Bounding the Impact of Unbounded Attacks in Stabilization: Supplementary File

Swan Dubois, Toshimitsu Masuzawa, and Sébastien Tixeuil

1 Discussion between Strong Stabilization and Pseudo Stabilization

There exists an analogy between the respective powers of (c, f)-strict stabilization and (t, c, f)-strong stabilization for the one hand, and self-stabilization and pseudo-stabilization for the other hand.

A *pseudo-stabilizing* protocol (defined in [1]) guarantees that every execution has a suffix that matches the specification, but it could never reach a legitimate configuration from which any possible execution matches the specification. In other words, a pseudo-stabilizing protocol can continue to behave satisfying the specification, but with having possibility of invalidating the specification in future. A particular schedule can prevent a pseudo-stabilizing protocol from reaching a legitimate configuration for arbitrarily long time, but cannot prevent it from executing its desired behavior (that is, a behavior satisfying the specification) for arbitrarily long time. Thus, a pseudo-stabilizing protocol is useful since desired behavior is eventually reached.

- 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 [2]. 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.

Similarly, every execution of a (t, c, f)-strongly stabilizing protocol has a suffix such that every c-correct process executes its desired behavior. But, for a (t, c, f)-strongly stabilizing protocol, there may exist executions such that the system never reach a configuration after which Byzantine processes never have the ability to disturb the c-correct processes: all the c-correct processes can continue to execute their desired behavior, but with having possibility that the system (resp. each process) could be disturbed at most t (resp. k) times by Byzantine processes in future. A notable but subtle difference is that the invalidation of the specification is caused only by the effect of Byzantine processes in a (t, c, f)-strongly stabilizing protocol, while the invalidation can be caused by a scheduler in a pseudo-stabilizing protocol.

2 Strongly-Stabilizing Spanning Tree Construction

2.1 Complete Proofs of Strong Stabilization of ss-ST

Lemma 1

Proof: Let ρ be a configuration of $\mathcal{LC}$. By definition of $spec$, it is obvious that ρ is 0-legitimate.

Note that no correct process is enabled by ss-ST in ρ. Consequently, no actions of ss-ST can be executed and we can deduce that ρ is 0-stable. □

Lemma 2

Proof: First, note that if all the correct processes are disabled in a configuration ρ, then ρ belongs to $\mathcal{LC}$. Thus, it is sufficient to show that ss-ST eventually reaches a configuration ρ_i in any execution (starting from any configuration) such that all the correct processes are disabled in ρ_i.

By contradiction, assume that there exists a correct process that is enabled infinitely often. Notice that once the root process r is activated, r becomes and remains disabled forever. From the assumption that all the correct processes form a connected subsystem, there exists two neighboring correct processes u and v such that u becomes and remains disabled and v is enabled infinitely often. Consider execution after u becomes and remains disabled. Since the daemon is weakly fair, v executes its action infinitely often. Then, eventually v designates u as its parent. It follows that v never becomes enabled again unless u changes $level_u$. Since u never becomes enabled, this leads to the contradiction. □

Lemma 3

Proof: Let ρ_0 be a configuration of $\mathcal{LC}$ and $e = \rho_0, \rho_1, \dots$ be an execution starting from ρ_0. First, we show that any 0-correct process takes at most Δ^d actions in e, where d is the diameter of the subsystem consisting of all the correct processes.

Let F be the set of Byzantine processes in e. Consider a subsystem S' consisting of all the correct processes: $S' = (P - F, L')$ where $L' = \{l \in L \mid l \in (P-F) \times (P-F)\}$. We prove by induction on the distance δ from the root in S' that a correct process v δ hops away from r in S' executes its action at most Δ^δ times in e.

- Induction basis ($\delta = 1$):
  Let v be any correct process neighboring to the root r. Since ρ_0 is a legitimate configuration, $prnt_r = 0$ and $level_r = 0$ hold at ρ_0 and remain unchanged in e. Thus, if $prnt_v = r$ and $level_v = 1$ hold in a configuration σ, then v never changes $prnt_v$ or $level_v$ in any execution starting from σ. Since $prnt_v = r$ and $level_v = 1$ hold within the first $\Delta_v - 1 \leq \Delta$ actions of v, v can execute its action at most Δ times.
- Induction step (with induction assumption):
  Let v be any correct process δ hops away from the root r in S', and u be a correct neighbor of v that is $\delta-1$ hops away from r in S' (this process exists by the assumption that the subgraph of correct processes of S is connected). From the induction assumption, u can execute its action at most $\Delta^{\delta-1}$ times.
  Assume that $prnt_v = u$ and $level_v = level_u + 1$ hold in a given configuration σ. We can observ that v is not enabled until u does not modify its state. Then, the round-robin order used for pointers modification allows us to deduce that v executes at most $\Delta_v \leq \Delta$ actions between two actions of u (or before the first action of u). By the induction assumption, u executes its action at most $\Delta^{\delta-1}$ times. Thus, v can execute its action at most $\Delta + \Delta \times (\Delta^{\delta-1}) = \Delta^\delta$ times.

Consequently, any 0-correct process takes at most Δ^d actions in e.

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 e, 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 $f\Delta^D$. It remains to show that any 0-disruption have a finite length to prove the result.

By contradiction, assume that there exists an infinite 0-disruption $d = \rho_i, \dots$ in e. This implies that for all $j \geq i$, ρ_j is not in $\mathcal{LC}$, which contradicts Lemma 2. Then, the result is proved. □

2.2 Time Complexities

Proposition 1: The $(f\Delta^d, 0, f)$-process-disruption time of *ss-ST* is Δ^d where d is the diameter of the subsystem consisting of all the correct processes.

Proof: This result directly follows from Theorem 1 and Lemma 3. □

Proposition 2: The $(f\Delta^d, 0, f)$-stabilization time of $ss\text{-}ST$ is $O((n-f)\Delta^d)$ rounds where f is the number of Byzantine processes and d is the diameter of the subsystem consisting of all the correct processes.

Proof: By the construction of the algorithm, any correct process v which has a correct neighbor u takes at most Δ steps between two actions of u.

Given two processes u and v, we denote by $d'(u,v)$ the distance between u and v in the subgraph of correct processes of S. We are going to prove the following property by induction on $i > 0$:

(P_i): any correct process v such that $d'(v,r) = i$ takes at most $2 \cdot \sum_{j=1}^{i} \Delta^j$ steps in any execution starting from any configuration.

- Induction basis ($i = 1$):
  Let v be a correct neighbor of the root r. By the algorithm, we know that the root r takes at most one step (because r is correct). By the previous remark, we know that v takes at most Δ steps before and after the action of r. Consequently, v takes at most 2Δ steps in any execution starting from any configuration.
- Induction step ($i > 1$ with induction assumption):
  Let v be a correct process such that $d'(v,r) = i$. Denote by u one neighbor of v such that $d'(u,r) = i-1$ (this process exists by the assumption that the subgraph of correct processes of S is connected).
  By the previous remark, we know that v takes at most Δ steps before the first action of u, between two actions of u and after the last action of u. By induction assumption, we know that u takes at most $2 \cdot \sum_{j=1}^{i-1} \Delta^j$ steps. Consequently, v takes at most A actions where:
  $$A = \Delta + \left(2 \cdot \sum_{j=1}^{i-1} \Delta^j\right) \cdot \Delta + \Delta = 2 \cdot \sum_{j=1}^{i} \Delta^j$$

Since there is $(n-f)$ correct processes and any correct process satisfies $d'(v,r) < d$, we can deduce that the system reach a legitimate configuration in at most $O((n-f)\Delta^d)$ steps of correct processes.

As a round counts at least one step of a correct process, we obtain the result. □

3 Strongly-Stabilizing Tree Orientation

3.1 Problem Definition

In this section, we consider only *tree systems*, *i.e.* distributed systems containing no cycles. We assume that all processes in a tree system are identical and thus no process is distinguished as a root.

Informally, *tree orientation* consists in transforming a tree system (with no root) into a rooted tree system. 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. But for simplicity, we use $prnt_v = k$ and $prnt_v = u$ (where u is the k-th neighbor of v) interchangeably.

The goal of tree orientation is to set $prnt_v$ of every process v to form a rooted tree. However, it is impossible to choose a single process as the root because of impossibility of symmetry breaking. Thus, instead of a single root process, a single *root link* is determined as the root: link (u,v) is the root link when processes u and v designate each other as their parents (Fig. 1(a)). From any process w, the root link can be reached by following the neighbors designated by the variables $prnt$.

When a tree system S has a Byzantine process (say w), w can prevent communication between subtrees of $S - \{w\}$[1]. Thus, we have to allow each of the subtrees to form a rooted tree independently. We define the specification predicate $spec(v)$ of the tree orientation as follows.

$$spec(v) : \forall u\ (\in N_v)[(prnt_v = u) \vee (prnt_u = v) \vee (u \text{ is Byzantine faulty})].$$

Note that the tree topology, the specification and the uniquiness of $prnt_v$ (for any process v) imply that, for any 0-legitimate configuration, there is at most one root link in any connected component of correct processes. Hence, in a fault-free system, there exists exactly one root link in any 0-legitimate configuration.

Figure 1 shows examples of 0-legitimate configurations (a) with no Byzantine process and (b) with

1. For a process subset P' $(\subseteq P)$, $S - P'$ denotes a distributed system obtained by removing processes in P' and their incident links.

a single Byzantine process w. The arrow attached to each process points the neighbor designated as its parent. Notice that, from Fig. 1(b), subtrees consisting of correct processes are classified into two categories: one is the case of forming a rooted tree with a root link in the subtree (T_1 in Fig. 1(b)), and the other is the case of forming a rooted tree with a root process, where the root process is a neighbor of a Byzantine process and designates the Byzantine process as its parent (T_2 in Fig. 1(b)).

3.2 Impossibility for Two Byzantine Processes

Tree orientation seems to be a very simple task. Actually, for tree orientation in fault-free systems, we can design a self-stabilizing protocol that chooses a link incident to a center process[2] as the root link: in case that the system has a single center, the center can choose a link incident to it, and in case that the system has two neighboring centers, the link between the centers become the root link. However, tree orientation becomes impossible if we have Byzantine processes. By the impossibility results of [3], we can show that tree orientation has no $(o(n), 1)$-strictly stabilizing protocol; *i.e.* the Byzantine influence cannot be contained in the sense of "strict stabilization", even if only a single Byzantine process is allowed.

An interesting question is whether the Byzantine influence can be contained in a weaker sense of "strong stabilization". The following theorem gives a negative answer to the question: if we have two Byzantine processes, bounding the number of disruptions is impossible. We prove the impossibility for more restricted schedules, called the *central daemon*, which disallows two or more processes to make actions at the same time. Notice that impossibility results under the central daemon are stronger than those under the distributed daemon in the sense that impossibility results under the central daemon also hold for the distributed daemon.

Theorem 1: Even under the central daemon, there exists no deterministic $(t, o(n), 2)$-strongly stabilizing protocol for tree orientation where t is any (finite) integer and n is the number of processes.

2. A process v is a center when v has the minimum eccentricity where eccentricity is the largest distance to a leaf. It is known that a tree has a single center or two neighboring centers.

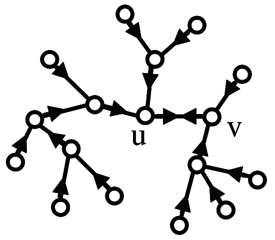
Proof: Let $S = (P, L)$ be a chain (which is a special case of a tree system) of n processes: $P = \{v_1,\ v_2, \ldots, v_n\}$ and $L = \{(v_i,\ v_{i+1}) \mid 1 \leq i \leq n-1\}$.

For purpose of contradiction, assume that there exists a $(t, o(n), 2)$-strongly stabilizing protocol A for some integer t. In the following, we show, for S with Byzantine processes v_1 and v_n, that A has an execution e containing an infinite number of $o(n)$-disruptions. This contradicts the assumption that A is a $(t, o(n), 2)$-strongly stabilizing protocol.

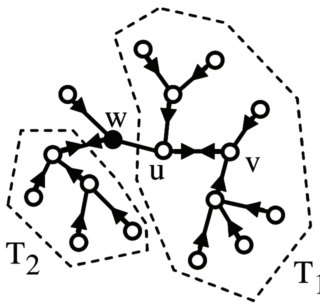
In S with Byzantine processes v_1 and v_n, A eventually reaches a configuration ρ_1 that is $o(n)$-legitimate for *spec* and $o(n)$-stable by definition of a $(t, o(n), 2)$-strongly stabilizing protocol. This execution to ρ_1 constitutes the prefix of e.

To construct e after ρ_1, consider another chain $S' = (P', L')$ of $3n$ processes and an execution of A on S', where let $P' = \{u_1,\ u_2, \ldots, u_{3n}\}$ and $L' = \{(u_i,\ u_{i+1}) \mid 1 \leq i \leq 3n-1\}$. We consider the initial configuration ρ'_1 of S' that is obtained by concatenating three copies (say S'_1, S'_2 and S'_3) of S in ρ_1 where only the central copy S'_2 is reversed right-and-left (Fig. 2). More formally, the state of w_i and of w_{2n+i} in ρ'_1 is the same as the one of v_i in ρ_1 for any $i \in \{1, \ldots, n\}$. Moreover, for any $i \in \{1, \ldots, n\}$, the state of w_{n+i} in ρ'_1 is the same as the one of v_i in ρ_1 with the following modification: if $prnt_{v_i} = v_{i-1}$ (respectively $prnt_{v_i} = v_{i+1}$) in ρ_1, then $prnt_{w_{n+i}} = w_{n+i+1}$ (respectively $prnt_{w_{n+i}} = w_{n+i-1}$) in ρ'_1. For example, if w denotes a center process of S (*i.e.* $w = v_{\lceil n/2 \rceil}$), then w is copied to $w'_1 = u_{\lceil n/2 \rceil}, w'_2 = u_{2n+1-\lceil n/2 \rceil}$ and $w'_3 = u_{2n+\lceil n/2 \rceil}$, but only $prnt_{w'_2}$ designates the neighbor in the different direction from $prnt_{w'_1}$ and $prnt_{w'_3}$. From the configuration ρ'_1, protocol A eventually reaches a legitimate configuration ρ''_1 of S' when S' has no Byzantine process (since a strongly stabilizimg protocol is self-stabilizig in a fault-free system). In the execution from ρ'_1 to ρ''_1, at least one $prnt$ variable of w'_1, w'_2 and w'_3 has to change (otherwise, it is impossible to guarantee the uniquiness of the root link in ρ''_1). Assume w'_i changes $prnt_{w'_i}$.

Now, we construct the execution e on S after ρ_1. The main idea of this proof is to construct an execution on S indistinguishable (for correct processes) from one of S' because Byzantine processes of S behave as correct processes of S'. Since v_1 and v_n are Byzantine processes in S, v_1 and v_n



(a) Case with no fault



(b) Case with Byzantine process w

Fig. 1. Tree orientation

can simulate behavior of the end processes of S'_i (*i.e.* $u_{(i-1)n+1}$ and u_{in}). Thus, S can behave in the same way as S'_i does from ρ'_1 to ρ''_1. Recall that process w'_1 modifies its pointer in the execution of S'_i does from ρ'_1 to ρ''_1. Consequently, we can construct the execution that constitutes the second part of e, where $prnt_w$ changes at least once. Letting the resulting configuration be ρ_2 (that coincides with the configuration ρ''_i of S'_i), ρ_2 is clearly $o(n)$-legitimate for *spec* and $o(n)$-stable. Thus, the second part of e contains at least one $o(n)$-disruption.

By repeating the argument, we can construct the execution e of A on S that contains an infinite number of $o(n)$-disruptions. □

3.3 A Strongly Stabilizing Protocol for a Single Byzantine Process

3.3.1 Protocol $ss\text{-}TO$

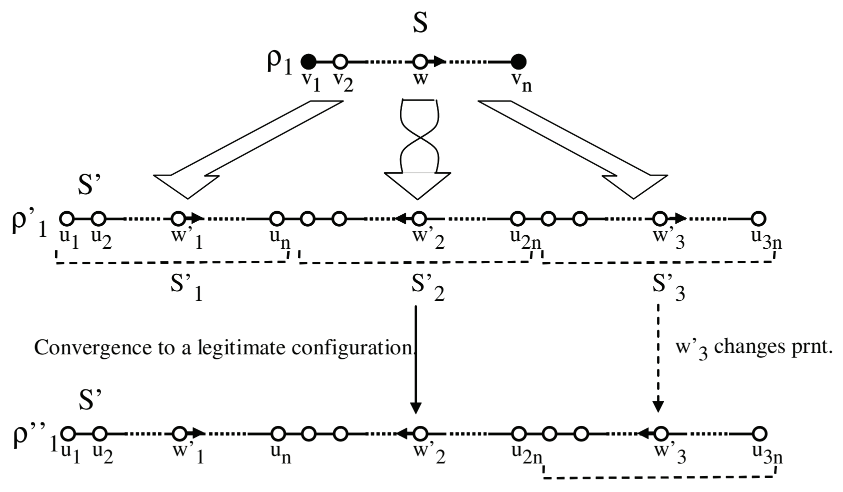
In the previous subsection, we proved that there is no strongly stabilizing protocol for tree orientation if two Byzantine processes exist. In this subsection, we consider the case with at most a single Byzantine process, and present a $(\Delta, 0, 1)$-strongly stabilizing tree orientation protocol $ss\text{-}TO$. Note that we consider the distributed daemon for this possibility result.

In a fault-free tree system, tree orientation can be easily achieved by finding a center process. A simple strategy for finding the center process is that each process v informs each neighbor u of the maximum distance to a leaf from u through v. The distances are found and become fixed from smaller ones. When a tree system contains a single Byzantine process, however, this strategy cannot prevent perturbation caused by wrong distances the Byzantine process provides: by reporting longer and shorter distances than the correct one alternatively, the Byzantine process can repeatedly pull the chosen center closer and push it farther.

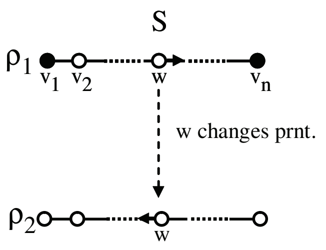
The key idea of protocol $ss\text{-}TO$ to circumvent the perturbation is to restrict the Byzantine influence to one-sided effect: the Byzantine process can pull the chosen root link closer but cannot push it farther. This can be achieved using a non-decreasing variable $level_v$ as follows: when a process v finds a neighbor u with a higher level, u chooses v as its parent and copies the level value from u. This allows the Byzantine process (say z) to make its neighbors choose z as their parents by increasing its own level. However, z can not make neighbor change their parents to other processes by decreasing its own level. Thus, the effect the Byzantine process can make is one-sided.

Protocol $ss\text{-}TO$ is presented in Fig. 3. For simplicity, we regard constant N_v as denoting the neighbors of v and regard variable $prnt_v$ as storing a parent of v. Notice that they should be actually implemented using the ordinal numbers of neighbors that v locally assigns.

The protocol is composed of three rules. The first one (`GA1`) is enabled when a process has a neighbor which provides a strictly greater level. When the rule is executed, the process chooses this neighbor as its parent and computes its new state in


S
ρ1
v1 v2 w vn
S'
ρ'1
u1 u2 w'1 un w'2 u2n w'3 u3n
S'1 S'2 S'3
Convergence to a legitimate configuration.
w'3 changes prnt.
S'
ρ''1
u1 u2 w'1 un w'2 u2n w'3 u3n

(a) Construction of S' from three copies of S and convergence of S'.



(b) Execution of S where w changes its parent.

Fig. 2. Construction of execution where w of S changes its parent infinitely often.

function of this neighbor. The rule GA2 is enabled when a process v has a neighbor u (different from its current parent) with the same level such that v is not the parent of u in the current oriented tree. Then, v chooses u as parent, increments its level by one and refresh its shared registers. The last rule (GA3) is enabled for a process when 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.

3.3.2 Closure of Legitimate Configurations of $ss\text{-}TO$

We refine legitimate configurations of protocol $ss\text{-}TO$ into several sets of configurations and show their properties. We cannot make any assumption on the initial values of register variables. But once a correct process v executes its action, variables of its output registers have values consistent with the process variables: $r\text{-}prnt_{v,prnt_v} = true$, $r\text{-}prnt_{v,w} = false$ ($w \in N_v - \{prnt_v\}$), and $r\text{-}level_{v,w} = level_v$ ($w \in N_v$) hold. In the following, we assume that all the variables of output registers of every correct process have consistent values.

First we consider the fault-free case.

Definition 1 ($\mathcal{LC}_0$): In a fault-free tree, we define the set of configurations $\mathcal{LC}_0$ as the set of configurations such that: (a) $spec(v)$ holds for every process v and (b) $level_u = level_v$ holds for any processes u and v.

```
constants of process $v$
    $\Delta_v$ = the degree of $v$;
    $N_v$ = the set of neighbors of $v$;
variables of process $v$
    $prnt_v$: a neighbor of $v$; // $prnt_v = u$ if $u$ is a parent of $v$.
    $level_v$: integer;
variables in shared register $r_{v,u}$
    $r\text{-}prnt_{v,u}$: boolean; // $r\text{-}prnt_{v,u} = true$ iff $u$ is a parent of $v$.
    $r\text{-}level_{v,u}$: integer; // the value of $level_v$
predicates
    $pred_1 : \exists u \in N_v [r\text{-}level_{u,v} > level_v]$
    $pred_2 : \exists u \in N_v - \{prnt_v\}[(r\text{-}level_{u,v} = level_v) \wedge (r\text{-}prnt_{u,v} = false)]$
    $pred_3 : ((r\text{-}prnt_{v,prnt_v}, r\text{-}level_{v,prnt_v}) \neq (true, level_v)) \vee$
            $(\exists u \in N_v - \{prnt_v\}, (r\text{-}prnt_{v,u}, r\text{-}level_{v,u}) \neq (false, level_v))$
atomic actions // represented in form of guarded actions
    GA1:$pred_1 \longrightarrow$
            Let $u$ be a neighbor of $v$ s.t. $r\text{-}level_{u,v} = \max_{w \in N_v} r\text{-}level_{w,v}$;
            $prnt_v := u$; $level_v := r\text{-}level_{u,v}$;
            $(r\text{-}prnt_{v,u}, r\text{-}level_{v,u}) := (true, level_v)$;
            for each $w \in N_v - \{u\}$ do $(r\text{-}prnt_{v,w}, r\text{-}level_{v,w}) := (false, level_v)$;
    GA2:$\neg pred_1 \wedge pred_2 \longrightarrow$
            Let $u$ be a neighbor of $v$ s.t. $(r\text{-}level_{u,v} = level_v) \wedge (r\text{-}prnt_{u,v} = false)$;
            $prnt_v := u$; $level_v := level_v + 1$;
            $(r\text{-}prnt_{v,u}, r\text{-}level_{v,u}) := (true, level_v)$;
            for each $w \in N_v - \{u\}$ do $(r\text{-}prnt_{v,w}, r\text{-}level_{v,w}) := (false, level_v)$;
    GA3:$\neg pred_1 \wedge \neg pred_2 \wedge pred_3 \longrightarrow$
            $(r\text{-}prnt_{v,prnt_v}, r\text{-}level_{v,prnt_v}) := (true, level_v)$;
            for each $w \in N_v - \{prnt_v\}$ do $(r\text{-}prnt_{v,w}, r\text{-}level_{v,w}) := (false, level_v)$;
```

Fig. 3. Protocol $ss\text{-}TO$ (actions of process v)

In any configuration of $\mathcal{LC}_0$, variables $prnt_v$ of all processes form a rooted tree with a root link as Fig. 1(a), and all variables $level_v$ have the same value.

Lemma 1: In a fault-free tree, once protocol $ss\text{-}TO$ reaches a configuration ρ in $\mathcal{LC}_0$, it remains at ρ.

Proof: Consider any configuration ρ in $\mathcal{LC}_0$. Since all variables $level_v$ have the same value, the guard of GA1 cannot be true in ρ. Since $spec(v)$ holds at every process in ρ, there exist no neighboring processes u and v such that $prnt_u \neq v$ and $prnt_v \neq u$ holds. It follows that the guard of GA2 cannot be true in ρ. Once each process executes an action, all the variables of its output registers are consistent with its local variables, and thus, the guard of GA3 cannot be true. □

For the case with a single Byzantine process, we define the following sets of configurations.

Definition 2 ($\mathcal{LC}_1$): Let z be the single Byzantine process in a tree system. A configuration is in the set $\mathcal{LC}_1$ if every subtree (or a connected component) of S-$\{z\}$ satisfies either the following (C1) or (C2).

(C1) (a) $spec(u)$ holds for every correct process u, (b) $prnt_v = z$ holds for the neighbor v of z, and (c) $level_w \geq level_x$ holds for any neighboring correct processes w and x where w is nearer than x to z.

(C2) (d) $spec(u)$ holds for every correct process u, and (e) $level_v = level_w$ holds for any correct processes v and w.

Definition 3 ($\mathcal{LC}_2$): Let z be the single Byzantine process in a tree system. A configuration is in the set $\mathcal{LC}_2$ if every subtree (or a connected component) of S-$\{z\}$ satisfies the condition (C1) of Definition 2.

In any configuration of $\mathcal{LC}_2$, every subtree forms the rooted tree with the root process neighboring the Byzantine process z. For configurations of $\mathcal{LC}_2$, the following lemma holds.

Lemma 2: Once protocol $ss\text{-}TO$ reaches a configuration ρ of $\mathcal{LC}_2$, it remains in configurations of $\mathcal{LC}_2$ and, thus, no correct process u changes $prnt_u$ afterward. That is, any configuration of $\mathcal{LC}_2$ is $(0,1)$-contained.

Proof: Consider any execution e starting from a configuration ρ of $\mathcal{LC}_2$. In ρ, every subtree of $S-\{z\}$ forms the rooted tree with the root process neighboring the Byzantine process z. Note that, as long as no correct process u changes $prnt_u$ in e, action `GA2` cannot be executed at any correct process. On the other hand, if a process u executes action `GA1` in e, $level_{prnt_u} \geq level_u$ necessarily holds immediately this action. Consequently, if we assume that no correct process u changes $prnt_u$ in e (by execution of `GA1`) then every configuration of e is in $\mathcal{LC}_2$. To prove the lemma, it remains to show that e contains no activation of `GA1` by a correct process. In the following, we show that any correct process u never changes $prnt_u$ in e.

For contradiction, assume that a correct process u changes $prnt_u$ first among all correct processes. Notice that every correct process v can execute `GA1` or `GA3` but cannot change $prnt_v$ before u changes $prnt_u$. Also notice that u changes $prnt_u$ to its neighbor (say w) by execution of `GA1` and w is a correct process. From the guard of `GA1`, $level_w > level_u$ holds immediately before u changes $prnt_u$. On the other hand, since w is a correct process, w never changes $prnt_w$ before u. This implies that $prnt_w = u$ holds immediately before u changes $prnt_u$, and thus $level_u \geq level_w$ holds. This is a contradiction. □

Notice that a correct process u may change $level_u$ by execution of `GA1` even after a configuration of $\mathcal{LC}_2$. For example, when the Byzantine process z increments $level_z$ infinitely often, every process u may also increment $level_u$ infinitely often.

Lemma 3: Any configuration ρ in $\mathcal{LC}_1$ is $(\Delta_z, 1, 0, 1)$-time contained where z is the Byzantine process.

Proof: Let ρ be a configuration of $\mathcal{LC}_1$. Consider any execution e starting from ρ. By the same discussion as the IEEEproof of Lemma 2, we can show that any subtree satisfying (C1) at ρ always keeps satisfying the condition and no correct process u in the subtree changes $prnt_u$ afterward.

Consider a subtree satisfying (C2) at ρ and let y be the neighbor of the Byzantine process z in the subtree. From the fact that variables $prnt_u$ form a rooted tree with a root link and all variables $level_u$ have the same value in the subtree at ρ, no process u in the subtree changes $prnt_u$ or $level_u$ unless y executes $prnt_y := z$ in e. When $prnt_y := z$ is executed, $level_y$ becomes larger than $level_u$ of any other process u in the subtree. Since the value of variable $level_u$ of each correct process u is non-decreasing, every correct neighbor (say v) of y eventually executes $prnt_v := y$ and $level_v := level_y$ (by `GA1`). By repeating the argument, we can show that the subtree eventually reaches a configuration satisfying (C1) in $O(d')$ rounds where d' is the diameter of the subtree. It is clear that any configuration before reaching the first configuration satisfying (C1) is not in $\mathcal{LC}_1$, and that each process u changes $prnt_u$ at most once during the execution.

Therefore, any execution e starting from ρ contains at most Δ_z 0-disruptions where each correct process u changes $prnt_u$ at most once. □

3.3.3 *Convergence of* $ss\text{-}TO$

We first show convergence of protocol $ss\text{-}TO$ to configurations of $\mathcal{LC}_0$ in a *fault-free* case.

Lemma 4: In a fault-free tree system, protocol $ss\text{-}TO$ eventually reaches a configuration of $\mathcal{LC}_0$ from any initial configuration.

Proof: We prove the convergence to a configuration of $\mathcal{LC}_0$ by induction on the number of processes n. It is clear that protocol $ss\text{-}TO$ reaches a configuration of $\mathcal{LC}_0$ from any initial configuration in case of $n = 2$.

Now assume that protocol $ss\text{-}TO$ reaches a configuration of $\mathcal{LC}_0$ from any initial configuration in case that the number of processes is $n-1$ (inductive hypothesis), and consider the case that the number of processes is n.

Let u be any leaf process and v be its only neighbor and ρ be an arbitrary configuration. In a first time, we show that any execution e starting from ρ reaches in a finite time a configuration such that $level_v \geq level_u$ holds. If this condition holds in ρ, we have the result. Otherwise ($level_v < level_u$), u is continuously enabled by `GA1` (until the condition is true). Hence, the condition becomes true (by an activation of v) or this action is executed by u in a finite time. In both cases, we obtain that $level_v \geq level_u$ holds in at most one round.

After that, process u can execute only guarded action `GA1` or `GA3` since $prnt_u = v$ always holds. Thus, after the first round completes, $prnt_u = v$ and $level_v \geq level_u$ always hold (indeed, v can only increase its $level$ variable and $level$ variable of u can only take greater values than v's). It follows that v never executes $prnt_v := u$ in the second round and later. This implies that e reaches in a finite time a configuration ρ' such that (a) $prnt_v \neq u$ always holds after ρ' , or (b) $prnt_v = u$ always holds after ρ' (since v cannot execute $prnt_v := u$ after ρ' if $prnt_v \neq u$).

In case (a), the behavior of v after ρ' is never influenced by u: v behaves exactly the same even when u does not exist. From the inductive hypothesis, protocol $ss\text{-}TO$ eventually reaches a configuration ρ'' such that $S - \{u\}$ satisfies the condition of $\mathcal{LC}_0$ and remains in ρ'' afterward (from Lemma 1). After u executes its action at ρ'', $level_u = level_v$ holds and thus the configuration of S is in $\mathcal{LC}_0$.

Now consider case (b), where we do not use the inductive hypothesis. The fact that $prnt_v = u$ (and $prnt_u = v$) always holds after ρ' implies that $level_v$ (and also $level_u$) remains unchanged after ρ'. Assume now that a neighbor w ($\neq u$) of v satisfies continuously $level_w \neq level_v$ or $prnt_w \neq v$ from a configuration ρ'' of e after ρ'. If w satisfies continuously $level_w > level_v$ from ρ'', then v executes `GA1` in a finite time, this is a contradiction. If w satisfies continuously $level_w < level_v$ from ρ'', then w executes `GA1` in a finite time and takes a $level$ value such that $level_w \geq level_v$, that contradicts the fact that w satisfies continuously $level_w < level_v$ from ρ''. This implies that $level_w = level_v$ and $prnt_w = v$ in a finite time in any execution starting from ρ'. As v does not modify its state after ρ', w is never enabled after ρ'. This implies that the fragment of S consisting of processes within distance two from u reaches a configuration satisfying the condition of $\mathcal{LC}_0$ and remains unchanged. We can now apply the same reasoning by induction on the distance of any process to u and show that $ss\text{-}TO$ eventually reaches a configuration in $\mathcal{LC}_0$ where link (u, v) is the root link.

Consequently, protocol $ss\text{-}TO$ reaches a configuration of $\mathcal{LC}_0$ from any initial configuration. □

Now, we consider the case with a single Byzantine process.

Lemma 5: In a tree system with a single Byzantine process, protocol $ss\text{-}TO$ eventually reaches a configuration of $\mathcal{LC}_1$ from any initial configuration.

Proof: Let z be the Byzantine process, S' be any subtree (or a connected component) of $S - \{z\}$ and y be the process in S' neighboring z (in S).

We prove, by induction on the number of processes n' of S', that S' eventually reaches a configuration satisfying the condition (C1) or (C2) of Definition 2.

It is clear that S' reaches a configuration satisfying (C1) from any initial configuration in case of $n' = 1$.

Now assume that S' reaches a configuration satisfying (C1) or (C2) from any initial configuration in case of $n' = k - 1$ (inductive hypothesis), and consider the case of $n' = k$ (≥ 2).

From $n' \geq 2$, there exists a leaf process u in S' that is not neighboring the Byzantine process z. Let v be the neighbor of u. Since processes u and v are correct processes, we can show the following by the same argument as the fault-free case (Lemma 4): after some configuration ρ, (a) $prnt_v \neq u$ always holds, or (b) $prnt_v = u$ always holds. In case (a), we can show from the inductive hypothesis that S' eventually reaches a configuration satisfying (C1) or (C2). In case (b), we can show that S' eventually reaches a configuration satisfying (C2) where link (u, v) is the root link.

Consequently, protocol $ss\text{-}TO$ reaches a configuration of $\mathcal{LC}_1$ from any initial configuration. □

The following main theorem is obtained from Lemmas 1, 2, 3, 4 and 5.

Theorem 2: Protocol $ss\text{-}TO$ is a $(\Delta, 0, 1)$-strongly stabilizing tree-orientation protocol.

3.3.4 Round Complexity of $ss\text{-}TO$

In this subsection, we focus on the round complexity of $ss\text{-}TO$. First, we show the following lemma.

Lemma 6: Let v and u be any neighbors of S. Let S' be the subtree of $S - \{v\}$ containing u and $h(v, u)$

be the largest distance from v to a leaf process of S'. If $S' \cup \{v\}$ contains no Byzantine process, $prnt_v := u$ of GA1 or GA2 can be executed only in the first $2h(v,u)$ rounds. Moreover, in round $2h(v,u)$+1 or later, $level_v$ remains unchanged as long as $prnt_v = u$ holds.

Proof: We prove the lemma by induction on $h(v,u)$.

First consider the case of $h(v,u) = 1$, where u is a leaf process. When the first round completes, all the output registers of every process becomes consistent with the process variables. Since u is a leaf process, $prnt_u = v$ always holds. It follows that process v can execute $prnt_v := u$ only in GA1. Once v executes its action in the second round, $level_v \geq level_u$ holds and $prnt_v := u$ of GA1 cannot be executed afterward (see proof of Lemma 4). Thus, $prnt_v := u$ of GA1 can be executed only in the first and second rounds. It is clear that in round 3 or later, $level_v$ remains unchanged as long as $prnt_v = u$ holds.

We assume that the lemma holds when $h(v,u) \leq k-1$ (inductive hypothesis) and consider the case of $h(v,u) = k$. We assume that $prnt_v := u$ of GA1 or GA2 is executed in round r, and show that $r \leq 2k$ holds in the following. Variable $level_v$ is also incremented in the action, and let ℓ be the resultant value of $level_v$. In the following, we consider two cases.

- Case that $prnt_v := u$ of GA1 is executed in round r: when $prnt_v := u$ is executed, $level_u = \ell$ holds. But $level_u < \ell$ holds when v executes its action in round $r-1$; otherwise, v reaches a state with $level_v \geq \ell$ in round $r-1$ and cannot execute $prnt_v := u$ (with $level_v := \ell$) in round r. This implies that u incremented $level_u$ to ℓ in round $r-1$ or r.
  In the case that u makes the increment of $level_u$ by GA1, u executes $prnt_u := w$ for w ($\neq v$) in the same action. Since $h(u,w) < h(v,u)$ holds, the action is executed in the first $2h(u,w)$ rounds from the inductive hypothesis. Consequently, $prnt_v := u$ of GA1 is executed in round $2h(u,w)+1$ ($< 2h(v,u)$) at latest.
  In the case that u makes the increment of $level_u$ by GA2, u executes $prnt_u := w$ for some w ($\in N_u$) in the same action, where $w = v$ may hold. For the case of $w \neq v$, we can show, by the similar argument to the above, that $prnt_v := u$ is executed in round $2h(u,w)+1$ ($< 2h(v,u)$) at latest. Now consider the case of $w = v$. Then $level_v = level_u = \ell - 1$, $prnt_v \neq u$ and $prnt_u \neq v$ hold immediately before u executes $prnt_u := v$ and $level_u := \ell$. Between the actions of $level_u := \ell - 1$ (with $prnt_u := w$ ($w \neq v$)) and $level_u := \ell$ (with $prnt_u := v$), v can execute its action at most once; otherwise, $level_v \geq \ell - 1$ holds after the first action, and $level_v \geq \ell$ or $prnt_v = u$ holds after the second action. This implies that $level_u := \ell - 1$ with $prnt_u := w$ ($w \neq v$) is executed in the previous or the same round as the action of $level_u := \ell$, and thus, in round $r-2$ or later. Since $h(u,w) < h(v,u)$ holds, the action is executed in the first $2h(u,w)$ rounds from the inductive hypothesis. Consequently, $prnt_v := u$ of GA1 is executed in round $2h(u,w)+2$ ($\leq 2h(v,u)$) at latest.
- Case that $prnt_v := u$ is executed in GA2: then $level_v = level_u = \ell - 1$, $prnt_v \neq u$ and $prnt_u \neq v$ hold immediately before v executes $prnt_v := u$ and $level_v := \ell$. Between the executions of $level_v := \ell - 1$ and $level_v := \ell$, u can execute its action at most once, and u executes $prnt_u := w$ for some w ($\neq v$) in the action.Since $h(u,w) < h(v,u)$ holds, this action is executed in the first $2h(u,w)$ rounds from the inductive hypothesis. Consequently, $prnt_v := u$ is executed in round $2h(u,w)+1$ ($< 2h(v,u)$).

It remains to show that $level_v$ remains unchanged in round $2h(v,u)$+1 or later, as long as $prnt_v = u$ holds. Now assume that $prnt_v = u$ holds at the end of round $2h(v,u)$.

- Case that $prnt_u = v$ holds at the end of round $2h(v,u)$: since $h(u,w) < h(v,u)$ for any $w \in N_u - \{v\}$, $prnt_u := w$ cannot be executed in round $2h(v,u)+1$ or later from the inductive hypothesis, and so $prnt_u = v$ holds afterward. Thus, it is clear that $level_v$ remains unchanged as long as $prnt_v = u$ (and $prnt_u = v$) holds.
- Case that $prnt_u \neq v$ holds at the end of round $2h(v,u)$: let $prnt_u = w$ hold for some $w \in N_u - \{v\}$ at the end of round $2h(v,u)$. Since $h(u,w) < h(v,u)$, $level_u$ remains unchanged as long as $prnt_u = w$ holds from the inductive hypothesis. It follows that $level_v$ remains unchanged as long as $prnt_v = u$ and

$prnt_u = w$ hold. Since $h(u,x) < h(v,u)$ for any $x \in N_u - \{v\}$, $prnt_u := x$ cannot be executed in round $2h(v,u)+1$ or later, but $prnt_u := v$ can be executed. Immediately after execution of $prnt_u := v$, $level_v = level_u$ holds if $prnt_v$ remains unchanged. Thus, it is clear that $level_v$ remains unchanged as long as $prnt_v = u$ (and $prnt_u = v$) holds.

□

The following lemma holds for the fault-free case.

Lemma 7: In a fault-free tree system, protocol *ss-TO* reaches a configuration of $\mathcal{LC}_0$ from any initial configuration in $O(d)$ rounds where d is the diameter of the tree system S.

Proof: Lemma 6 implies that, after round $2d+1$ or later, no process v changes $prnt_v$ or $level_v$ and thus the configuration remains unchanged. Lemma 4 guarantees that the final configuration is a configuration in $\mathcal{LC}_0$. □

For the single-Byzantine case, the following lemma holds.

Lemma 8: In a tree system with a single Byzantine process, protocol *ss-TO* reaches a configuration of $\mathcal{LC}_1$ from any initial configuration in $O(n)$ rounds.

Proof: Let z be the Byzantine process and S' be any subtree of $S-\{z\}$. Let v be the neighbor of z in S'. From Lemma 6, v cannot execute $prnt_v := w$ for any $w \in N_v - \{z\}$ in round $2d'+1$ or later, where d' is the diameter of S'. We consider the following two cases depending on $prnt_v$.

- Case 1: there exists $w \in N_v - \{z\}$ such that $prnt_v = w$ at the end of round $2d'$ and $prnt_v$ remains unchanged during the following d' rounds (from round $2d'+1$ to round $3d'$).
  From Lemma 6, $level_v$ also remains unchanged during the d' rounds. By the similar discussion to that in proof of Lemma 6, we can show that S' reaches a configuration satisfying the condition (C2) of Definition 2 by the end of round $3d'$.
- Case 2: $prnt_v = z$ at the end of round $2d'$ or there exists at least one configuration during the following d' rounds (from round $2d'+1$ to round $3d'$) such that $prnt_v = z$ holds.
  Let c be the configuration where $prnt_v = z$ holds. From Lemma 6, $prnt_v = z$ always holds after c. We can show, by induction of k that, a fraction of S' consisting of processes with distance up to k from v satisfies the condition (C1) at the end of k rounds after c. Thus, S' reaches a configuration satisfying the condition (C1) of Definition 2 by the end of round $4d'$.

After a subtree reaches a configuration satisfying the condition (C2), its configuration may change into one satisfying the condition (C1) and the configuration may not satisfy (C1) or (C2) during the transition. However, Lemma 3 guarantees that the length of the period during the subtree does not satisfy (C1) or (C2) is $O(d')$ rounds, where d' is the diameter of the subtree. Since the total of diameters of all the subtrees in $S-\{z\}$ is $O(n)$, the convergence to a configuration of $\mathcal{LC}_1$ satisfying (C1) or (C2) can be delayed at most $O(n)$ rounds.

□

Finally, we can show the following theorem.

Theorem 3: Protocol *ss-TO* is a $(\Delta, 0, 1)$-strongly stabilizing tree-orientation protocol. The protocol reaches a configuration of $\mathcal{LC}_0 \cup \mathcal{LC}_1$ from any initial configuration. The protocol may move from a legitimate configuration to an illegitimate one because of the influence of the Byzantine process, but it can stay in illegitimate configurations during the total of $O(n)$ rounds (that are not necessarily consecutive) in the whole execution.

Proof: Theorem 2 shows that *ss-TO* is a $(\Delta, 0, 1)$-strongly stabilizing tree-orientation protocol. Lemma 7 and 8 guarantee that *ss-TO* reaches a configuration of $\mathcal{LC}_0 \cup \mathcal{LC}_1$ from any initial configuration within $O(n)$ rounds. For the case with a single Byzantine process (say z), each subtree of $S-\{z\}$ may experience an illegitimate period (not satisfying the condition (C1) or (C2)) after such a configuration. However, Lemma 3 guarantees that the length of the illegitimate period is $O(d')$ where d' is the diameter of the subtree. Since the total of diameters of all the subtrees in $S-\{z\}$ is $O(n)$, the total length of the periods that does not satisfy (C1) or (C2) is $O(n)$ rounds. □

REFERENCES

[1] James E. Burns, Mohamed G. Gouda, and Raymond E. Miller. Stabilization and pseudo-stabilization. *Distributed Computing*, 7(1):35–42, 1993.

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

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