Communication-Efficient Self-stabilization in Wireless Networks*

Tomoya Takimoto, Fukuhito Ooshita,
Hirotsugu Kakugawa, and Toshimitsu Masuzawa

Graduate School of Information Science and Technology, Osaka University, Japan
{t-takimt,f-oosita,kakugawa,masuzawa}@ist.osaka-u.ac.jp

Abstract. A self-stabilizing protocol is guaranteed to eventually reach a safe (or legitimate) configuration even when started from an arbitrary configuration. Most of self-stabilizing protocols require each process to keep communicating with all of its neighbors forever even after reaching a safe configuration. Such permanent communication impairs efficiency, but is necessary in nature of self-stabilization.

The concept of communication-efficiency was introduced to reduce communication after reaching a safe configuration. The previous concept targets the point-to-point communication model, and is not appropriate to the wireless network model where a process can locally broadcast a message to its neighbors all at once.

In this paper, we refine the concept of the communication-efficiency for the wireless network model, and investigate its possibility in self-stabilization for some fundamental problems; the minimal (connected) dominating set problem, the maximal independent set problem, and the spanning tree construction problem.

1 Introduction

A *self-stabilizing* protocol [1] is guaranteed to eventually reach a safe (or legitimate) configuration even when started from an arbitrary configuration. This property enables self-stabilizing protocols to autonomously adapt to transient faults and dynamical topology changes of networks. A main concern in efficiency of self-stabilizing protocols is efficiency *in convergence* after faults, i.e., the *convergence time* required to reach a safe configuration from any configuration. The convergence time is a natural efficiency measure of self-stabilizing protocols since it is very similar to the time complexity measure of ordinary (non-self-stabilizing) protocols. However, a crucial difference in communication cost between self-stabilizing and ordinary protocols lies in the cost of communication *after convergence* to a safe configuration; self-stabilizing protocols cannot allow any process to terminate its communication even after reaching a safe configuration, while ordinary ones can eventually allow every process to terminate all the

* This work is supported in part by Grant-in-Aid for Scientific Research ((B)20300012. (B)22300009, (B)23700056, (C)24500039) of JSPS.

A.W. Richa and C. Scheideler (Eds.): SSS 2012, LNCS 7596, pp. 1–15, 2012.
© Springer-Verlag Berlin Heidelberg 2012

activity. Especially, in practical applications, communication efficiency is more important *after* convergence than *during* convergence: self-stabilizing protocols are expected to stay at safe configurations most of the time since deviations from the safe configurations caused by transient faults or topology changes occur infrequently. Nevertheless, most of self-stabilizing protocols require each process to keep communicating with all the neighbors forever.

To circumvent the inefficiency after reaching a safe configuration, the concept of *communication-efficiency* was introduced [2,3]. The concept is targeting the *point-to-point* communication model in the sense that the communication efficiency is achieved by reducing the number of process pairs that keep communicating with each other after reaching a safe configuration. Thus, the concept is not appropriate to the wireless network model where a process can locally *broadcast* a message to its neighbors all at once.

Contribution of this paper: The contribution of this paper is threefold.

1. We introduce new communication efficiency measures for the wireless network model, *k-broadcast-stability* and *k*(*-average*)*-broadcast-efficiency* [1]. Informally, the k-broadcast-stability guarantees that at most k processes keep (locally) broadcasting after reaching a safe configuration. On the other hand, the k(-average)-broadcast-efficiency guarantees that at most k processes broadcast messages every step (on average) after reaching a safe configuration (the broadcasting processes can differ at different steps). Notice that these concepts are derived from the k-stability and the k-efficiency in [2] as refinements for the wireless network model.
2. Concerning the broadcast-stability, we show the following results.
   - For the the *minimal connected dominating set* (MCDS) problem and the *spanning tree construction* (ST) problem, the $(n-1)$-broadcast-stability is *impossible* to attain, where n is the number of processes in the network. This result implies that any self-stabilizing protocol for the problems requires all the processes to keep broadcasting forever.
   - For the *minimal dominating set* (MDS) problem and the *maximal independent set* (MIS) problem, $(I_G^{min}-1)$-broadcast-stability is *impossible* to attain but I_G^{Max}-broadcast-stability is *attainable*, where I_G^{min} and I_G^{Max} are respectively the minimum and the maximum sizes of the maximal independent set of the network.

   An interesting observation from these results is that the connectivity requirement of the minimal dominating set makes an essential difference in possibility of the broadcast-stability.
3. Concerning the average-broadcast-efficiency, we show the following results.
   - For the MCDS, the MDS, the MIS and the ST problems, $o(n)$-average-broadcast-efficiency is *practically impossible* to attain if processes know no upper bound of n. More precisely, the convergence time of any $o(n)$-average-broadcast-efficient protocol cannot be bounded.

[1] These measures are also summarized in a brief survey paper [4].

– When every process knows an upper bound N of n, there exists a *universal* 1-average-broadcast-efficient protocol with the convergence time $O(N)$ for *all the static problems*. The strong result is obtained by presenting a transformer from a silent self-stabilizing protocol with a known convergence time T to a 1-average-broadcast-efficient self-stabilizing protocol with the convergence time $O(T + N)$.

Related works: Aguilera et al. [5] introduced the concept of *communication-efficiency* in implementation of failure detector Ω. Following the work, some papers investigated communication-efficiency in failure detector implementation [6,7,8,9]. The implementations in [5,6,9] can tolerate any number of crash processes and require only $n - 1$ unidirectional links to carry messages forever.

Some works [2,3] discussed communication-efficiency of self-stabilizing protocols in the point-to-point communication model. The communication-efficiency is achieved by reducing the number of process pairs that keep communicating with each other after convergence to a safe configuration. They introduced two concepts, *stability* and *efficiency*, as quantitative measures of communication-efficiency. Informally, the stability guarantees that the number of process pairs that keep communicating with each other is limited. On the other hand, the efficiency guarantees that the number of process pairs that communicate with each other is limited at every step (the pairs can differ at different steps). They investigated the stability and the efficiency for the vertex coloring problem, the MIS problem, the maximal matching problem [2], and the ST problem [3]. Another challenge, *communication adaptability*, to reduce communication complexity after reaching a safe configuration is presented in [10].

Kutten et al. [11] pointed out that reducing the communication overhead after convergence leads longer convergence time, and results in increasing the communication overhead during convergence. They presented a randomized self-stabilizing protocol that succeeds to reduce the communication overhead both during and after convergence for the ST problem.

The rest of the paper is organized as follows. Section 2 presents definitions of the wireless network model, self-stabilizing protocols, and the communication-efficiency. Sections 3 and 4 investigate possibility of the broadcast-stability, and Sections 5 and 6 investigate possibility of the broadcast-efficiency. Section 7 concludes this paper.

2 Preliminaries

2.1 System Model

We consider distributed systems with a (local) broadcast communication primitive such as wireless communication. The system model is defined as follows.

A *distributed system* is modeled by an undirected labeled graph $G = (P(G), L(G))$, where $P(G)$ is the set of n processes and $L(G)$ is the set of bidirectional communication links. Each process $v \in P(G)$ has a unique ID denoted by ID_v. A link connecting processes v and w is denoted by (v, w). We say w is a *neighbor*

of v if $(v, w) \in L(G)$, and the set of neighbors of v is denoted by $N_G(v)$ (or simply by $N(v)$ when G is apparent). We consider only connected distributed systems and denote the set of such systems by $\mathcal{G}$.

A process can communicate with its neighbors by (local) *broadcast*: when a process v broadcasts a message, each neighbor $w \in N(v)$ of v receives the message. We assume that the broadcast is reliable and all the neighbors correctly receive the message.

An important feature of wireless networks such as ad hoc ones is that a process is unaware of its neighbors until it receives messages from them. Thus, we assume each process v has no knowledge of $N(v)$ or $|N(v)|$. Also, when v receives a message, it cannot identify its sender unless the message contains the sender's ID.

A process is modeled by a state machine and a *configuration* of a distributed system G is specified by an n-tuple $c = (s_0, s_1, \ldots, s_{n-1})$, where s_i stands for the state of process v_i $(0 \leq i \leq n-1)$. We consider only a *synchronous* distributed system where all processes execute actions in a lockstep fashion. In each synchronous step, every process executes the following three operations.

1. Broadcast a message (depending on its state) to all the neighbors.
2. Receive messages from neighbors that are sent in the beginning of the step.
3. Update its state (depending on its state and the received messages).

When the configuration changes from c to c' in a step, we denote the transition by $c \mapsto c'$. *Execution* of a distributed system is an infinite sequence $E = c_0, c_1, c_2, \ldots$ satisfying $c_j \mapsto c_{j+1}$ $(j \geq 0)$, where c_0 is called the *initial configuration*. In this paper, we consider only deterministic protocols, and thus, execution starting from the initial configuration c_0 is uniquely determined.

2.2 Self-stabilizing Protocol

A *problem* is defined on the *output variables* of processes and specifies the requirement that the output variables should satisfy. A problem is called *static* when the output variables should be eventually stable with satisfying the problem requirement. All the problems considered in this paper are static.

A configuration c is called *safe* for a static problem when it satisfies the problem requirement and the output variables of all processes remain unchanged in the execution starting from c. A protocol is called *self-stabilizing* for a static problem if it eventually reaches a safe configuration even when starting from any initial configuration.

2.3 Communication Efficiency

The previous concept of communication efficiency [2,3] aims to reduce the number of communicating process pairs. But it is not adequate to a distributed system with a broadcast communication primitive, where a process can send a message to all the neighbors by a single broadcast operation. We introduce an alternative concept of communication efficiency for distributed systems with

a broadcast communication primitive. Intuitively, communication-efficiency in such a system should aim to reduce the number of broadcasts after convergence.

In the following definitions, $k : \mathcal{G} \to \mathbb{N}$ denotes a function from a distributed system to a positive integer, and $S_{A,G}(c)$ denotes the set of processes that broadcast messages in the step starting at configuration c of a protocol A .

Definition 1 ($k(G)$-broadcast-stability). *A self-stabilizing protocol A is* $k(G)$-broadcast-stable *if, for any execution $c_0, c_1, \ldots$ of A in any distributed system $G \in \mathcal{G}$, there exists i such that a suffix $c_i, c_{i+1}, \ldots$ satisfies*

$$\left|\bigcup_{j \geq i} S_{A,G}(c_j)\right| \leq k(G). \qquad \square$$

The $k(G)$-broadcast-stability guarantees that eventually at most $k(G)$ processes keep broadcasting, and thus, at least $n - k(G)$ processes eventually stop broadcasting.

Definition 2 ($k(G)$-broadcast-efficiency). *A self-stabilizing protocol A is* $k(G)$-broadcast-efficient *if, for any execution $c_0, c_1, \ldots$ of A in any distributed system $G \in \mathcal{G}$, there exists i such that a suffix $c_i, c_{i+1}, \ldots$ satisfies*

$$\forall j \geq i, \ \ |S_{A,G}(c_j)| \leq k(G). \qquad \square$$

The $k(G)$-broadcast-efficiency guarantees that eventually at most $k(G)$ processes broadcast messages in every step. Note that the broadcasting processes can differ at different steps. It is clear that $k(G)$-broadcast-stability implies $k(G)$-broadcast-efficiency, but the converse does not hold.

A relaxed variation of the $k(G)$-broadcast-efficiency is to allow at most $k(G)$ processes to broadcast messages in every step *on average*, which attains communication-efficiency practically equivalent to the $k(G)$-broadcast-efficiency. The variation is defined as follows.

Definition 3 ($k(G)$-average-broadcast-efficiency). *A self-stabilizing protocol A is* $k(G)$-average-broadcast-efficient *if, for any execution $c_0, c_1, \ldots$ of A in any distributed system $G \in \mathcal{G}$, there exists i such that a suffix $c_i, c_{i+1}, \ldots$ satisfies*

$$\lim_{j \to \infty} \frac{1}{j-i+1} \sum_{k=i}^{j} |S_{A,G}(c_k)| \leq k(G). \qquad \square$$

3 Impossibility of Broadcast-Stability

This section presents impossibility results concerning the broadcast-stability. The impossibility proofs are based on the simple observations that any process cannot become aware of changes of the processes that never broadcast messages. The details of the observations are as follows.

Observation 1: Let G be a distributed system, $E = c_0, c_1, \ldots$ be an execution of a protocol A in G, and U be a set of processes that never broadcast messages in E. Consider any distributed system G' obtained from G by removing some processes in U, removing some links between remaining processes in

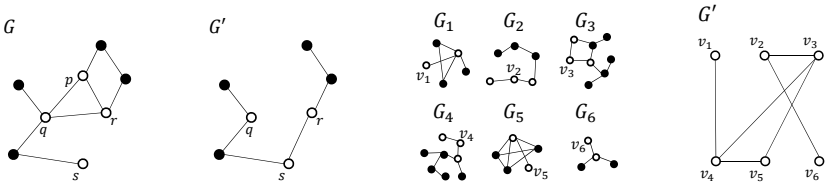

Fig. 1. Construction of G' in Observations. (a) G and $U = \{p, q, r, s\}$ in Observation 1. (b) An example of G' in Observation 1. (c) $G_1, G_2, \ldots, G_6$ in Observation 2. (d) An example of G' in Observation 2.

U, and adding some links between remaining processes in U (Fig. 1(a),(b)). (When G' obtained by the above modification is disconnected, any of its connected components is considered as G'.) Then, $E' = c'_0, c'_1, \ldots$ is an execution of A in G' such that the state of every process at c'_i is the same as that at c_i for every i ($i \geq 0$).

Observation 2: Let $G_1, G_2, \ldots, G_n$ be distributed systems such that the process sets are mutually disjoint, $E_x = c_{x,0}, c_{x,1}, \ldots$ be an execution of a protocol A in G_x, and v_x be a process (if exists) that never broadcasts or receives messages in E_x ($1 \leq x \leq n$). Consider any distributed system G' consisting of processes $\{v_1, v_2, \ldots, v_n\}$ and arbitrarily added links (Fig. 1(c),(d)). Then, $E' = c'_0, c'_1, \ldots$ is an execution of A in G' such that the state of every process v_x at c'_i is the same as that at $c_{x,i}$ for every x ($1 \leq x \leq n$) and i ($i \geq 0$).

3.1 Spanning Tree Construction

The *spanning tree construction (ST) problem* requires each process to select a neighbor as its parent so that the parent relations of all processes form a spanning tree of the distributed system. Each process has an output variable *prnt* to store the ID of its parent. In the process selected as the root of the spanning tree, *prnt* stores its own ID.

The broadcast-stability is impossible to attain for the ST problem:

Theorem 1. *Let $k(G)$ be any function satisfying $k(G) < n(= |P(G)|)$ for any distributed system $G \in \mathcal{G}$. There is no $k(G)$-broadcast-stable self-stabilizing protocol for the spanning tree construction problem.*

Proof. Assume, for contradiction, that there exists a $k(G)$-broadcast-stable self-stabilizing protocol A for the ST problem. Then, any execution of A in G has a suffix $E = c_i, c_{i+1}, \ldots$ in which a spanning tree is constructed (and remains unchanged) and at least one process never broadcasts messages.

(a) Case that there exists a non-leaf process, say v, of the spanning tree that never broadcasts messages in E: Let G' be the distributed system obtained from G by removing v, and $E' = c'_i, c'_{i+1}, \ldots$ be the execution of A in G' described

in Observation 1. In E', the output variable *prnt* of every process remains unchanged. However, the variables of all the processes do not form a spanning tree, since there is a neighbor $w \in N_G(v)$ of v that selects v as its parent but v does not exist in G'. This implies that protocol A cannot reach a safe configuration in E', which is a contradiction.

(b) Case that only leaf processes of the spanning tree never broadcast messages in E: Let G' be the distributed system obtained from G by removing all the processes that never broadcast in E, and $E' = c'_i, c'_{i+1}, \ldots$ be the execution of A in G' described in Observation 1. No process stops broadcasting in E', which contradicts $k(G')$-broadcast-stability for $k(G') < |P(G')|$. □

3.2 Minimal Connected Dominating Set

A *dominating set* of G is a subset $D \subseteq P(G)$ of processes such that each process $v \in P(G) - D$ has a neighbor in D. If the subgraph $G[D]$ of G induced by a dominating set D is connected, D is a *connected* dominating set of G. If no proper subset of (connected) dominating set D is a (connected) dominating set, D is a *minimal* (connected) dominating set.

The *minimal connected dominating set (MCDS) problem* requires us to choose processes so that they should form a MCDS of the distributed system. Each process has a boolean output variable *mcds* and stores `true` when it is a member of the MCDS.

Theorem 2. *Let $k(G)$ be any function satisfying $k(G) < n(= |P(G)|)$ for any distributed system $G \in \mathcal{G}$. There is no $k(G)$-broadcast-stable self-stabilizing protocol for the minimal connected dominating set problem.*

Proof. We can prove this theorem by similar argument to the proof of Theorem 1, with restricting our attention to distributed systems of line topology ($n \geq 6$). Notice that the MCDS of such a system is uniquely determined as the set of all the processes except for the two end processes. □

3.3 Minimal Dominating Set and Maximal Independent Set Problems

For the minimal dominating set (MDS) problem, we can prove the following impossibility result, which is weaker than Theorem 2 for the MCDS problem. Actually, we can present a broadcast-stable self-stabilizing protocol for the MDS problem in Section 4. These results show that the connectivity requirement of the minimal dominating set brings an essential difference in possibility of the broadcast-stability.

Theorem 3. *Let $k(G)$ be any function satisfying $k(G) < I_G^{min}$ for any distributed system $G \in \mathcal{G}$ where I_G^{min} is the minimum size of the maximal independent set of G. There is no $k(G)$-broadcast-stable self-stabilizing protocol for the minimal dominating set problem.*

Proof. We restrict our attention to distributed systems of line topology, and assume, for contradiction, that there exists a $k(G)$-broadcast-stable self-stabilizing protocol A for the MDS problem.

Let $\mathcal{P}_n$ be the set of n-process distributed systems of line topology. Since $I_G^{min} = \lceil n/3 \rceil$ holds for any $G \in \mathcal{P}_n$, at least $n - \lceil n/3 \rceil + 1$ processes eventually stop broadcasting in any execution of A. In case of $n = 3\ell$ for a positive integer ℓ, this implies that there exist, in any execution of A, three consecutive processes that eventually stop broadcasting. Notice that the process in the middle of the three never broadcasts or receives messages after some configuration in the execution. Thus, Observation 2 allows us to construct a distributed system of line topology by choosing such a process from each of some distributed systems.

More concretely, we consider n distributed systems $P_1, P_2, \ldots, P_n$ ($P_i \in \mathcal{P}_n$) such that their ID sets are mutually disjoint. We choose, from each P_i, a process v_i that never broadcasts or receives messages after some configuration in an execution. Following Observation 2, we can arbitrarily add links, and thus, we can construct a line that does not satisfy the MDS specification. This is a contradiction. □

An *independent set* of G is a subset $I \subseteq P(G)$ such that no processes in I are neighboring. If no proper superset of an independent set I is an independent set, I is a *maximal* independent set (MIS). The following theorem on impossibility of the MIS problem is derived from Theorem 3 since any MIS is a MDS.

Theorem 4. *Let $k(G)$ be any function satisfying $k(G) < I_G^{min}$ for any distributed system $G \in \mathcal{G}$. There is no $k(G)$-broadcast-stable self-stabilizing protocol for the maximal independent set problem.* □

4 Broadcast-Stable Self-stabilizing Protocol

Theorems 1 and 2 imply that all processes have to keep broadcasting forever in any self-stabilizing protocol for the ST and the MCDS problems. Thus the broadcast-stability is impossible to attain for these problems. In this section, we show possibility of the broadcast-stability for the other two problems, the MDS and the MIS problems.

A broadcast-stable self-stabilizing protocol, *stable-MIS*, for the MIS problem is presented in Protocol 1. Each process v has a boolean output variable ind_v to denote whether v is a member of the constructed MIS: v is a member of the MIS iff $ind_v = \texttt{true}$.

The main idea to achieve the broadcast-stability is that only the processes in the constructed MIS are allowed to keep broadcasting. The assumption of synchronous distributed systems guarantees that a process v has a neighbor in the MIS if and only if v receives a message. To break symmetry, the message contains the sender's ID and a process with a larger ID has higher priority to become a member of the MIS.

The following two lemmas obviously hold.

Protocol 1. *stable-MIS* : actions of process v in each step.

Output Variables:
1: ind_v : boolean;
Internal Variables:
2: M_v : set of messages;
Actions:
3: **if** $ind_v = \texttt{true}$ **then**
4: Broadcast($\{ID_v\}$);
5: **end if**
6: $M_v \leftarrow$ received messages;
7: **if** $\exists m \in M_v[m.ID > ID_v]$ **then**
8: $ind_v \leftarrow \texttt{false}$;
9: **else**
10: $ind_v \leftarrow \texttt{true}$;
11: **end if**

Lemma 1 (Safe configuration). *A configuration c of protocol stable-MIS is safe if c satisfies the following conditions:*

1. $\forall v \in P(G),\ [(ind_v = \texttt{true}) \Rightarrow (\forall w \in N(v),\ ind_w = \texttt{false})]$.
2. $\forall v \in P(G),\ [(ind_v = \texttt{false}) \Rightarrow \exists w \in N(v)\ (ID_w > ID_v \wedge ind_w = \texttt{true})]$.

□

Lemma 2 (Convergence). *Starting from any initial configuration, protocol stable-MIS reaches a safe configuration in at most n steps.* □

The following theorem is derived from Lemmas 1, 2, and the fact that only processes in the constructed MIS keep broadcasting. The theorem also holds for the MDS, since protocol *stable-MIS* is also a solution to the MDS problem (Recall any MIS is a MDS). In the theorem, I_G^{Max} denotes the size of the *maximum* independent set of $G \in \mathcal{G}$.

Theorem 5. *Protocol stable-MIS is a I_G^{Max}-broadcast-stable self-stabilizing protocol for the maximal independent set problem and the minimal dominating set problem. It reaches a safe configuration in at most n steps in any distributed system $G \in \mathcal{G}$.* □

5 Impossibility of Broadcast-Efficiency

The average-broadcast-efficiency can be improved by prohibiting each process from broadcasting a message *every step* and, instead, by allowing each process to broadcast a message only *every k steps* for some positive integer k. This reduces the average-broadcast-efficiency by a factor of k (e.g., from n to n/k), but may increase the convergence time by a factor of k. Actually, we can show a negative result when no upper bound of the number of processes n is available at any process: $o(n)$-average-broadcast-efficiency is *practically* impossible to attain

in the sense that any protocol with such efficiency has no upper bound (even depending on n) on the convergence time.

The negative result is obtained by the argument based on the following observation, which is similar to Observation 2. Observation 2 was used to show existence of an infinite execution in which no process broadcasts messages. On the other hand, the following observation is used to show existence of a sufficiently (but finite) long fragment of execution in which no process broadcasts messages.

Observation 3: Let $G_1, G_2, \ldots, G_n$ be distributed systems such that the process sets are mutually disjoint, $E_x = c_{x,0}, c_{x,1}, \ldots$ be an execution of a protocol A in G_x, and v_x be a process (if exists) that never broadcasts or receives messages in an execution prefix $E_x^t = c_{x,0}, c_{x,1}, \ldots, c_{x,t}$ $(1 \leq x \leq n)$.
Consider any distributed system G' consisting of n processes $\{v_1, v_2, \ldots, v_n\}$ and arbitrarily added links. Then, $E' = c'_0, c'_1, \ldots, c'_t$ is an execution prefix of A in G' such that the state of every process v_x at c'_i is the same as that at $c_{x,i}$ for every x $(1 \leq x \leq n)$ and i $(0 \leq i \leq t)$.

The following lemma holds for distributed systems of line topology.

Lemma 3. *Consider any execution E of any $k(G)$-average-broadcast-efficient protocol A in a distributed system $G \in \mathcal{P}_m$ for any $m > 3$. There exist three consecutive processes and an execution fragment composed of $\lfloor \frac{m-3}{3k(G)} \rfloor$ steps such that the three processes never broadcast messages during the execution fragment.* □

The following theorem can be obtained by argument similar to the proof of Theorem 3.

Theorem 6. *Assume no upper bound of the number of processes n is available at any process, and consider any $o(n)$-average-broadcast-efficient protocol A for the maximal independent set problem, the minimal dominating set problem, the minimal connected dominating set problem, or the spanning tree construction problem. Then, the convergence time of A cannot be bounded by any T (which may depend on n).*

Proof. We restrict our attention to distributed systems of line topology, and assume, for contradiction, that there exists a $k(G)$-average-broadcast-efficient self-stabilizing protocol A for the problem, where $k(G) = o(n)$, with the convergence time bounded by T (which may depend on n).

Let $G_1, G_2, \ldots, G_n$ $(G_i \in \mathcal{P}_m)$ be n distributed systems of m-process line topology such that the process sets are mutually disjoint. Each G_i contains a process, say v_i, that is the middle of the three consecutive processes in Lemma 3. Note that v_i never broadcasts or receives messages during the execution fragment of length $\lfloor \frac{m-3}{3k(G_i)} \rfloor$ in G_i. Observation 3 allows us to construct a distributed system G of line topology consisting of $v_1, v_2, \ldots, v_n$ so that its initial configuration is not safe and no process broadcasts messages in the first $\lfloor \frac{m-3}{3k(G_j)} \rfloor$ steps, where

$k(G_j) = \max_{1 \leq i \leq n} k(G_i)$. By setting m to satisfy $\lfloor \frac{m-3}{3k(G_j)} \rfloor > T$ ($k(G_j) = o(m)$ guarantees existence of such m), we can show that the convergence time of A is longer than T, which is a contradiction. □

6 Broadcast-Efficient Self-stabilizing Protocol

In Section 5, we showed that the $o(n)$-average-broadcast-efficiency is impossible to attain without greatly sacrificing the convergence time if no upper bound of n is available. However, the situation drastically changes if an upper bound N of n is available. In this section, we show that knowledge of N allows the 1-average-broadcast-efficiency without paying high penalty on the convergence time: we present a method for transforming any silent self-stabilizing protocol with a known (upper bound of) convergence time T to a 1-average-broadcast-efficient self-stabilizing protocol with the convergence time $O(T + N)$.

We assume that an upper bound N of n is known to each process. We also assume that a self-stabilizing protocol A given as an *input* of the transformation satisfies the following conditions (called *input conditions*).

(a) A is *silent*, that is, no process changes its state after reaching a safe configuration.
(b) In any execution, every process broadcasts a message every step.
(c) In any unsafe configuration c, there exists at least one process that changes its state in the step from c.
(d) An upper bound T of the convergence time is known.

Protocol 2 presents the transformation method by showing the protocol *Eff(A)* resulting from the input protocol A. The main idea for attaining the 1-average-broadcast-efficiency is to allow each process to broadcast a message *only every N steps* after reaching a safe configuration of A. It is obvious that such *slowdown* in activity brings the 1-average-broadcast-efficiency to A. However it is impossible for a self-stabilizing protocol to correctly detect a safe configuration. Especially, the fault positive in the detection may increase the convergence time of A from T to NT: when processes prematurely slow down before reaching a safe configuration, the convergence time may become N times longer.

The idea for overcoming this difficulty comes from the following observations.

1. Even when every process prematurely detects a safe configuration and restricts it to broadcast a message only every N steps, each process can acquire the states of all the neighbors in N steps. Thus, from the input condition (c), there exists a process that changes its state in N steps. This implies that the premature slowdown can be detected in N steps.
2. When a process detects the premature slowdown, it can signal, in n steps, all processes to stop the slowdown and come back to *ordinary execution* of A that allows all processes to broadcast messages every step.
3. Once all processes execute ordinary execution of A during T steps, the protocol reaches and remains at a safe configuration.

To realize a method inspired from the above observations, we use a *local clock* of each process and *clock synchronization*. Each process v has variable $clock_v : \{0, \ldots, T + 2N\}$ as a local clock, and uses it as follows.

1. The local clocks of all the processes become synchronized by executing
$$clock_v \leftarrow \min\{\min\{clock_w | w \in N_G(v) \cup \{v\}\} + 1, T + 2N\}.$$
The intuition is that processes try to adjust their clocks to that with the lowest clock value, and the clock value is bounded by $T + 2N$. As long as $clock_v$ is synchronized with those of the neighbors, the clock value increases by one every step until it reaches $T + 2N$.
2. Any clock value less than $T + 2N$ suggests that a safe configuration is not reached yet (though the suggestion may not be correct). Thus, it makes process v execute ordinary execution of protocol A.
3. Clock value $T+2N$ suggests that the configuration is safe (though the suggestion may not be correct). Thus, it makes process v slowdown and broadcast a message only every N steps.
Even when $clock_v = T+2N$ holds, process v executes actions of A on receipt of messages. If the configuration is safe, v never changes its state. Thus, a change of v's state implies detection of an unsafe configuration. In this case, v also executes $clock_v \leftarrow 0$ to come back to ordinary execution of A. The clock synchronization mechanism brings, in n steps, all processes back to ordinary execution of A. We call the process v that executes $clock_v \leftarrow 0$ an *initiator*.

As we show later, if a safe configuration is not reached sufficiently long time, the clocks of all processes become synchronized before the clock value of any process reaches $2N$. Once the clocks become synchronized, all processes execute ordinary execution of A. The ordinary execution lasts during at least T steps (until the clock value reaches $T+2N$), which guarantees that protocol A reaches a safe configuration.

In Protocol 2, each process v has variable $state_v$ to store its current state of A, and executes protocol A using the variable. Each process also has variable $cache_v$ to store the set of messages: for each neighbor $w \in N(v)$, the latest message from w is stored in $cache_v$. Process v uses the stored messages to execute actions of A, when v receives no message from a neighbor, say w. This situation occurs when w slows down. Notice that $cache_v$ may store false messages from non-neighbors (that may not exist in the system) in the initial configuration. To eliminate such false messages, v removes the stored message from w if v receives no message during $N + 1$ steps (for simplicity, this action is omitted in Protocol 2).

Lemma 4 (Safe configuration). *A configuration of Eff(A) is safe if the following conditions are satisfied.*

- $\forall v \in P(G)$, $clock_v = T + 2N$.
- $\forall v \in P(G)$, $cache_v$ *stores the latest message from each neighbor* $w \in N(v)$.
- *Letting* c' *be an* n*-tuple* $(state_{v_0}, \ldots, state_{v_{n-1}})$ *where* $\{v_0, \ldots, v_{n-1}\} = P(G)$, c' *is a safe configuration of* A. □

Protocol 2. *Eff(A)* : Actions of process v in each step.

Constants:
1: N : upper bound on n
2: T : upper bound on the convergence time of the input protocol A
Variables:
3: $state_v$: a state of v in protocol A
4: $cache_v$: a set of messages received from neighbors
5: $clock_v$: $\{0, ..., T + 2N\}$
6: $count_v$: $\{0, ..., N - 1\}$
Actions:
7: **if** $clock_v < T + 2N$ **then** // process v executes ordinary execution of A
8: Broadcast$((ID_v, msg_v, clock_v))$; //msg_v is the message sent by A
9: $M_v \leftarrow$ received messages;
10: Update $cache_v$ using M_v to store the latest message from each neighbor;
11: Update $state_v$ using $cache_v$ by actions of A;
12: $clock_v \leftarrow \min(\{m.clock | m \in M_v\} \cup \{clock_v\}) + 1$;
13: $count_v \leftarrow 0$;
14: **else** // process v executes slowdown execution of A
15: **if** $count_v = 0$ **then**
16: Broadcast$((ID_v, msg_v, clock_v))$; //msg_v is the message sent by A
17: **end if**
18: $M_v \leftarrow$ received messages;
19: Update $cache_v$ using M_v to store the latest message from each neighbor;
20: Update $state_v$ using $cache_v$ by actions of A;
21: **if** $state_v$ changed **then** // an unsafe configuration is detected
22: $clock_v \leftarrow 0$;
23: **else if** $\exists m \in M_v[m.clock < T + 2N - 1]$ **then**
24: $clock_v \leftarrow \min\{m.clock | m \in M_v\} + 1$;
25: **end if**
26: $count_v \leftarrow (count_v + 1) \bmod N$;
27: **end if**

Lemma 5 (Convergence). *Starting from any initial configuration, protocol Eff(A) reaches a safe configuration within* $2T + 4N$ *steps.*

Proof. We consider any execution $E = c_0, c_1, \ldots$ of *Eff(A)*.

(a) For the case that no initiator appears by c_{T+N} (in the first $T + N$ steps), we show that *Eff(A)* reaches a safe configuration by c_{T+2N}: It is clear that the clocks of all processes become synchronized by c_{n-1} (although they may already reach $T + 2N$). Consider the following T steps (from c_{n-1} to c_{T+n-1}). If the clock value does not reach $T + 2N$ by c_{T+n-1}, all processes execute ordinary execution of A during the T steps, and thus, $(state_{v_0}, \ldots, state_{v_{n-1}})$ reaches a safe configuration of A by c_{T+n-1} and remains unchanged after. Then, the clock value reaches $T + 2N$ by c_{T+2N}, and thus, *Eff(A)* reaches a safe configuration by c_{T+2N}. On the other hand, consider the case that the clock value reaches $T + 2N$ by c_{T+n-1}. Let c_g $(g \leq T + n - 1)$ be the configuration at which the clock value reaches $T + 2N$. If c_g is unsafe, there appears an initiator in the next

step. Thus, c_g is safe, since we are considering the case that no initiator appears by c_{T+N}. Consequently, *Eff(A)* reaches a safe configuration by c_{T+2N}.

(b) For the case that an initiator appears by c_{T+N}, we show that *Eff(A)* reaches a safe configuration by c_{2T+4N}: Let c_f $(f \leq T+N)$ be the configuration such that the first initiator, say v, appears in the step from c_{f-1} to c_f. Since $clock_v \leftarrow 0$ is executed in the step, the clock synchronization mechanism spreads the effect to all the processes in the following $n-1$ steps. Thus, the values of all clocks are no greater than $n-1$ at c_{f+n-1}, whether other initiators appear in the period from c_f to c_{f+n-1} or not. This implies that the clocks of all processes become synchronized by c_{f+2n-2} since no initiator appears in the period from c_{f+n-1} to c_{f+2n-2}. Since the clock value at c_{f+2n-2} is no greater than $2n$, all processes execute ordinary execution during T steps from c_{f+2n-2} to $c_{T+f+2n-2}$, and $(state_{v_0}, \ldots, state_{v_{n-1}})$ becomes a safe configuration of A by $c_{T+f+2n-2}$. The clock value reaches $T+2N$ by $c_{T+2N+f+n-1}$. Consequently, *Eff(A)* reaches a safe configuration by c_{2T+4N}. □

It is clear that Protocol *Eff(A)* is 1-average-broadcast-efficient. Thus, the following theorem is obtained from Lemmas 4 and 5.

Theorem 7. *Protocol Eff(A) is a self-stabilizing protocol for the problem that the input self-stabilizing protocol A is targeting. It is also 1-average-broadcast-efficient and its convergence time is $O(T+N)$, where T is the convergence time of A and N is a known upper bound of the number of processes n.* □

Many self-stabilizing protocols proposed so far satisfy the input conditions: for example, self-stabilizing protocols for the MIS problem (also for the MDS problem) with the convergence time $O(n)$ [12], and for the ST problem with the convergence time $O(N)$ [13]. Application of the transformer to these protocols provides 1-average-broadcast-efficient self-stabilizing protocols with the convergence time $O(N)$ for these problems. More interestingly, we can obtain a much more general result: the 1-average-broadcast-efficiency is attainable for *any static problem* without paying high penalty on the convergence time.

Theorem 8. *When an upper bound N of the number of processes n is available at every process, there exists a 1-average-broadcast-efficient self-stabilizing protocol with the convergence time $O(N)$ for* any static problem. □

To prove Theorem 8, we consider a *universal static problem Π* on any distributed system G: each process has an *input*, and is required to compute an *output* that is (deterministically) computable from the *complete information of G* (i.e., a graph G where each node v is labeled with ID_v and the input of v).

The core of a self-stabilizing protocol for Π is to construct the complete information of G at each process, and it can be achieved by letting all processes know the tuple $(ID_v, NID_v, Input_v)$ of every process v, where NID_v is the set of IDs of v's neighbors and $Input_v$ is the input given at v. There exists a self-stabilizing protocol for constructing the complete information of G at each process with the convergence time $O(N)$, and it satisfies the input conditions of the transformer. Thus, we can obtain Theorem 8.

7 Conclusion

In this paper, we introduced and investigated communication-efficiency of self-stabilizing protocols in the distributed system model with the (local) broadcast primitive. A wireless network is a typical example of the system model, and the communication-efficiency may play an important role in reducing energy-consumption, which is a critical issue in the wireless networks. As the first step, we considered only the *reliable* broadcast primitive. However, to consider a *lossy* broadcast primitive is practically important and is one of our future works.

Another future work is to introduce the *sleep* mode in which a process stop broadcasting and receiving messages to save energy consumption, while we assume, in this paper, a process is allowed to receive messages all the time.

References

1. Dolev, S.: Self-stabilization. The MIT press (2000)
2. Devismes, S., Masuzawa, T., Tixeuil, S.: Communication efficiency in self-stabilizing silent protocols. In: Proc. of IEEE ICDCS, pp. 474–481 (2009)
3. Masuzawa, T., Izumi, T., Katayama, Y., Wada, K.: Brief Announcement: Communication-Efficient Self-stabilizing Protocols for Spanning-Tree Construction. In: Abdelzaher, T., Raynal, M., Santoro, N. (eds.) OPODIS 2009. LNCS, vol. 5923, pp. 219–224. Springer, Heidelberg (2009)
4. Masuzawa, T.: Silence Is Golden: Self-stabilizing Protocols Communication-Efficient after Convergence. In: Défago, X., Petit, F., Villain, V. (eds.) SSS 2011. LNCS, vol. 6976, pp. 1–3. Springer, Heidelberg (2011)
5. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Stable Leader Election. In: Welch, J.L. (ed.) DISC 2001. LNCS, vol. 2180, pp. 108–122. Springer, Heidelberg (2001)
6. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: On implementing omega with weak reliability and synchrony assumptions. In: Proc. of PODC, pp. 306–314 (2003)
7. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Communication-efficient leader election and consensus with limited link synchrony. In: Proc. of PODC, pp. 328–337 (2004)
8. Biely, M., Widder, J.: Optimal message-driven implementations of omega with mute processes. ACM TAAS 4(1), 4:1–4:22 (2009)
9. Larrea, M., Fernández, A., Arévalo, S.: Optimal implementation of the weakest failure detector for solving consensus. In: Proc. of SRDS, pp. 52–59 (2000)
10. Dolev, S., Schiller, E.: Communication adaptive self-stabilizing group membership service. IEEE TPDS 14(7), 709–720 (2003)
11. Kutten, S., Zinenko, D.: Low Communication Self-stabilization through Randomization. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 465–479. Springer, Heidelberg (2010)
12. Ikeda, M., Kamei, S., Kakugawa, H.: A space-optimal self-stabilizing algorithm for the maximal independent set problem. In: Proc. of PDCAT, pp. 70–74 (2002)
13. Arora, A., Gouda, M.G.: Distributed Reset. IEEE Trans. Comp. 43(9), 1026–1038 (1994)