Communication-efficient Self-stabilizing Protocols for Spanning-Tree Construction

Toshimitsu Masuzawa†* Taisuke Izumi‡ Yoshiaki Katayama‡ Koichi Wada‡

†Osaka University, Japan, masuzawa@ist.osaka-u.ac.jp
‡Nagoya Institute of Technology, Japan, {t-izumi, katayama, wada}@nitech.ac.jp

Abstract

A self-stabilizing protocol can eventually recover its intended behavior even when started from an arbitrary configuration. Most of self-stabilizing protocols require every pair of neighboring processes to communicate with each other repeatedly and forever even after converging to legitimate configurations. Such permanent communication impairs efficiency, but is necessary in nature of self-stabilization: if we allow a process to stop its communication with other processes, the process may initially start and remain forever at a state inconsistent with the states of other processes. So it is challenging to minimize the number of process pairs communicating after convergence.

We investigate possibility of *communication-efficient* self-stabilization, which allows only $O(n)$ pairs of neighboring processes to communicate repeatedly after convergence. For spanning-tree construction, we show the following results: (a) communication-efficiency is attainable when a unique root is designated a priori, (b) communication-efficiency is impossible to attain when each process has a unique identifier but without a designated unique root, and (c) communication-efficiency becomes attainable with process identifiers if each process initially knows an upper bound of the network size.

Keywords self-stabilization, fault-tolerance, spanning-tree construction, communication-efficiency

1 Introduction

A self-stabilizing protocol [9] can eventually recover its intended behavior without external intervention even when started from an arbitrary configuration (or global state). Because of its high and autonomous adaptability to transient faults and dynamical topology changes of networks, self-stabilization attains much attention of designers and practioners of distributed systems.

The high adaptability of self-stabilizing protocols is usually acquired at the cost of efficiency. A main concern in efficiency considered so far is efficiency *in convergence* after faults, i.e., the *stabilization time* that is the maximum amount of time required to recover its intended behavior after faults occur. This metric is undoubtedly meaningful to evaluate efficiency in adaptation to faults. However, another crucial difference in cost between self-stabilizing protocols and classical (or non-self-stabilizing) ones lies in the cost required for communication *after convergence*. The difference is quite evident in protocols for static problems such as leader election and spanning-tree construction. Self-stabilizing protocols cannot allow any process to terminate its communication after convergence, while classical ones can allow every process to terminate all the activity. So the termination requirement we can expect for self-stabilizing protocols (for static problems) is

*Contact author: Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka, 565-0871, Japan, FAX: +81-6-6879-4119

at the best the *silence property*, which implies that no process changes its state after convergence. Even in silent protocols, processes are required to communicate with neighboring processes to check consistency repeatedly and forever; otherwise, a process may initially start and remain forever at a state inconsistent with the states of other processes, and the protocols cannot guarantee convergence to legitimate configurations.

Most of self-stabilizing protocols proposed so far require every process to communicate with every neighboring process repeatedly after convergence. This leads high communication load in networks and make self-stabilizing protocols unacceptable in some real situations. Only a few papers [7, 8] are dedicated to improving communication efficiency after convergence; a self-stabilizing protocol is said to be *communication-efficient* if only a small number of process pairs communicate forever.

Contribution of this work In this paper, we introduce $\diamond$-k-*communication-efficiency* as an efficiency measure of self-stabilization after convergence. We investigate possibility of $\diamond$-k-communication-efficient self-stabilization for *spanning-tree construction* and show the following results.

(a) When a unique root is designated a priori, there exists a self-stabilizing protocol that allows each process, after convergence, to read the state only from its parent (i.e., $\diamond$-$(n-1)$-communication-efficient where n is the number of processes in the distributed system).

(b) When each process has a unique identifier but a unique root is not predetermined, no protocol can allow even a single pair of processes to eventually stop communication between them (i.e., $\diamond$-$o(m)$-communication-efficiency is unattainable where m is the number of links in the distributed system).

(c) When each process has a unique identifier and knows an upper bound N ($n \leq N < 2n$) of n a priori (but a unique root is not predetermined), there exists a protocol that allows each process, after convergence, to read the states only from its parent and children (i.e., $\diamond$-$2(n-1)$-communication-efficient). The restriction $N < 2n$ on the upper bound N is the weakest for attaining communication-efficiency because communication-efficiency becomes unattainable when $N = 2n$.

The results (a) and (b) bring out the contrast between self-stabilization and communication-efficient self-stabilization: these show that existence of a unique root is sufficient for attaining communication-efficiency but existence of unique process identifiers is not, however, self-stabilizing (but not communication-efficient) spanning-tree construction is possible with either assumption.

Related works Anguilera *et al.* [2] introduced the concept of *communication-efficiency* in the implementation of failure detector Ω[1] in partially synchronous systems. Following the work, some papers investigated possibility of communication-efficient implementations of failure detector Ω (e.g., [3, 4, 6, 11]). The implementations in [2, 3, 11] can tolerate any number of crash processes and require only $n-1$ unidirectional links to carry messages forever. Anguilera *et al.* [4] presented another implementation that requires only f unidirectional links to carry messages forever where at most f processes may crash. Biely and Widder [6] presented a message-driven implementation (or not relied on timers) of Ω, which is also communication-efficient in the sense that only $f+2$ processes broadcast messages forever where at most f processes may crash.

[1] Roughly speaking, failure detector Ω eventually provides all processes with the identifier of a unique correct process (i.e., a leader).

Delporte-Gallet *et al.* [7] considered self-stabilizing leader election that can tolerate process crashes as well as transient faults. They presented an algorithm in the fully-synchronous system that requires only $n-1$ unidirectional links to carry messages forever.

The above communication-efficiency considers *global* communication and aims to reduce the *total* number of links carrying messages. When we consider communication activity of individual processes, all the above algorithms require a process (or $f+2$ processes in [6]) to repeatedly broadcast messages to all other processes (or f other processes in [4]). Devismes *et al.*[8] focused on communication-efficiency with a *local* criterion. They considered the shared-state model and introduced the concept of $\diamond$-k-*stability* to self-stabilization that allows each process to repeatedly read the states only from k neighbors after convergence. They investigated possibility of the $\diamond$-k-stability for the maximal independent set and the maximal matching.

Spanning-tree construction is one of the most investigated subjects in self-stabilization and many protocols have been presented (see [10] for a survey). But most of the protocols require every process to communicate repeatedly with its every neighbor after convergence. To the best of our knowledge, no previous paper has shed light on communication-efficiency of self-stabilizing spanning-tree construction.

2 Preliminaries

2.1 System model

A *distributed system* $\mathcal{S} = (P, L)$ consists of set $P = \{v_0, v_1, \ldots, v_{n-1}\}$ of processes and set L of bidirectional (communication) links. A link connects two distinct processes. When a link connects processes v and w, this link is denoted by (v, w). We say w is a *neighbor* of v if $(v, w) \in L$ and the set of neighbors of v is denoted by N_v. In this paper, we interchangeably use the terms, a distributed system and a network.

Each process v is a state machine and its action is defined by *guarded actions* of the form $\langle guard_v \rangle \longrightarrow \langle statement_v \rangle$. The guard $\langle guard_v \rangle$ of process v is a Boolean expression on its own state and its neighbors' states. When the guard evaluates true, $\langle statement_v \rangle$ is executed to change the state of v. This model is called the *shared-state model*. For convenience, we use *variables* of each process to define the process state and the variables read by a neighbor are called *communication variables*.

A *configuration* (i.e., a global state) of a distributed system is specified by an n-tuple $\sigma = (s_0, s_1, \ldots, s_{n-1})$ where s_i stands for the state of process v_i. A process v is said to be *enabled* at a configuration σ when v has a guarded action whose guard is true at σ. A process v is said to be *disabled* at σ when it is not enabled at σ.

Let $\sigma = (s_0, s_1, \ldots, s_{n-1})$ and $\sigma' = (s'_0, s'_1, \ldots, s'_{n-1})$ be configurations and Q be any set of processes. We denote the transition from σ to σ' by $\sigma \overset{Q}{\mapsto} \sigma'$, when σ changes to σ' by actions of every enabled process in Q (or all the enabled processes in Q make actions but no other processes make actions). Reading states and executing actions are done atomically. We sometimes simply denote $\sigma \mapsto \sigma'$ without specifying the set of processes Q.

A *schedule* is an infinite sequence $\mathcal{Q} = Q_1, Q_2, \ldots$ of nonempty sets of processes. In this paper, we assume that any schedule is *weakly fair*, that is, all processes appear infinitely often in any schedule. If an infinite sequence of configurations $E = \sigma_0, \sigma_1, \sigma_2, \ldots$ satisfies $\sigma_j \overset{Q_{j+1}}{\mapsto} \sigma_{j+1}$ $(j \geq 0)$, then E is called an *execution* starting from σ_0 by schedule $\mathcal{Q}$.

A schedule specifies the order of processes that are activated to execute their guarded actions. In this paper, we consider an *asynchronous* distributed system where we make no assumption on

the speed of processes. This implies that all weakly fair schedules are possible to occur.

To evaluate the time complexity of protocols, we introduce (asynchronous) *rounds* for an execution E. Let $E = \sigma_0, \sigma_1, \sigma_2, \ldots$ be an execution by a schedule $\mathcal{Q} = Q_1, Q_2, \ldots$. The first round of E is defined to be the minimal partial execution $\sigma_0, \sigma_1, \ldots, \sigma_k$ that satisfies $P = \cup_{1 \leq j \leq k} Q_j$. The second and later rounds of E are defined recursively for the execution $\sigma_k, \sigma_{k+1}, \sigma_{k+2}, \ldots$.

2.2 Self-stabilizing protocols

Spanning tree construction considered in this paper is a so-called *static problem*, i.e. it requires the system to find and hold a static solution. For clarity, we introduce *output variables* at each process and consider a problem is defined by a *specification predicate* over the output variables of all the processes. A configuration is said to be *legitimate* if the output variables of all the processes satisfy the specification predicate, i.e., the system holds a solution of the problem.

A *self-stabilizing protocol* (for a static problem) is defined by two requirements, *convergence* and *closure*. The convergence requires that the protocol eventually reaches a legitimate configuration. The closure requires that every process never changes its output variables at a legitimate configuration. This implies that the protocol remains at legitimate configurations once it reaches a legitimate configuration. A self-stabilizing protocol is said to be *silent* if the states (including all variables) of all processes eventually remain unchanged.

Problem specification The *spanning-tree construction* problem is defined as follows. Each process v has an output variable $prnt_v$ to designate one of its neighbors as a parent. For the root process r, $prnt_r = \perp$ holds to denote that it has no parent. The specification predicate is satisfied if and only if variables $prnt_v$ of all the processes form a spanning-tree of the distributed system.

2.3 Communication-efficiency

Let $E = \sigma_0, \sigma_1, \ldots$ be an execution by a schedule $\mathcal{Q} = Q_1, Q_2, \ldots$. For each process $v \in Q_i$, let $R_v^i(E)$ be the set of v's neighbors from which v reads some communication variables in the step from σ_{i-1} to σ_i. Let $R_v(E) = R_v^1(E) \cup R_v^2(E) \cup \cdots$.

Definition 1 ($\diamond$-k-communication-efficiency) *A protocol is* $\diamond$-k-communication-efficient *if in every execution E, there is a suffix E' such that $\sum_{i=0}^{n-1} |R_{v_i}(E')| \leq k$.* □

Most of self-stabilizing protocols proposed so far are $\diamond$-$2m$-communication efficient where $m = |L|$: every process repeatedly reads some communication variables from all of its neighbors. Previous works such as [2, 3, 4, 7, 11] say a protocol is communication efficient if only $n-1$ unidirectional communication links are used, and thus, such a protocol is $\diamond$-$(n-1)$-communication-efficient.

While the $\diamond$-k-communication-efficiency focuses on the total number of communicating process pairs, Devismes *et al.* [8] introduced the concept of $\diamond$-k-*stability* with a local criterion of communication-efficiency.

Definition 2 ($\diamond$-k-Stable) [8] *A protocol is* $\diamond$-k-stable *if in every computation E, there is a suffix E' such that every process v satisfies $|R_v(E')| \leq k$.* □

Protocol 3.1 A communication-efficient self-stabilizing protocol *Tree-R*

```
communication variables of process v
    d_v: integer; /* distance from the root r on the constructed tree

local variables of process v
    prnt_v: ⊥ or a neighbor of v; /* prnt_v = u ∈ N_v if u is a parent of v.
    consistent_v: bool; /* consistent_v = true iff v is locally consistent

actions of root process v(= r)
    true ⟶ prnt_v = ⊥; d_v = 0; consistent_v := true;

actions of non-root process v(≠ r)
    switch (consistent_v)
        case true:
            d_v ≠ d_{prnt_v} + 1 ⟶ consistent_v := false;
            /* v reads only d_{prnt_v} from prnt_v
        case false:
            true ⟶
                prnt_v := w (∈ N_v) s.t. d_w = min{d_u | u ∈ N_v};
                d_v := d_{prnt_v} + 1; consistent_v := true;
            /* v reads d_u from every neighbor u
```

3 Communication-efficient Spanning-Tree Construction

3.1 Protocol *Tree-R* on rooted networks

Protocol 3.1 presents a $\diamond$-$(n-1)$-communication-efficient and $\diamond$-1-stable self-stabilizing protocol *Tree-R* for constructing a spanning-tree on arbitrary networks. The protocol assumes existence of a *unique root* process, say r.

Protocol *Tree-R* is a silent protocol, that is, every process eventually stops changing its state. In the final configuration, variables *prnt* of all the processes form a spanning-tree and d_v of each process v stores the distance from the root r on the tree. The main idea for attaining the communication-efficiency is that every process v other than r checks its consistency only by comparing its variable d_v with that of its parent.

In the protocols presented in this paper, we use `switch` statements on variable $consistent_v$ to clarify the process behavior at legitimate configurations. In the legitimate configurations, variable $consistent_v$ stores `true` and each process v executes the actions corresponding to the value `true`. For example, in protocol *Tree-R*, non-root process v reads only d_{prnt_v} from $prnt_v$.

Legitimate configurations of protocol *Tree-R* are defined as follows.

Definition 3 (Legitimate configuration) *A configuration σ of protocol Tree-R is* legitimate *if σ satisfies the following conditions.*

1. *For the root process r, $prnt_r = \perp$, $d_r = 0$ and $consistent_r =$ `true` hold.*
2. *For every process v other than r, $prnt_v \in N_v$, $d_v = d_{prnt_v} + 1$ and $consistent_v =$ `true` hold.*

□

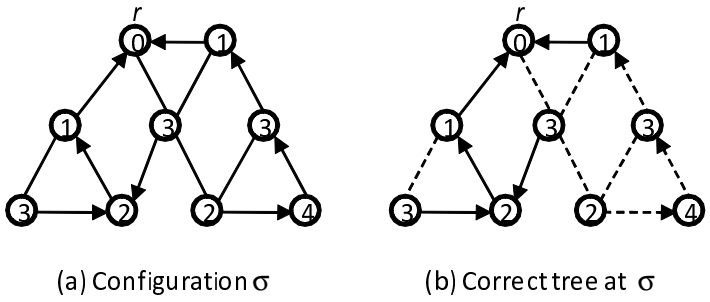

Figure 1: A correct tree of protocol *Tree-R*

The following lemma on the closure and the silence clearly holds.

Lemma 1 (Closure and silence) *Once protocol Tree-ID reaches a legitimate configuration, it remains at the configuration forever.* □

To show the convergence, we introduce the concept of *correct (partial) tree*.

Definition 4 (Correct tree) *Let σ be any configuration of protocol Tree-R. We define a* correct tree T_σ *at σ as follows.*

1. *The root process r is included in T_σ if and only if $prnt_r = \perp$, $d_r = 0$ and $consistent_v =$* `true` *hold.*
2. *A process v other than r is included in T_σ if and only if $prnt_v$ is included in T_σ, and $prnt_v \in N_v$, $d_v = d_{prnt_v} + 1$ and $consistent_v =$* `true` *hold.* □

Figure 1 presents an example of a correct tree. In the figure, the process designated by variable $prnt_v$ of each process v is denoted by an arrow and the value of d_v is described in the process. We assume $consistent_v =$ **true** holds for $v = r$ absolutely and for non-root process v if $d_v = d_{prnt_v} + 1$ holds. Figure 1 (a) presents a configuration, say σ, and Figure 1 (b) presents the correct tree at σ by solid arrows.

Lemma 1 can be easily extended to the following lemma on the correct trees.

Lemma 2 (Stability of correct tree) *Once a process, say v, is included in the correct tree T_σ at a configuration σ, v never changes its state (i.e., the values of $prnt_v$, d_v and $consistent_v$) after σ.* □

Lemma 2 implies that the correct tree never shrinks as execution of protocol *Tree-R* proceeds.

Lemma 3 (Convergence) *Starting from any configuration, protocol Tree-R eventually reaches a legitimate configuration.*

Proof It is sufficient to show that the correct tree eventually covers the whole network. Assume, for contradiction, that the correct tree never covers the whole network. This implies from Lemma 2 that we have the *final* correct tree T, which does not cover the whole network. Let P_T and $P_{\overline{T}}$ be sets of processes in T and out of T respectively.

Let σ be a configuration where the final correct tree is constructed. Consider any configuration σ' after σ, and let $d^{min}_{\sigma'}$ be the minimum d_v value among all the processes in $P_{\overline{T}}$ at σ'. No process

$v \in P_{\overline{T}}$ can assign its neighbor $u \in N_v \cap P_T$ to $prnt_v$; otherwise v becomes a member of the correct tree, which contradicts the assumption that the correct tree never changes after σ. This implies that every process $v \in P_{\overline{T}}$ eventually chooses (or keeps) a process $w \in P_{\overline{T}}$ as its parent and assigns (or keeps) a value greater than $d^{min}_{\sigma'}$ to d_v. Consequently, the value of d_v at every process $v \in P_{\overline{T}}$ grows unboundedly. This makes a process $v \in P_{\overline{T}}$ neighboring to a process in P_T eventually choose $u \in P_T$ as its parent since u eventually becomes to have the minimum value of d_u among all the neighbors of v. This implies that a process $v \in P_{\overline{T}}$ is newly involved into the correct tree, which is a contradiction. □

Theorem 1 *Protocol Tree-R is a self-stabilizing silent protocol for constructing a spanning-tree in an arbitrary network with a unique root. It is $\diamond$-$(n-1)$-communication-efficient and $\diamond$-1-stable, and its stabilization time is $O(n)$ rounds.*

Proof It is clear from Lemmas 1 and 3 that *Tree-R* is a self-stabilizing silent protocol for constructing a spanning-tree in an arbitrary network with a unique root. It is also clear that, in any legitimate configuration, the root r never executes a read action and every process v other than r reads only the state of its parent, and thus, the protocol is $\diamond$-$(n-1)$-communication-efficient and $\diamond$-1-stable.

At any configuration σ, let d^{min}_{σ} be the minimum value of d_v among all the processes out of the correct tree at σ. By an argument similar to that in the proof of Lemma 3, we can see that d^{min}_{σ} becomes larger than $n-1$ in the first $2n$ rounds of any execution. Note that each process v updates d_v only when $consistent_v =$ `false` holds, which can be executed at most every two rounds. This implies that *Tree-R* reaches a configuration in $2n$ rounds where any process v out of the correct tree has a greater value of d_v than any process u in the correct tree, since the value of d_u cannot grow beyond $n-1$ in the correct tree. In any execution from such a configuration, at least one process is newly involved into the correct tree every two rounds until the correct tree covers the whole network. Consequently, the stabilization time is at most $4n$ rounds. □

3.2 Impossibility on networks with process IDs

Protocol *Tree-R* assumes existence of a unique root process. Instead of the assumption, we sometimes assume for self-stabilizing spanning-tree construction that processes have distinct identifiers; the process with the minimum (or maximum) identifier is elected as the root and the spanning-tree is constructed using the root. Self-stabilizing spanning-tree construction is possible with the assumption of distinct identifiers (but without a predetermined unique root) [1]. However, the following theorem shows that $\diamond$-$o(m)$-communication-efficiency is impossible to attain under the assumption, where m is the number of links in the distributed system.

Theorem 2 *Any self-stabilizing protocol for constructing a spanning-tree in a non-rooted network with distinct process identifiers requires repeated communication between every pair of neighbors even after convergence to a legitimate configuration.*

Proof Assume for contradiction that there exists a self-stabilizing spanning-tree construction protocol $\mathcal{A}$ such that there exists a pair of processes in every network that never communicate after the convergence. That is, in every execution, there exists a pair of processes such that they never read the state of each other after a legitimate configuration.

Consider two networks $\mathcal{S}_1 = (P_1, L_1)$ and $\mathcal{S}_2 = (P_2, L_2)$ and assume that $P_1 \cap P_2 = \emptyset$. In the executions of protocol $\mathcal{A}$, there exists a pair of neighbors, say u_1 and v_1 in $\mathcal{S}_1$ (resp. u_2 and v_2 in

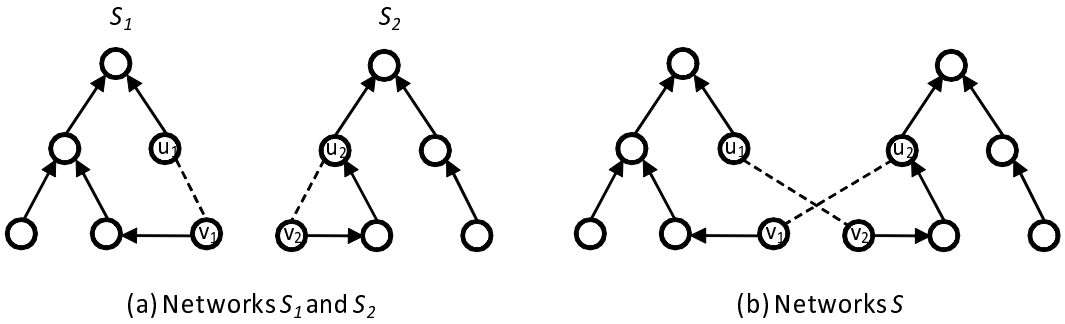

Figure 2: Network $\mathcal{S}$ constructed from $\mathcal{S}_1$ and $\mathcal{S}_2$

$\mathcal{S}_2$), such that the processes never communicate with each other after a legitimate configuration σ_1 (resp. σ_2).

Now consider a network $\mathcal{S} = (P, L)$ where $P = P_1 \cup P_2$ and $L = ((L_1 \cup L_2) - \{(u_1, v_1), (u_2, v_2)\}) \cup \{(u_1, v_2), (u_2, v_1)\}$ (Fig. 2). Consider an initial configuration of $\mathcal{S}$ where processes in P_1 and P_2 are at the same states as in σ_1 and σ_2 respectively. Each process of $\mathcal{S}$ behaves in the same way as they are in $\mathcal{S}_1$ or $\mathcal{S}_2$, and thus, this contradicts the assumption that protocol $\mathcal{A}$ constructs a spanning-tree in an arbitrary network. □

3.3 Protocol *Tree-ID* on networks with process IDs

In this subsection, we first show that communication-efficiency is attainable if each process has a unique identifier and initially knows the number n of processes in the network. Then, we briefly show how to modify the protocol so that the protocol can work with an upper bound N ($0 \leq N < 2n$) of n. It is easy to see that the proof of Theorem 2 does not hold if each process knows the network size: the two configurations of the proof become distinguishable and the contradiction cannot be derived.

Protocol 3.2 presents a $\diamond$-$2(n-1)$-communication-efficient self-stabilizing protocol *Tree-ID* for constructing a spanning-tree on arbitrary networks. The protocol assumes that each process v has a unique identifier id_v and knows the number n of processes in the network. Each process v has variables $root_v$, $size_v$ and $CHLD_v$ in addition to the variables used in protocol *Tree-R*. Variable $root_v$ stores the root identifier of the tree v currently belongs to, $CHLD_v$ stores the children of v, and $size_v$ stores the number of processes in the subtree rooted at v.

The strategy for attaining the communication-efficiency is the same as protocol *Tree-R*: every process v other than r checks its consistency only by comparing its variable d_v with that of its parent. However, this strategy alone is not sufficient. Consider the initial configuration where multiple trees exist, and variables $prnt_v$ and d_v of every process are consistent. If there exists a pair of processes that never communicate with each other, the processes may fail to detect existence of other trees (in the same way as the case of the proof of Theorem 2).

To circumvent the difficulty, we make use of the number n of processes: the root finds the number of processes in its tree and allows each process in the tree to communicate with all of its neighbors until the tree covers the whole network. When a process finds another tree rooted at a process with a larger identifier than its current root, the process changes its parent to join the newly found tree. Consequently, the tree rooted at the process r with the maximum identifier eventually covers the whole network, which is detected when $size_r = n$ holds. The variable $consistent_v$ is used to relay the information whether the tree covers the whole network or not.

After convergence to a legitimate configuration, each process reads the states of its children as well as that of its parent, and thus, the protocol *Tree-ID* is $\diamond$-$2(n-1)$-communication efficient.

Definition 5 (Legitimate configuration) *A configuration σ of protocol Tree-ID is* legitimate *if σ satisfies the following conditions.*

1. *For the process, say r, with the maximum identifier in the network, $root_r = id_r$, $prnt_r = \perp$ and $d_r = 0$ hold.*
2. *For every process v other than r, $root_v = id_r$, $prnt_v \in N_v$ and $d_v = d_{prnt_v} + 1$.*
3. *For every process u, $CHLD_u = \{w \in N_u \mid prnt_w = id_u\}$, $size_u =$ the number of processes in the subtree rooted at u, and $consistent_u =$ `true`.* □

The following lemma on the closure and the silence clearly holds.

Lemma 4 (Closure and silence) *Once protocol Tree-ID reaches a legitimate configuration, it remains at the configuration forever.* □

Lemma 5 (Convergence) *Starting from any configuration, protocol Tree-ID eventually reaches a legitimate configuration in $O(n)$ rounds.*

Proof sketch Initially *fake identifiers* (or identifiers of non-existing processes in the network) may be stored in variable *root* of processes. Such fake identifiers are removed in $O(n)$ rounds since the attached distances increase by at least one in every two rounds and the identifier is ignored when the distance becomes $n-1$. In the following, we consider execution after all the fake identifiers are removed.

Once the process r with the maximum identifier is activated, variables $root_r$, $prnt_r$ and d_r are set to id_r, $\perp$ and 0 respectively and remain unchanged afterward. It follows that any tree rooted at a process, say s, other than r cannot contain all the processes, and thus, s sets $consistent_s$ to `false` in $O(n)$ rounds and keeps it as long as $prnt_s = \perp$. When the `false` value is relayed to a process, say u, u sets variable $root_u$ to the maximum identifier stored in *root* of its neighbors. From this fact, we can show that $root_u = id_u$ holds at every process u in $O(n)$ rounds. On the other hand, by an argument similar to that for the fake identifiers, we can show that all the cycles formed by variables *prnt* are removed in $O(n)$ rounds. Thus, in $O(n)$ rounds, the protocol reaches a configuration, say σ, where $root_u = id_r$ holds at every process u and variables *prnt* of all the processes form no cycle; i.e., *prnt* of all the processes form a spanning-tree rooted at r.

After configuration σ, each process u may change the value of $prnt_u$ when the value of d_{prnt_u} becomes $n-1$. This implies that there exists the gap of distance (i.e., $d_v > d_{prnt_v} + 1$) at some process v on the path from the root r to $prnt_u$. Since the minimum value of gap-causing distance (i.e., d_{prnt_v} in the above case) increases by one in every two rounds, such gaps disappear in $O(n)$ rounds. Thus, the protocol reaches a configuration in $O(n)$ rounds after which no process changes the value of $prnt_u$. When the value of $prnt_u$ is fixed at each process u, it is clear that the protocol reaches a legitimate configuration in $O(n)$ rounds. □

In protocol *Tree-ID*, every process repeatedly reads the states only from its parent and children at a legitimate configuration, thus, the protocol is $\diamond$-$2(n-1)$-communication-efficient. The following theorem is immediately derived from Lemmas 4 and 5.

Theorem 3 *Protocol Tree-ID is a self-stabilizing silent protocol for constructing a spanning-tree in arbitrary networks where each process has a distinct identifier and knows the number n of processes a priori. It is $\diamond$-$2(n-1)$-communication-efficient and its stabilization time is $O(n)$ rounds.*

Protocol 3.2 A communication-efficient self-stabilizing protocol *Tree-ID*

```
constants of process v
    id_v: identifier; /* identifier of v
    n: integer; /* the number of processes in the network

communication variables of process v
    prnt_v: identifier or ⊥; /* prnt_v = id_u (u ∈ N_v) if u is a parent of v.
    root_v: identifier; /* identifier of the root
    d_v: integer; /* distance from the root on the constructed tree
    size_v: integer; /* the number of processes in the subtree rooted at v
    consistent_v: bool; /* consistent_v = true iff v is locally consistent

local variables of process v
    CHLD_v: subset of neighbors of v; // set of v's children

actions of process v
    switch(consistent_v)
        case true
            ((prnt_v = ⊥ and (root_v ≠ id_v or d_v ≠ 0 or size_v ≠ n)) or
            (prnt_v ≠ ⊥ and (root_v < id_v or root_v ≠ root_{prnt_v} or d_v ≠ d_{prnt_v} + 1 ))or
            ∃u ∈ CHLD_v[prnt_u ≠ id_v] or size_v ≠ Σ_{u∈CHLD_v} size_u + 1 or
            consistent_{prnt_v} = false)
            ⟶ consistent_v := false;
        case false
            true ⟶
                root_v := max({root_u | u ∈ N_v, d_u ≤ n − 2} ∪ {id_v});
                if root_v = id_v then
                    prnt_v := ⊥; d_v := 0;
                else if (prnt_v ≠ ⊥ and root_{prnt_v} = root_v and d_{prnt_v} ≤ n − 2)
                    d_v := d_{prnt_v} + 1;
                else
                    prnt_v := id_u s.t. d_u = min{d_w | w ∈ N_v, root_w = root_v};
                    d_v := d_{prnt_v} + 1;
                CHLD_v := {u ∈ N_v | prnt_u = id_v};
                size_v := Σ_{u∈CHLD_v} size_u + 1;
                if ((prnt_v = ⊥) and (size_v = n)) or
                    ((prnt_v ≠ ⊥) and (consistent_{prnt_v} = true))) then
                    consistent_v :=true;
```

Protocol *Tree-ID* utilizes the number n of processes to detect completion of spanning-tree construction. Until the spanning-tree is constructed, every process communicates with all of its neighbors to detect another existing tree. When the root r detects the completion, it transfers each process to the communication-saving mode by announcing the completion using variable *consistent*.

When each process initially knows the identifiers of all the neighbors instead of n, $\diamond$-$2(n-1)$-communication-efficiency is attainable: by broadcasting the process identifiers in the constructed tree, each process need not communicate with neighbors connected by non-tree links. Thus, at a legitimate configuration, each process communicates only with its parent and children.

Protocol using an upper bound of the network size Protocol *Tree-ID* uses the *exact* number n of processes in the network, however, it can be modified to work with an *upper bound* N of n satisfying $n \leq N < 2n$.

The key observation for the modification is that only a single tree can contain the majority of the processes. This leads us to the following strategy for attaining communication-efficiency: to reduce the communication after convergence, a process reads the states only from its parent and (a subset of) its children when it belongs to a tree (called a *major tree*) consisting of $N/2$ processes or more. On the other hand, to detect another tree, a process reads the states from all the neighbors when it belongs to a tree (called a *minor tree*) with less than $N/2$ processes. When a process in a minor tree finds a major tree in its neighbors, the process changes its parent to join the major tree.

In the modified protocol, each process has the same variables and executes almost the same code as protocol *Tree-ID*. The outline of the modification is as follows.

1. To transfer processes into the communication-saving mode, we should detect that a major tree is formed, instead of detecting that spanning-tree construction is completed: the condition $size_v = n$ (resp. $size_v \neq n$) for consistency (resp. inconsistency) at a root v (such that $prnt_v = \perp$) is modified to $size_v \geq N/2$ (resp. $size_v < N/2$).

2. Every process has possibility to become the root of the spanning-tree while the process with the maximum identifier necessarily becomes the root in protocol *Tree-ID*: the condition $root_v < id_v$ for inconsistency at a non-root process v is removed.

3. When a process v finds a major tree in its neighbors, v should join the major tree: when process v changes the value of $root_v$, it adopts the value of $root_u$ if $consistent_u =$`true` holds at a neighbor u (which implies u is contained in the major tree). If such a neighbor does not exist, a larger identifier has higher priority as in protocol *Tree-ID*.

By the above strategy, we can design a $\diamond$-$2(n-1)$-communication-efficient self-stabilizing protocol for the spanning-tree construction.

Remark 1: Distinct from protocol *Tree-ID*, variable $CHLD_v$ does not store all the children of v but stores a subset of the children. This is caused as follows. Let v be a process in the major tree and have set $consistent_v$ to `true`. Let u be its neighbor in another tree. When u finds v, it joins the major tree by setting $prnt_u$ to id_v. But v reads the states only from its parent and processes in $CHLD_v$, v cannot detect the new child u and cannot add u to $CHLD_v$.

Remark 2: Some self-stabilizing (but not communication-efficient) protocols for spanning-tree construction (e.g., [5]) use an upper bound N of the number n of processes. The upper bound is utilized to detect fake identifiers and cycles (formed by variables *prnt*) that exist in the initial configuration. The protocols work with any upper bound N.

Distinct from these protocols, we make restriction on N such that $N < 2n$. This restriction is necessary to attain the communication-efficiency. Actually by an argument similar to that in the

proof of Theorem 2, we can show that the communication-efficiency is unattainable when $N = 2n$: every pair of processes have to communicate repeatedly and forever. In this sense, the restriction of $N < 2n$ is the weakest for attaining communication-efficiency.

4 Conclusions

We introduced $\diamond$-*k-communication-efficiency* as an efficiency measure of self-stabilization after convergence. We investigated possibility of the communication-efficiency in self-stabilizing spanning-tree construction, and showed the positive and negative results.

The spanning-trees constructed by the presented protocols are not breadth-first-trees or depth-first-trees, while most of self-stabilizing (but not communication-efficient) protocols construct breadth-first-trees or depth-first-trees. By an argument similar to that in the paper, we can show that $\diamond$-$O(n)$-communication-efficiency is unattainable for constructing these restricted spanning-trees. One of our extended work will contain the characterization of spanning-trees that allow communication-efficient solutions.

Acknowledgements We are grateful to Sébastien Tixeuil and Stéphane Devismes for their helpful discussions.

References

[1] Afek, Y., Kutten, S., and Yung, M.: The local detection paradigm and its applications to self-stabilization, *Theoretical Computer Science*, Vol. 186, Issues 1–2, pp. 199–229 (1997).

[2] Aguilera, M. K., Delporte-Gallet, C., Fauconnier, H., and Toueg, S.: Stable leader election, *Proceedings of the 15th International Symposium on Distributed Computing*, pp.108–122 (2001).

[3] Aguilera, M. K., Delporte-Gallet, C., Fauconnier, H., and Toueg, S.: On implementing omega with weak reliability and synchrony assumptions, *Proceedings of the 22rd ACM Symposium on Principles of Distributed Computing*, pp.306–314 (2003).

[4] Aguilera, M. K., Delporte-Gallet, C., Fauconnier, H., and Toueg, S.: Communication-efficient leader election and consensus with limited link synchrony, *Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing*, pp.328–337 (2004).

[5] Arora, A. and Gouda, M.G.: Distributed reset, *IEEE Transactions on Computers*, Vol.43, No.9, pp.1026–1038 (1994).

[6] Biely M. and Widder J.: Optimal message-driven implementations of omega with mute processes, *ACM Transactions on Autonomous and Adaptive Systems*, Vol.4, No.1, pp.4:1–4:22 (2009).

[7] Delporte-Gallet, C., Devismes, S., and Fauconnier, H.: Robust stabilizing leader election, *Proceedings of the 9th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pp.219–233 (2007).

[8] Devismes, S., Masuzawa, T., and Tixeuil, S.: Communication efficiency in self-stabilizing silent protocols, *Proceedings of the 29th International Conference on Distributed Computing Systems*, pp.474–481 (2009).

[9] Dolev, S.: Self-stabilization, *MIT Press* (2000).

[10] Gärtner, F.C.: A survey of self-stabilizing spanning-tree construction algorithms, *Technical Report IC/2003/38, Swiss Federal Institute of Technology* (2003).

[11] Larrea, M., Fernandez, A., and Arevalo, S.: Optimal implementation of the weakest failure detector for solving consensus, *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems*, pp.52–59 (2000).