &\geq min\left\{d, \min_{u \in B \cup \{r\}}\{d(p,u)\}\right\} + 1 \\ &\geq min\{d, \delta - 1\} + 1 \\ &\geq min\{d, \delta\} \\ &\geq min\left\{d, \min_{u \in B \cup \{r\}}\{d(v,u)\}\right\} \end{aligned}$$

We can deduce that $I_d(\rho') = true$, that concludes the proof. □

Let $\mathcal{LC}$ be the following set of configurations:

$$\mathcal{LC} = \{\rho \in C \,|\, (\rho \text{ is } S_B\text{-legitimate for } spec) \wedge (I_D(\rho) = true)\}$$

Lemma 2 *Any configuration of $\mathcal{LC}$ is $(S_B, n-1)$-TA contained for spec.*

Proof Let ρ be a configuration of $\mathcal{LC}$. By construction, ρ is S_B-legitimate for *spec*.

In particular, the root process satisfies: $prnt_r = \perp$ and $level_r = 0$. By construction of $\mathcal{SSBFS}$, r is not enabled and then never modifies its O-variables (since the guard of the rule of r does not involve the state of its neighbors).

In the same way, any process $v \in V \setminus (S_B \cup \{r\})$ satisfies: $prnt_v \in N_v$, $level_v = level_{prnt_v} + 1$, and $level_{prnt_v} = \min_{u \in N_v}\{level_u\}$. Note that, as $v \in V \setminus (S_B \cup \{r\})$ and $spec(v)$ holds in ρ, we have: $level_v = d(v,r)$. Hence, process v is not enabled in ρ. It remains so until none of its neighbors u modifies its $level_u$ variable to a value α such that $\alpha \leq level_v - 2$.

Assume that there exists an execution e starting from ρ in which a neighbor u of a process $v \in V \setminus (S_B \cup \{r\})$ modifies $level_u$ to satisfy $level_u \leq level_v - 2$ (without loss of generality, assume that u is the first process to modify $level_u$ in such a way in e). Note that $\min_{p \in B \cup \{r\}}\{d(u,p)\} \geq d(v,r) - 1$ (otherwise, we have a contradiction with the fact that $d(v,r) = \min_{p \in B \cup \{r\}}\{d(v,p)\}$ and that v and u are neighbors). Hence, we have:

$$\begin{aligned} \min_{p \in B \cup \{r\}}\{d(u,p)\} &\geq d(v,r) - 1 \\ &> d(v,r) - 2 \\ &> level_u \end{aligned}$$

This contradicts the closure of predicate I_D established in Lemma 1.

Consequently, there exists no such execution and process v remains infinitely disabled and then never modifies its O-variables. This concludes the proof. □

Lemma 3 *Starting from any configuration, any execution of $\mathcal{SSBFS}$ reaches a configuration of $\mathcal{LC}$ in a finite time.*

Proof We are going to prove the following property by induction on $d \in \{0, \dots, D\}$:

$(\mathcal{P}_d)$: Starting from any configuration, any run of $\mathcal{SSBFS}$ reaches a configuration ρ such that $I_d(\rho) = true$ and in which any process $v \notin S_B$ such that $d(v,r) \leq d$ satisfies $spec(v)$.

Initialization: $d = 0$.

Let ρ be an arbitrary configuration. Then, it is obvious that $I_0(\rho)$ is satisfied.

If a process $v \notin S_B$ satisfies $d(v,r) \leq 0$, then $v = r$. If v does not satisfy $spec(v)$ in ρ, then v is continuously enabled. Since the scheduling is fair, v is activated in a finite time and then v satisfies $spec(v)$ in a finite time. Then, we proved that $(\mathcal{P}_0)$ holds.

Induction: $d \geq 1$ and $\mathcal{P}_{d-1}$ is true.

We know, by $\mathcal{P}_{d-1}$, that any run of $\mathcal{SSBFS}$ under a distributed fair scheduler reaches a configuration ρ such that $I_{d-1}(\rho) = true$ and in which any process $v \notin S_B$ such that $d(v,r) \leq d-1$ satisfies $spec(v)$.

Let $E_d = \left\{ v \in V \,\middle|\, \min_{u \in B \cup \{r\}}\{d(v,u)\} \geq d \right\}$. Note that $I_{d-1}(\rho)$ implies that $\forall v \in E_d, level_v \geq d-1$ (since $\forall v \in E_d, min\left\{d-1, \min_{u \in B \cup \{r\}}\{d(v,u)\}\right\} = d-1$ by construction).

Note that any process $v \in E_d$ such that $level_v = d-1$ is enabled by $\boldsymbol{(R_v)}$ since we have: $level_{prnt_v} \geq d-1$ (by $I_{d-1}(\rho)$ and the fact that $prnt_v$ is a neighbor of v) and thus $level_v = d-1 < level_{prnt_v} + 1$. Moreover, this rule remains enabled until v is activated by closure of $I_{d-1}(\rho)$ (see Lemma 1). As the scheduling is fair, we deduce that any process $v \in E_d$ such that $level_v = d-1$ is activated in any run starting from ρ and $level_v \geq d$ holds. Then, we can conclude that any run starting from ρ reaches in a finite time a configuration ρ' such that $I_d(\rho') = true$.

Let $v \notin S_B$ be a process such that $d(r,v) = d$. We distinguish the following two cases:

Case 1: $spec(v)$ holds in ρ' (and then $level_v = d$).
By closure of I_d, any configuration of any run starting from ρ' satisfies I_d. Moreover, v satisfies $d(v,r) < \underset{u \in B}{min}\{d(v,u)\}$. Hence, there exists a BFS path from v to r. By construction, process v is then not enabled (remind that any neighbor u of v satisfies: $level_u \geq min\left\{d, \underset{w \in B \cup \{r\}}{min}\{d(u,w)\}\right\} \geq d$). In conclusion, v always satisfies $spec(v)$ in any run starting from ρ'.

Case 2: $spec(v)$ does not hold in ρ'.
By construction of ρ', we can split N_v into two sets S and $\bar{S}$ such that any process u of S satisfies $level_u = d(r,u) = d-1$ and $spec(u)$ (and thus there exists a BFS path from u to r) and any process u of $\bar{S}$ satisfies $level_u \geq d$ (remind that $I_d(\rho') = true$ and then $level_u \geq min\left\{d, \underset{p \in B \cup \{r\}}{min}\{d(u,p)\}\right\} \geq d$).
As $spec(v)$ does not hold in ρ', we can deduce that v is enabled in ρ'. As I_d is closed (by Lemma 1), we can deduce that v remains enabled. Since the scheduling is fair, we conclude that v is activated in a finite time in any run starting from ρ' and then $prnt_v$ is a process of S that implies that v satisfies $spec(v)$ in a finite time in any run starting from ρ'.

In conclusion, $\mathcal{P}_d$ is true, that ends the induction.

Then, it is easy to see that $\mathcal{P}_D$ (where D is the diameter of the system) implies the result.
□

Theorem 3 *$\mathcal{SSBFS}$ is a $(S_B, n-1)$-TA strictly stabilizing protocol for spec.*

Proof This result is a direct consequence of Lemmas 2 and 3. □

$(2m, S_B^*, n-1)$-TA strong stabilization Let be $E_B = S_B \setminus S_B^*$ (*i.e.* E_B is the set of process v such that $d(r,v) = \underset{b \in B}{min}\{d(v,b)\}$).

Lemma 4 *If ρ is a configuration of $\mathcal{LC}$, then any process $v \in E_B$ is activated at most Δ_v times in any execution starting from ρ.*

Proof Let ρ be a configuration of $\mathcal{LC}$ and v a process of E_B. By construction, there exists a neighbor u of v such that $u \in V \setminus S_B$. Then, we know that $spec(u)$ holds in ρ. By Lemma 2, we are ensured that $spec(u)$ remains true in any configuration of any execution starting from ρ. In particular, $level_u = d(r,u)$. By closure of $I_D(\rho)$, we know that $level_p \geq d(r,u)$ for any neighbor p of v. Consequently, $level_u = \underset{q \in N_v}{min}\{level_q\}$. This implies that, if $prnt_v = u$ and $level_v = level_u + 1$ in a configuration ρ', then $spec(v)$ is satisfied and v takes no actions in any execution starting from ρ'.

Then, the construction of the macro *choose* implies that u is chosen as v's parent in at most Δ_v actions of v. This implies the result. □

Lemma 5 *If ρ is a configuration of $\mathcal{LC}$ and v is a process such that $v \in E_B$, then for any execution e starting from ρ either*

1. *there exists a configuration ρ' of e such that $spec(v)$ is always satisfied after ρ', or*
2. *v is activated in e.*

Proof Let ρ be a configuration of $\mathcal{LC}$ and v be a process such that $v \in E_B$. By contradiction, assume that there exists an execution starting from ρ such that (i) $spec(v)$ is infinitely often false in e and (ii) v is never activated in e.

For any configuration ρ, let us denote by $P_v(\rho) = (v_0 = v, v_1 = prnt_v, v_2 = prnt_{v_1}, \ldots, v_k = prnt_{v_{k-1}}, p_v = prnt_{v_k})$ the maximal sequence of processes following pointers $prnt$ (maximal means here that either $prnt_{p_v} = \perp$ or p_v is the first process such that there $p_v = v_i$ for some $i \in \{0, \ldots, k\}$).

Let us study the following cases:

Case 1: $prnt_v \in V \setminus S_B$ in ρ.
Since $\rho \in \mathcal{LC}$, $prnt_v$ satisfies $spec(prnt_v)$ in ρ and in any execution starting from ρ (by Lemma 2). If v does not satisfy $spec(v)$ in ρ, then we have $level_v \neq level_{prnt_v} + 1$ in ρ. Then, v is continuously enabled in e and we have a contradiction between assumption (ii) and the fairness of the scheduling. This implies that v satisfies $spec(v)$ in ρ. The closure of I_D (established in Lemma 1) ensures us that v is never enabled in any execution starting from ρ. Hence, $spec(v)$ remains true in any execution starting from ρ. This contradicts the assumption (i) on e.

Case 2: $prnt_v \notin V \setminus S_B$ in ρ.
By the assumption (i) on e, we can deduce that there exists infinitely many configurations ρ' such that a process of $P_v(\rho')$ is enabled. By construction, the length of $P_v(\rho')$ is finite for any configuration ρ' and there exists only a finite number of processes in the system. Consequently, there exists at least one process which is infinitely often enabled in e. Since the scheduler is fair, we can conclude that there exists at least one process which is infinitely often activated in e.

Let A_e be the set of processes which are infinitely often activated in e. Note that $v \notin A_e$ by assumption (ii) on e. Let $e' = \rho' \ldots$ be the suffix of e which contains only activations of processes of A_e. Let p be the first process of $P_v(\rho')$ which belongs to A_e (p exists since at least one process of P_v is enabled when $spec(v)$ is false). By construction, the prefix of $P_v(\rho'')$ from v to p in any configuration ρ'' of e remains the same as the one of $P_v(\rho')$. Let p' be the process such that $prnt_{p'} = p$ in e' (p' exists since $v \neq p$ implies that the prefix of $P_v(\rho')$ from v to p counts at least two processes). As p is infinitely often activated and as any activation of p modifies the value of $level_p$ (it takes at least two different values in e'), we can deduce that p' is infinitely often enabled in e' (since the value of $level_{p'}$ is constant by construction of e' and p). Since the scheduler is fair, p' is activated in a finite time in e', that contradicts the construction of p.

In the two cases, we obtain a contradiction with the construction of e, that proves the result. $\square$

Let $\mathcal{LC}^*$ be the following set of configurations:

$$\mathcal{LC}^* = \{\rho \in C \,|(\rho \text{ is } S_B^*\text{-legitimate for } spec) \wedge (I_D(\rho) = true)\}$$

Note that, as $S_B^* \subseteq S_B$, we can deduce that $\mathcal{LC}^* \subseteq \mathcal{LC}$. Hence, properties of Lemmas 4 and 5 also apply to configurations of $\mathcal{LC}^*$.

Lemma 6 *Any configuration of $\mathcal{LC}^*$ is $(2m, \Delta, S_B^*, n-1)$-TA time contained for spec.*

Proof Let ρ be a configuration of $\mathcal{LC}^*$. As $S_B^* \subseteq S_B$, we know by Lemma 2 that any process v of $V \setminus S_B$ satisfies $spec(v)$ and takes no action in any execution starting from ρ.

Let v be a process of E_B. By Lemmas 4 and 5, we know that v takes at most Δ_v actions in any execution starting from ρ. Moreover, we know that v satisfies $spec(v)$ after its last action (otherwise, we obtain a contradiction between the two lemmas). Hence, any process of E_B takes at most $\Delta_v \leq \Delta$ actions and then, there are at most $\sum_{v \in V} \Delta_v = 2m$ S_B^*-TA-disruptions in any execution starting from ρ.

By definition of a TA time contained configuration, we obtain the result. $\square$

Lemma 7 *Starting from any configuration, any execution of $\mathcal{SSBFS}$ reaches a configuration of $\mathcal{LC}^*$ in a finite time under a distributed fair scheduler.*

Proof Let ρ be an arbitrary configuration. We know by Lemma 3 that any execution starting from ρ reaches in a finite time a configuration ρ' of $\mathcal{LC}$.

Let v be a process of E_B. By Lemmas 4 and 5, we know that v takes at most Δ_v actions in any execution starting from ρ'. Moreover, we know that v satisfies $spec(v)$ after its last action (otherwise, we obtain a contradiction between the two lemmas). This implies that any execution starting from ρ' reaches a configuration ρ'' such that any process v of E_B satisfies $spec(v)$. It is easy to see that $\rho'' \in \mathcal{LC}^*$, that ends the proof. $\square$

Theorem 4 *$\mathcal{SSBFS}$ is a $(2m, S_B^*, n-1)$-TA strongly stabilizing protocol for spec.*

Proof This result is a direct consequence of Lemmas 6 and 7. $\square$

5.3 Optimality of containment areas of the $min + 1$ protocol

Theorem 5 *Even under the central daemon, there exists no $(A_B, 1)$-TA strictly stabilizing protocol for BFS spanning tree construction where $A_B \subsetneq S_B$.*

Proof This is a direct application of the Theorem 2 of [9]. $\square$

Theorem 6 *Even under the central daemon, there exists no $(t, A_B, 1)$-TA strongly stabilizing protocol for BFS spanning tree construction where $A_B \subsetneq S_B$ and t is any (finite) integer.*

Proof Let $\mathcal{P}$ be a $(t, A_B, 1)$-TA strongly stabilizing protocol for BFS spanning tree construction protocol where $A_B \subsetneq S_B^*$ and t is a finite integer. We must distinguish the following cases:

Consider the following system: $V = \{r, u, u', v, v', b\}$ and $E = \{\{r, u\}, \{r, u'\}, \{u, v\}, \{u', v'\}, \{v, b\}, \{v', b\}\}$ (b is a Byzantine process). We can see that $S_B^* = \{v, v'\}$. Since $A_B \subsetneq S_B$, we have: $v \notin A_B$ or $v' \notin A_B$. Consider now the following configuration ρ_0: $prnt_r = prnt_b = \perp$, $level_r = level_b = 0$, $prnt$ and $level$ variables of other processes are arbitrary (see Figure 3, other variables may have arbitrary values but other variables of b are identical to those of r).

Assume now that b takes exactly the same actions as r (if any) immediately after r. Then, by symmetry of the execution and by convergence of $\mathcal{P}$ to $spec$, we can deduce that the system reaches in a finite time a configuration ρ_1 (see Figure 3) in which: $prnt_r = prnt_b = \perp$, $prnt_u = prnt_{u'} = r$, $prnt_v = prnt_{v'} = b$, $level_r = level_b = 0$ and $level_u = level_{u'} = level_v = level_{v'} = 1$ (because this configuration is the only one in which all correct process v satisfies $spec(v)$ when $prnt_r = prnt_b = \perp$ and $level_r = level_b = 0$). Note that ρ_1 is A_B-legitimate for $spec$ and A_B-stable (whatever A_B is).

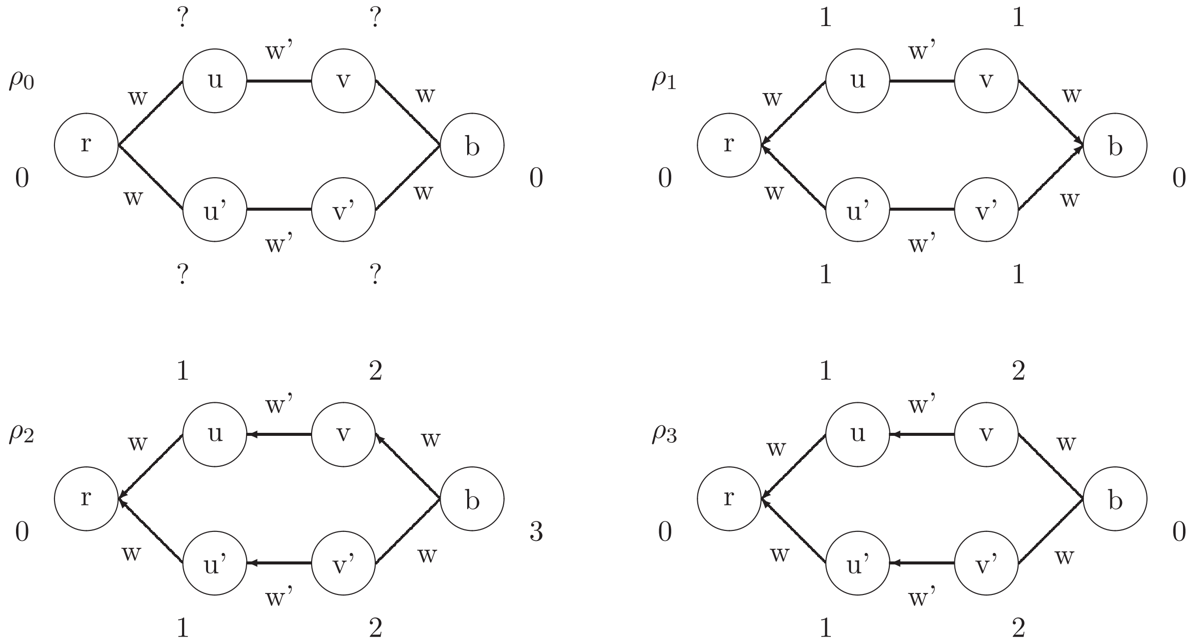

Figure 3: Configurations used in proof of Theorem 6.

Assume now that b behaves as a correct process with respect to $\mathcal{P}$. Then, by convergence of $\mathcal{P}$ in a fault-free system starting from ρ_1 which is not legitimate (remember that a strongly-stabilizing protocol is a special case of self-stabilizing protocol), we can deduce that the system reaches in a finite time a configuration ρ_2 (see Figure 3) in which: $prnt_r = \perp$, $prnt_u = prnt_{u'} = r$, $prnt_v = u$, $prnt_{v'} = u'$, $prnt_b = v$ (or $prnt_b = v'$), $level_r = 0$, $level_u = level_{u'} = 1$ $level_v = level_{v'} = 2$ and $level_b = 3$. Note that processes v and v' modify their O-variables in the portion of execution between ρ_1 and ρ_2 and that ρ_2 is A_B-legitimate for *spec* and A_B-stable (whatever A_B is). Consequently, this portion of execution contains at least one A_B-TA-disruption (whatever A_B is).

Assume now that the Byzantine process b takes the following state: $prnt_b = \perp$ and $level_b = 0$. This step brings the system into configuration ρ_3 (see Figure 3). From this configuration, we can repeat the execution we constructed from ρ_0. By the same token, we obtain an execution of $\mathcal{P}$ which contains c-legitimate and c-stable configurations (see ρ_1) and an infinite number of A_B-TA-disruption (whatever A_B is) which contradicts the $(t, A_B, 1)$-TA strong stabilization of $\mathcal{P}$. □

6 Conclusion

In this article, we are interested in the BFS spanning tree construction in presence of both systemic transient faults and permanent Byzantine failures. As this task is global, it is impossible to solve it in a strictly stabilizing way. We proved then that there exists no solution to this problem even if we consider the weaker notion of strong stabilization.

Then, we provide a study of Byzantine containment properties of the well-known $min + 1$ protocol. This protocol is one of the simplest self-stabilizing protocols which solve this problem. However, we prove that it achieves optimal area containment with respect to the notion of topology-aware strict and strong stabilization. All our results are summarized in the above table.

	BFS spanning tree construction
(c, f)-strict stabilization (for any c and f)	Impossible (Theorem 1)
(t, c, f)-strong stabilization (for any t, c, f)	Impossible (Theorem 2)
(A_B, f)-TA strict stabilization (for any f and $A_B \subsetneq S_B$)	Impossible (Theorem 5)
(S_B, f)-TA strict stabilization (for $0 \leq f \leq n-1$)	Possible (Theorem 3)
(t, A_B, f)-TA strong stabilization (for any f, t and $A_B \subsetneq S_B^*$)	Impossible (Theorem 6)
(t, S_B^*, f)-TA strong stabilization (for $0 \leq f \leq n-1$)	Possible (Theorem 4, $t = 2m$)

Using the result of [10] about r-operators, we can easily extend results of this paper to some others problems as depth-search or reliability spanning trees. This work raises the following open questions. Has any other global static task as leader election or maximal matching a topology-aware strictly or/and strongly stabilizing solution ? We can also wonder about non static tasks as mutual exclusion (recall that local mutual exclusion has a strictly stabilizing solution provided by [17]).

References

[1] Michael Ben-Or, Danny Dolev, and Ezra N. Hoch. Fast self-stabilizing byzantine tolerant digital clock synchronization. In Rida A. Bazzi and Boaz Patt-Shamir, editors, *PODC*, pages 385–394. ACM, 2008.

[2] Ariel Daliot and Danny Dolev. Self-stabilization of byzantine protocols. In Ted Herman and Sébastien Tixeuil, editors, *Self-Stabilizing Systems*, volume 3764 of *Lecture Notes in Computer Science*, pages 48–67. Springer, 2005.

[3] Ajoy Kumar Datta and Maria Gradinariu, editors. *Stabilization, Safety, and Security of Distributed Systems, 8th International Symposium, SSS 2006, Dallas, TX, USA, November 17-19, 2006, Proceedings*, volume 4280 of *Lecture Notes in Computer Science*. Springer, 2006.

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

[5] Danny Dolev and Ezra N. Hoch. On self-stabilizing synchronous actions despite byzantine attacks. In Andrzej Pelc, editor, *DISC*, volume 4731 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 2007.

[6] S. Dolev. *Self-stabilization.* MIT Press, March 2000.

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

[8] Swan Dubois, Toshimitsu Masuzawa, and Sébastien Tixeuil. Self-stabilization with byzantine tolerance for global tasks. Research report inria-00484645 (http://hal.inria.fr/inria-00484645/en/), INRIA, 05 2010.

[9] Swan Dubois, Toshimitsu Masuzawa, and Sébastien Tixeuil. The Impact of Topology on Byzantine Containment in Stabilization. Research report inria-00481836 (http://hal.inria.fr/inria-00481836/en/), INRIA, 05 2010.

[10] Bertrand Ducourthial and Sébastien Tixeuil. Self-stabilization with r-operators. *Distributed Computing*, 14(3):147–162, July 2001.

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

[12] Ezra N. Hoch, Danny Dolev, and Ariel Daliot. Self-stabilizing byzantine digital clock synchronization. In Datta and Gradinariu [3], pages 350–362.

[13] Shing-Tsaan Huang and Nian-Shing Chen. A self-stabilizing algorithm for constructing breadth-first trees. *Inf. Process. Lett.*, 41(2):109–117, 1992.

[14] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[15] Toshimitsu Masuzawa and Sébastien Tixeuil. Bounding the impact of unbounded attacks in stabilization. In Datta and Gradinariu [3], pages 440–453.

[16] 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, December 2007.

[17] Mikhail Nesterenko and Anish Arora. Tolerance to unbounded byzantine faults. In *21st Symposium on Reliable Distributed Systems (SRDS 2002)*, page 22. IEEE Computer Society, 2002.

[18] Yusuke Sakurai, Fukuhito Ooshita, and Toshimitsu Masuzawa. A self-stabilizing link-coloring protocol resilient to byzantine faults in tree networks. In *Principles of Distributed Systems, 8th International Conference, OPODIS 2004*, volume 3544 of *Lecture Notes in Computer Science*, pages 283–298. Springer, 2005.

[19] 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, November 2009.