Communication Efficiency in Self-Stabilizing Silent Protocols

Stéphane Devismes
VERIMAG UMR 5104
Université Joseph Fourier
Grenoble, France
stephane.devismes@imag.fr

Toshimitsu Masuzawa
Graduate School of Information Science and Technology
Osaka University
Osaka, Japan
masuzawa@ist.osaka-u.ac.jp

Sébastien Tixeuil
LIP6 UMR 7606
Université Pierre et Marie Curie
Paris, France
sebastien.tixeuil@lip6.fr

Abstract

In this paper, our focus is to lower the communication complexity of self-stabilizing protocols below *the need of checking every neighbor forever. Our contribution is threefold:* (i) *We provide new complexity measures for communication efficiency of self-stabilizing protocols, especially in the stabilized phase or when there are no faults,* (ii) *On the negative side, we show that for non-trivial problems such as coloring, maximal matching, and maximal independent set, it is impossible to get (deterministic or probabilistic) self-stabilizing solutions where* every *participant communicates with less than every neighbor in the stabilized phase, and* (iii) *On the positive side, we present protocols for maximal matching and maximal independent set such that a fraction of the participants communicates with* exactly one *neighbor in the stabilized phase.*

1. Introduction

Self-stabilization [1] is a general paradigm to provide forward recovery capabilities to distributed systems and networks. Intuitively, a protocol is self-stabilizing if it is able to recover without external intervention from any catastrophic transient failure. Among the many self-stabilizing solutions available today [2], the most useful ones for real networks are those that admit efficient implementations.

Most of the literature is dedicated to improving efficiency after failures occur, *i.e.*, minimizing the stabilization time — the maximum amount of time one has to wait before failure recovery. While this metric is meaningful to evaluate the efficiency in the presence of failures, it fails at capturing the overhead of self-stabilization when there are no faults, or after stabilization. In order to take forward recovery actions in case of failures, a self-stabilizing protocol has to gather information from other nodes in order to detect inconsistencies. Of course, a *global* communication mechanism will lead to a large coverage of anomaly detection [3] at the expense of an extremely expensive solution when there are no faults, since information about every participant has to be repetitively sent to every other *participant*. As pointed out in [4], the amount of information that has to be gathered highly depends on the task to be solved if only the output of the protocol is to be used for such anomaly detection. The paper also points out that more efficient schemes could be available for some particular implementations. However, to the best of our knowledge, the minimal amount of communicated information in self-stabilizing systems is still *fully local* [4]–[6]: when there are no faults, every participant has to communicate with every other *neighbor* repetitively.

In this paper, our focus is to lower the communication complexity of self-stabilizing protocols *below* the need of checking every neighbor. A quick observation shows that non-existent communication is impossible in the context of self-stabilization: the initial configuration of the network could be such that the specification is violated while no participant is sending nor getting neighboring information, resulting in a deadlock. On the other side, there exist problems (such as coloring, maximal matching, maximal independent set) that admit solutions where participants only have to communicate with their full set of neighbors. We investigate the possibility of intermediate solutions (*i.e.* where participants communicate repetitively only with a *strict subset* of their neighbors) that would lead to more efficient implementations in stabilized phase or when there are no faults. Good candidates for admitting such interesting complexity solutions are *silent* protocols [7]: a silent protocol is a self-stabilizing protocol that exhibits the additional property that after stabilization, communication is fixed between every pair of neighbors (that is, neighbors repetitively communicate the same information forever). We thus concentrate on lowering communication complexity requirements for silent self-stabilizing protocols.

In more details, the contribution of the paper is threefold:

1) We provide new complexity measures for communication efficiency of self-stabilizing protocols, especially in the stabilized phase or when there are no faults. Our notion of communication efficiency differs from the one introduced in [8] (that was subsequently used for fault-tolerant non-self-stabilizing systems [8]–[10] and then extented to fault-tolerant self-stabilizing systems

This work was supported by ANR SHAMAN and EGIDE SAKURA projects. This work was also supported in part by MEXT: Global COE Program and JSPS: Grant-in-Aid for Scientific Research ((B)19300017).

– *a.k.a. ftss* – [11]). The essential difference is that the efficiency criterion of [8] is *global* (eventually only $n-1$ communication channels are used) while our notion is local (eventually processes only communicate with a strict subset of their neighbors). As noted in [8]–[11], global communication efficiency often leads to solutions where one process needs to periodically send messages to every other process. In contrast, with our notion, the communication load is entirely distributed and balanced.

2) On the negative side, we show two impossibility results holding for a wide class of problems. This class includes many classical distributed problems, *e.g.*, coloring, maximal matching, and maximal independent set. We first show that there is no (deterministic or probabilistic) self-stabilizing solutions for such problems in arbitrary anonymous networks where *every* participant has to communicate with a strict subset of its neighbors once the system is stabilized. We then show that it is even more difficult to self-stabilize these problems if the communication constraint must always hold. Indeed, even with symmetry-breaking mechanisms such as a leader or acyclic orientation of the network, those tasks remain impossible to solve.
3) On the positive side, we present two protocols such that a fraction of the participants communicates with *exactly one* neighbor in the stabilized phase.

The remaining of the paper is organized as follows. Section 2 presents the computational model we use in this paper. We introduce in Section 3 new complexity measures for communication efficiency. Sections 4 and 5 describe our negative and positive results. Section 6 provides some concluding remarks and open questions.

2. Model

A *distributed system* is a set Π of n communicating state machines called *processes*. Each process p can directly communicate using *bidirectional* media with a restricted subset of processes called *neighbors*. We denote by $\Gamma.p$ the set of p's neighbors and by $\delta.p$ the degree of p. We consider here distributed systems having an *arbitrary connected topology*, modelized by an undirected connected graph $G = (\Pi, E)$ where E is a set of m edges representing the bidirectional media between neighboring processes. In the sequel, Δ denotes the degree of G and D its diameter.

We assume that each process p can distinguish any two neighbors using *local indices*, that are numbered from 1 to $\delta.p$. We indifferently use the *label* q to designate the process q or the local index of q in the code of some process p. We will often use the *anonymous* assumption which states that the processes may only differ by their degrees.

Communications are carried out using a finite number of *communication variables* that are maintained by each process. Communication variables maintained by process p can be read and written by p, but only read by p's neighbors. Each process p also maintains a finite set of *internal variables* that may only be accessed by p. Each variable ranges over a fixed domain. We use uppercase letters to denote communication variables and lowercase ones to denote internal variables. Some variables can be *constant*. We refer to a variable v of the process p as $v.p$. The *state* of a process is defined by the values of its variables. A *configuration* is an instance of the states of all processes. The *communication state* of a process is its state restricted to its communication variables. A *communication configuration* is an instance of the communication states of all processes.

A *protocol* is a collection of n sequential *local algorithms*, each process executing one local algorithm. A process updates its state by executing its local algorithm. A local algorithm consists of a finite set of guarded actions of the form $\langle guard \rangle \rightarrow \langle action \rangle$. A guard is a Boolean predicate over the variables of the process and the communication variables of its neighbors. An *action* is a sequence of statements assigning new values to its variables. An action can be executed only if its guard is *true*. We assume that the execution of any action is *atomic*. An action is said *enabled* in some configuration if its guard is *true* in this configuration. By extention, we say that a process is enabled if at least one of its actions is enabled.

A *computation* C is an infinite sequence $(\gamma_0 s_0 \gamma_1)$, $\ldots (\gamma_i s_i \gamma_{i+1})$, $\ldots$ such that for any $i \geq 0$: *(i)* γ_i is a configuration, *(ii)* s_i is a non-empty subset of processes chosen according to a *scheduler*, and *(iii)* each configuration γ_{i+1} is obtained after all processes in s_i execute from γ_i one of their enabled actions, if any.[1] Any triplet $(\gamma_i s_i \gamma_{i+1})$ is called a *step*. Any finite sequence of consecutive steps of C starting from γ_0 is a *prefix* of C. A *suffix* of C is any computation obtained by removing a finite sequence $(\gamma_0 s_0 \gamma_1), \ldots, (\gamma_k s_k \gamma_{k+1})$ from C. The suffix associated to the prefix $(\gamma_0 s_0 \gamma_1) \ldots, (\gamma_{i-1} s_{i-1} \gamma_i)$ is the suffix of C starting from γ_i. A configuration γ' is said *reachable* from the configuration γ if and only if there exists a computation starting from γ that contains the configuration γ'.

A *scheduler* is a predicate that determines which are the possible computations. We assume here a *distributed fair scheduler*. *Distributed* means that any non-empty subset of processes can be chosen in each step to execute an action. *Fair* means that every process is selected infinitely many times to execute an action. We assume priority on the guarded actions that are induced by the order of appearance of the actions in the code of the protocols. Actions appearing first have higher priority than those appearing last.

To evaluate the time complexity, we use the notion of *round* [12]. This notion captures the execution rate of the slowest process in any computation. The first *round* of an

1. If all processes in s_i are disabled in γ_i, then $\gamma_{i+1} = \gamma_i$

computation C, noted C', is the minimal prefix of C where every process has been activated by the scheduler. Let C'' be the suffix associated to C'. The second *round* of C is the first round of C'', and so on.

A configuration *conforms* to a predicate if this predicate is satisfied in this configuration; otherwise the configuration *violates* the predicate. By this definition every configuration conforms to the predicate *true* and none conforms to the predicate *false*. Let R and S be predicates on configurations. Predicate R is *closed* with respect to the protocol actions if every configuration of any computation that starts in a configuration conforming to R also conforms to R. Predicate R *converges* to S if R and S are closed and every computation starting from a configuration conforming to R contains a configuration conforming to S.

Self-Stabilization [13] can be defined as follows: *A protocol stabilizes to a predicate R if and only if true converges to R.* In any protocol that stabilizes to the predicate R, any configuration that conforms to R is said *legitimate*. Conversely, any configuration that violates R is said *illegitimate*.

All protocols presented in this paper are *silent* [7]: *A protocol is silent if and only if starting from any configuration, it converges to a configuration after which the values of its communication variables are fixed.* We call *silent configuration* any configuration from which the values of all communication variables are fixed.

3. Measures for Communication Efficiency

3.1. k-efficiency

We are interested in designing self-stabilizing protocols where processes do not communicate with all their neighbors during each step. The k-*efficiency* defined below allows to compare protocols following this criterion.

Definition 1 (k-efficient) *A protocol is said to be* k*-efficient if in every step of its possible computations, every process reads communication variables of at most* k *neighbors.*

3.2. Space Complexity

To compare the space complexity of distributed algorithms, we distinguish two complexity criteria.

Definition 2 (Communication Complexity) *The* communication complexity *of a process p is the maximal amount of memory p reads from its neighbors in any given step.*

Definition 3 (Space complexity) *The* space complexity *of a process p is the sum of the local memory space (that is, the space needed for communication and internal variables) and the communication complexity of p.*

3.3. Communication Stability

In our protocols, some processes may read the communication variables of every neighbor forever, while other processes may eventually read the communication variable of a single neighbor. We emphasize this behavior by introducing the k-*stability* and two weakened forms: the $\Diamond$-k-*stability* and the $\Diamond$-$(\mathtt{x},\mathtt{k})$-*stability*.

Let $C = (\gamma_0 s_0 \gamma_1), \ldots (\gamma_{i-1} s_{i-1} \gamma_i), \ldots$ be a computation. Let $R_p^i(C)$ be the set of neighbors from which p reads some communication variables in step $(\gamma_i s_i \gamma_{i+1})$. Let $\sharp R_p(C) = |R_p^0(C) \cup \cdots \cup R_p^i(C) \cup \ldots|$.

Definition 4 (k-Stable) *A protocol is* k-stable *if in every computation C, every process p satisfies $\sharp R_p(C) \leq k$.*

Definition 5 ($\Diamond$-k-Stable) *A protocol is* $\Diamond$-k-stable *if in every computation C, there is a suffix C' such that every process p satisfies $\sharp R_p(C') \leq k$.*

Definition 6 ($\Diamond$-(x,k)-Stable) *A protocol is* $\Diamond$-$(\mathtt{x},\mathtt{k})$-stable *if in every computation C, there are a subset $\mathcal{S}$ of x processes and a suffix C' such that every process $p \in \mathcal{S}$ satisfies $\sharp R_p(C') \leq k$.*

4. Impossibility Results

We now provide a general condition on the output of communication variables that prevents the existence of some communication stable solutions. Informally, if the communication variables of two neighboring processes p and q can be in two states α_p and α_q that are legitimate separately but not simultaneously, there exists no $\Diamond$-k-stable solution for $k < \Delta$. This condition, that we refer to by the notion of *neighbor-completeness* is actually satisfied by every silent self-stabilizing solution to the problems we consider in the paper: maximal independent set, maximal matching.

Definition 7 (neighbor-completeness) *A protocol A is said* neighbor-complete *for predicate P if and only if A is silent, self-stabilizes to P, and for every process p, there exists a communication state of p, say α_p, such that:*

1. *There exists a silent configuration where the communication state of p is α_p.*
2. *For every neighbor of p, say q, there exists a communication state of q, say α_q, such that:*
   (a) *Every configuration where the communication state of p is α_p and the communication state of q is α_q violates P.*
   (b) *There exists a silent configuration where the communication state of q is α_q.*

Theorem 1 *There is no* $\Diamond$-k-stable *(even probabilistic)* neighbor-complete *protocol working in arbitrary anonymous networks of degree* $\Delta > \mathtt{k}$.

Proof: Assume, by the contradiction, that there exists a $\Diamond$-k-*stable* protocol that is (deterministically or probabilistically) *neighbor-complete* for a predicate P in any anonymous network of degree $\Delta > \mathtt{k}$.

To show the contradiction, we prove that for any $\Delta > 0$, there exist topologies of degree Δ for which there is no $\Diamond$-k-*stable* protocol that is *neighbor-complete* for P with $\mathtt{k} = \Delta - 1$. This result implies the contradiction for any $\mathtt{k} < \Delta$.

We first consider the case $\Delta = 2$. (The case $\Delta = 1$ can be easily deduce using a network of two processes and following the same construction as the one for $\Delta = 2$.) We will then explain how to generalize the case $\Delta = 2$ for any $\Delta \geq 2$.

Case $\Delta = 2$ *and* $\mathtt{k} = \Delta - 1$*:* Consider an anonymous chain of five processes p_1, p_2, p_3, p_4, and p_5. (Figure 1 may help the reader.)

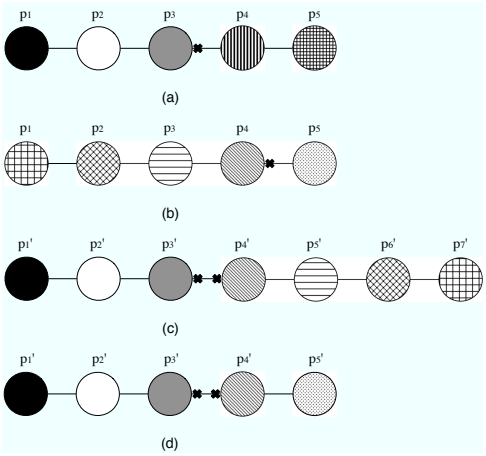

Figure 1. The black crosses indicate that the communication variable of a process is not read by a neighbor

By Definition, there exists a communication state of p_3, say α_3 such that:

1. There exists a silent configuration γ_3 where the communication state of p_3 is α_3.
2. For every neighbor of p_3, p_i ($i \in \{2, 4\}$), there exists a communication state of p_i, say α_i, such that:
   (a) Any configuration where the communication state of p_3 is α_3 and the communication state of p_i is α_i violates P.
   (b) There exists a silent configuration γ_i where the communication state of p_i is α_i.

From the configuration γ_3, the system eventually reaches a silent configuration γ'_3 from which p_3 stops to read the communication variables of one neighbor because the degree of p_3 is equal to Δ. Without loss of generality, assume that this neighbor is p_4. (As in the configuration (a) in Figure 1.) As γ_3 is silent, the communication state of p_3 in γ'_3 is the same as in γ_3: α_3.

Similarly, from γ_4 (γ_i with $i = 4$), the system eventually reaches a silent configuration γ'_4 from which p_4 stops to read the communication variables of one neighbor p_j with $j \in \{3,5\}$ and where the communication state of p_4 is α_4.

Consider the following two cases:

- $p_j = p_5$. (As in the configuration (b) in Figure 1). Consider a new network of seven processes: p'_1, ..., p'_7. Assume the following initial configuration γ: Any process p'_i with $i \in \{1, 2, 3\}$ has the same state as p_i in γ'_3, p'_4 has the state of p_4 in γ'_4, p'_5 has the state of p_3 in γ'_4, p'_6 has the state of p_2 in γ'_4, and p'_7 has the state of p_1 in γ'_4. (This configuration corresponds to the configuration (c) of Figure 1.) We can then remark that p'_3 is in the same situation that p_3 in the configuration γ'_3, so p'_3 does not read the communication variables of p'_4. Similarly, p'_4 does not read the communication variables of p'_3. Moreover, no process modifies the content of its communication variable, otherwise they can do the same in γ'_3 or γ'_4 and this contradicts the fact that γ'_3 and γ'_4 are silent. Hence, γ is silent and, as p'_3 and p'_4 have the same communication state in γ as p_3 in γ_3 and p_4 in γ_4, γ violates P. Thus, any computation starting from γ never converges to a configuration satisfying P, *i.e.*, protocol A is not self-stabilizing for P, a contradiction.
- $p_j = p_3$. This case is similar to the previous one: By constructing a configuration such as the configuration (d) in Figure 1, we also obtain a contradiction.

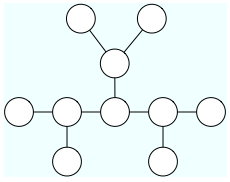

Figure 2. Generalization for $\Delta = 3$

The previous proof can be generalizated for $\mathtt{k} = \Delta - 1$ and $\Delta > 2$ using a graph of $\Delta^2 + 1$ nodes where there is a node of degree Δ (the role of this node is the same as node p_3 in the case $\mathtt{k} = \Delta - 1$ and $\Delta = 2$) that is linked to Δ nodes of degree Δ. Each of these last Δ nodes being linked to $\Delta - 1$ pendent nodes. Figure 2 depicts the generalization for $\Delta = 3$. □

Definition 8 (Dag-orientation) *Let* $\mathcal{S}.p$ *be the set of possible states of process* p*. We say that a system is* dag-oriented iff *for every process* p*, there exists a function* $f_p : \mathcal{S}.p \mapsto 2^{\Gamma.p}$ *and a subset* $Succ.p \subseteq \Gamma.p$ *such that:*

- $\forall \alpha_p \in \mathcal{S}.p$, $f_p(\alpha_p) = Succ.p$, *and*
- *The directed subgraph* $G' = (\Pi, E')$ *where* $E' = \{(p,q), p \in \Pi \wedge q \in Succ.p\}$ *is a* dag.[2]

The next theorem shows that even assuming a *rooted* and/or *dag-oriented* network, it is impossible to design $\mathtt{k}$-*stable neighbor-complete* protocols for $\mathtt{k} < \Delta$. The theorem can be proven using a constructive argument similar to the one used in the proof of Theorem 1.

Theorem 2 *Let* $\mathtt{k} < \Delta$. *There is no* $\mathtt{k}$-stable *(even probabilistic)* neighbor-complete *protocol in any arbitrary rooted and dag-oriented network.*

5. Protocols

We now illustrate the notions of 1-*efficiency* and $\Diamond$-$(\mathtt{x}, 1)$-*stability* in self-stabilization with two protocols for the *maximal independent set* and the *maximal matching*, respectively. These two protocols are designed for colored network of arbitrary topology. By colored we mean that every process p has a constant $C.p$ such that for every neighbor q, $C.p \neq C.q$. We also assume that the colors are ordered following the relation $\prec$. We prove the correctness and study the stabilization time of the two protocols. Finally, we exhibit lower bounds on the number of processes that are eventually "1-stable".

5.1. Maximal Independent Set

We first consider the *maximal independent set* (MIS) problem. An *independent set* of the network is a subset of processes such that no two distinct processes of this set are neighbors. An independent set S is said *maximal* if no proper superset of S is an independent set.

In the *maximal independent set* problem, each process p computes a local Boolean function $inMIS.p$ that decides if p is in the maximal independent set. The *MIS predicate* is $true$ if and only if the subset $\{q \in \Pi, inMIS.q\}$ is a maximal independent set of the network. In any self-stabilizing MIS protocol, the legitimate configurations are those satisfying the *MIS predicate*.

We propose in Figure 3 a 1-*efficient* protocol called $\mathcal{MIS}$ that stabilizes to the *MIS predicate*. $\mathcal{MIS}$ works in arbitrary networks assuming local coloring on processes. Using such colors is very usefull mainly because of the following theorem:

Theorem 3 *Let* E' *be the set of oriented edges such that* $(p,q) \in E'$ *if and only if* p *and* q *are* neighbors *and* $C.p \prec C.q$. *The oriented graph* $G' = (\Pi, E')$ *is a directed acyclic graph (*dag*).*

2. Directed Acyclic Graph

Proof: Assume, by the contradiction, that there is a cycle $p_0 \dots p_k$ in G'. Then, there is an oriented edge (p_k, p_0) which means that: (1) p_0 and p_k are neighbors and (2) $C.p_k \prec C.p_0$. Now, $\forall i \in [0 \dots k-1]$, $C.p_i \prec C.p_{i+1}$. So, by transitivity, $C.p_0 \prec C.p_k$ and this contradicts (2). □

In $\mathcal{MIS}$, any process p maintains the communication variable $S.p$ that has two possible states: $Dominator$ or $dominated$. $S.p$ states if p is in the independent set ($Dominator$) or not ($dominated$). Hence, in $\mathcal{MIS}$, the function $inMIS.p$ just consists in testing if $S.p = Dominator$. The legitimate configurations of $\mathcal{MIS}$ are those satisfying:

1. $\forall p \in \Pi$, $(S.p = Dominator) \Rightarrow (\forall q \in \Gamma.p, S.q = dominated)$.
2. $\forall p \in \Pi$, $(S.p = dominated) \Rightarrow (\exists q \in \Gamma.p, S.q = Dominator)$.

The first condition states that the set of $Dominators$ is an independent set, while the second condition states that the independent set is maximal.

We now outline the principles of $\mathcal{MIS}$. First, we use the internal variable, cur, to get the communication efficiency: a process p only reads the communication state of the neighbor pointed out by $cur.p$. Then, depending of $S.p$, each process p adopts the following strategy:

- If $S.p = Dominator$, then p checks one by one (in a round robin manner) the communication states of its neighbors until it points out a neighbor q that is also a $Dominator$. In such a case, either p or q must become $dominated$ to satisfy Condition 1. We then use the colors to make a deterministic choice between p and q. Note that in a legitimate configuration, every $Dominator$ process continues to check its neighbors all the time.
- If $S.p = dominated$, then p must have the guarantee that one of its neighbor is a $Dominator$. Hence, p switches $S.p$ from $dominated$ to $Dominator$ if the neighbor it points out with $cur.p$ is not a $Dominator$ (*i.e.*, $S.(cur.p) = dominated$). Also, to have a faster convergence time, p switches $S.p$ from $dominated$ to $Dominator$ if the neighbor it points out with $cur.p$ has a greater color (even if it is a $Dominator$).

We now show the correctness of $\mathcal{MIS}$ (Theorem 4). We then show in Theorem 5 that $\mathcal{MIS}$ is $\Diamond$-$(\lfloor \frac{\mathcal{L}_{max}+1}{2} \rfloor, 1)$-*stable* where $\mathcal{L}_{max}$ is the length (number of edges) of the longest elementary path in the network.

We show that $\mathcal{MIS}$ stabilizes to the *MIS predicate* in two steps: (1) We first show that any silent configuration of $\mathcal{MIS}$ satisfies the *MIS predicate*. (2) We then show that $\mathcal{MIS}$ reaches a silent configuration starting from any configuration in $O(\Delta \sharp \mathcal{C})$ rounds where $\sharp \mathcal{C}$ is the number of colors used in the network.

Communication Variable: $S.p \in \{Dominator, dominated\}$ **Communication Constant:** $C.p$: color **Internal Variable:** $cur.p \in [1 \ldots \delta.p]$

Actions:

$(S.(cur.p) = Dominator \wedge C.(cur.p) \prec C.p \wedge S.p = Dominator) \rightarrow S.p \leftarrow dominated$

$[(S.(cur.p) = dominated \vee C.p \prec C.(cur.p)) \wedge (S.p = dominated)] \rightarrow S.p \leftarrow Dominator; cur.p \leftarrow (cur.p \bmod \delta.p) + 1$

$(S.p = Dominator) \rightarrow cur.p \leftarrow (cur.p \bmod \delta.p) + 1$

Figure 3. Protocol $\mathcal{MIS}$ for any process p

Lemma 1 *Any silent configuration of $\mathcal{MIS}$ satisfies the* MIS predicate.

Proof: The silent configurations of $\mathcal{MIS}$ are those from which all the S variables are fixed. So, in such a configuration γ, any $Dominator$ has no neighbor that is also a $Dominator$, otherwise at least one of the $Dominator$ process eventually becomes a $dominated$ process following the first action of the protocol. Hence, the set of $Dominator$ processes in γ is an independent set. Moreover, any $dominated$ process has a $Dominator$ as neighbor in γ. Actually the neighbor pointed out by the cur-pointer is a $Dominator$. Hence the independent set in γ is maximal. □

Let $CSET = \{C.p, p \in \Pi\}$, $\sharp\mathcal{C} = |CSET|$, and $\forall c \in CSET$, $\texttt{Rank}(c) = |\{c' \in CSET, c' \prec c\}|$.

Lemma 2 *Starting from any configuration, any computation of $\mathcal{MIS}$ reaches a silent configuration in at most $\Delta \times \sharp\mathcal{C}$ rounds.*

Proof: This lemma can be deduced by proving the following induction: $\forall p \in \Pi, \texttt{Rank}(C.p) = i$, the variable $S.p$ is fixed after at most $\Delta \times (i+1)$ rounds. □

By Lemmas 1 and 2, follows:

Theorem 4 *$\mathcal{MIS}$ is a* 1*-efficient protocol that stabilizes to the* MIS predicate *in any colored network.*

The following theorem shows a lower bound on the number of processes that are eventually "1-stable".

Theorem 5 *$\mathcal{MIS}$ is $\Diamond$-$(\lfloor \frac{\mathcal{L}_{max}+1}{2} \rfloor, 1)$-stable where $\mathcal{L}_{max}$ is the length (number of edges) of the longest elementary path in the network.*

Proof: Let $\mathcal{L}_{max}$ be the length (number of edges) of the longest elementary path in the network. Once stabilized, at most $\lceil \frac{\mathcal{L}_{max}+1}{2} \rceil$ processes in this path are $Dominators$, otherwise at least two $Dominators$ are neighbors and the system is not stabilized. As a consequense, at least $\lfloor \frac{\mathcal{L}_{max}+1}{2} \rfloor$ processes are $dominated$ in a silent configuration and $\mathcal{MIS}$ is $\Diamond$-$(\lfloor \frac{\mathcal{L}_{max}+1}{2} \rfloor, 1)$-*stable*. □

5.2. Maximal Matching

We now consider the *maximal matching* problem. A matching of the network is a subset of edges in which no pair of edges has a common incident process. A matching M is *maximal* if no proper superset of M is also a matching.

In *maximal matching problem*, each process p computes $\delta.p$ local Boolean functions $inMM[q].p$ (one for each neighbor q) that decide if the edge $\{p, q\}$ is in the maximal matching. The *maximal matching predicate* is $true$ if and only if the subset of edges $\{\{p, q\} \in E, inMM[q].p \vee inMM[p].q\}$ is a *maximal matching* of the network. In any self-stabilizing maximal matching protocol, the legitimate configurations are those satisfying the *maximal matching predicate*.

We propose in Figure 4 a 1-*efficient* protocol called $\mathcal{MATCHING}$ that stabilizes to the *maximal matching* predicate. The proposed protocol works in arbitrary networks still assuming the (local) coloring on processes.

$\mathcal{MATCHING}$ derives from the protocol in [14], but with some adaptations to get the 1-*efficiency*. As previously, each process p has the communication constant color $C.p$ and uses the internal cur-pointer to designate the current neighbor from which it reads the communication variables.

The basic principle of the protocol is to create pairs of *married* neighboring processes, the edges linking such pairs being in the maximal matching. To that goal, every process p maintains the variable $PR.p$. Either $PR.p$ points out a neighbor or is equal to 0. Two neighboring processes are *married* if and only if their PR-values point out to each other. A process that is not married is said *unmarried*. The predicate $PRmarried(p)$ states if the process p is currently married, or not. Hence, for every process p and every p's neighbor q, $inMM[q].p \equiv (PRmarried(p) \wedge PR.p = q)$. If $PR.p = 0$, then this means that p is unmarried and does not currently try to get married. In this case, p is said *free*. If $PR.p \neq 0$, then p is either married or tries to get married with the neighbor pointed out by $PR.p$. Hence, the value of $PR.p$ is not sufficient to allow all neighbors of p to determine its current status (married or unmarried). We use the Boolean variable $M.p$ to let neighboring processes of p know if p is married or not. Using these variables, the protocol is composed of 6 actions (ordered from the highest to the lowest priority). Using these actions, each process p applies the following strategies:

- p is only allowed to be (or try to get) married with the neighbor pointed out by $cur.p$. So, if $PR.p \notin$

Communication Variables:
$M.p \in \{true, false\}$

$PR.p \in \{0 \ldots \delta.p\}$

Communication Constant:
$C.p$: color

Internal Variable:
$cur.p \in [1 \ldots \delta.p]$

Predicate:
$PRmarried(p) \equiv (PR.p = cur.p \wedge PR.(cur.p) = p)$

Actions:

$(PR.p \notin \{0, cur.p\})$	$\rightarrow$	$PR.p \leftarrow cur.p$
$(M.p \neq PRmarried(p))$	$\rightarrow$	$M.p \leftarrow PRmarried(p)$
$(PR.p = 0 \wedge PR.(cur.p) = p)$	$\rightarrow$	$PR.p \leftarrow cur.p$
$(PR.p = cur.p \wedge PR.(cur.p) \neq p \wedge (M.(cur.p) \vee C.(cur.p) \prec C.p))$	$\rightarrow$	$PR.p \leftarrow 0$
$(PR.p = 0 \wedge PR.(cur.p) = 0 \wedge C.p \prec C.(cur.p) \wedge \neg M.(cur.p))$	$\rightarrow$	$PR.p \leftarrow cur.p$
$(PR.p = 0 \wedge (PR.(cur.p) \neq 0 \vee C.(cur.p) \prec C.p \vee M.(cur.p)))$	$\rightarrow$	$cur.p \leftarrow (cur.p \bmod \delta.p) + 1$

Figure 4. Protocol $\mathcal{MATCHING}$ For any process p

$\{0, cur.p\}$ then $PR.p$ is set to $cur.p$. Actually, if $PR.p = q$ such that $q \notin \{0, cur.p\}$, then $PR.p = q$ since the initial configuration.

- p must inform its neighbors of its current status, *i.e.* married or unmarried, using $M.p$. To compute the value of $M.p$ we use the predicate $PRmarried(p)$: if $M.p \neq PRmarried(p)$, then $M.p$ is set to $PRmarried(p)$.
- If p is free and p is pointed out by the PR-variable of a neighbor q, this means that q proposes to p to get married. In this case, p accepts by setting $PR.p$ to q.
- p resets $PR.p$ to 0 when the neighbor pointed out by $PR.p$ (i) is married with another process or (ii) has a lower color than p (*w.r.t.*, $\prec$). Condition (i) prevents p to wait for an already married process. Condition (ii) is used to break the initial cycles of PR-values.
- If p is free, then it must try to get married. The two last rules achieve this goal. p tries to find a neighbor that is free and having a higher color than itself (to prevent cycle creation). So, p increments $cur.p$ until finding an neighbor that matches this condition. In this latter case, p sets $PR.p$ to $cur.p$ in order to propose a marriage.

We now show the correctness of $\mathcal{MATCHING}$ (Theorem 6). We then show in Theorem 7 that $\mathcal{MATCHING}$ is $\Diamond$-$(\lceil \frac{2m}{2\Delta-1} \rceil, 1)$-*stable*.

Lemma 3 *In any silent configuration of $\mathcal{MATCHING}$, every process is either* free *or* married*.*

Proof: Assume, by the contradiction, that there is a silent configuration of $\mathcal{MATCHING}$ where there is a process p_0 that is neither *free* nor *married*. Then, by definition, $PR.p_0 = p_1$ such that $p_1 \neq 0$ (p_0 is not *free*) and $PR.p_1 \neq p_0$ (p_0 is *unmarried*). Also, $cur.p_0 = p_1$ otherwise p_0 is enabled to set $PR.p_0$ to $cur.p_0$, this contradicts the facts that the configuration is silent. Similarly, the fact that p_0 is *unmarried* implies that $M.p_0 = false$.

As $PR.p_1 \neq p_0$ and $cur.p_0 = p_1$, we have $M.p_1 = false$ and $C.p_0 \prec C.p_1$ otherwise p_0 is enabled to set $PR.p_0$ to 0 and the configuration is not silent, a contradiction. In addition, $M.p_1 = false$ implies that p_1 is *unmarried*. Also, p_1 cannot be *free* otherwise p_1 eventually modify $PR.p_1$ (in the worst case, p_1 increments $cur.p_1$ until $cur.p_1 = p_0$ and then sets $PR.p_1$ to p_0). To sum up, p_1 is a neighbor of p_0 such that $C.p_0 \prec C.p_1$ and that is neither *free* nor *married*.

Repeating the same argument for p_1 as we just did for p_0, it follows that p_1 has a neighbor p_2 such that $C.p_1 \prec C.p_2$ and that is neither *free* nor *married*, and so on.

However, the sequence of processes p_0, p_1, $p_2 \ldots$ cannot be extended indefinitely since each process must have a lower color than its preceding one. Hence, this contradicts the initial assumption. □

Lemma 4 *Any silent configuration of $\mathcal{MATCHING}$ satisfies the* maximal matching predicate*.*

Proof: We show this lemma in two steps: (i) First we show that, in a silent configuration, the set A of edges $\{p, q\}$ such that $(PRmarried(p) \wedge PR.p = q)$ is a matching. (ii) Then, we show that this matching is a maximal. □

The next lemma can be trivially deduce from the fact that if a process p initially satisfies $PR.p \notin \{0, cur.p\}$, then it sets $PR.p$ to $cur.p$ during the first round using the first action of the protocol.

Lemma 5 *After the first round, every process p satisfies $PR.p \in \{0, cur.p\}$ forever.*

Lemma 6 *Let $A \in \Pi$ be a maximal connected subset of* unmarried *processes in some configuration after the first round. If $|A| \geq 2$, then after at most $2\Delta + 2$ rounds the size of A decreases by at least 2.*

Proof: Let S be the suffix of the computation that starts after the end of the first round. Let $\Gamma(A)$ be the set of process p such that $p \notin A$ and p has a neighbor $q \in A$.

First the size of A cannot increase because once *married*, a process remains *married* forever. Assume now, by the contradiction, that A does not decrease of at least 2 during $2\Delta + 2$ rounds in S. This implies that no two process of A get married during this period.

Let S' be the prefix of S containing $2\Delta + 2$ rounds. We show the contradiction using the following four steps:

1. After one round in S', every process p satisfies: $(p \in \Gamma(A) \Rightarrow M.p) \wedge (p \in A \Rightarrow \neg M.p)$.
2. After two rounds in S', for every process p, if $PR.p \neq 0$, then $PR.p \neq 0$ holds until the end of S'.
3. After $\Delta + 2$ rounds in S', for every process p in A, we have either (1) $PR.p = q$, $q \in A$ or (2) $PR.p = 0$ and every neighbor $q \in A$ satisfies $PR.q \in A$.
4. In at most $2\Delta + 2$ rounds, at least two neighboring processes in A get married, which contradicts the initial assumption.

□

Lemma 7 *Starting from any configuration, any computation of $\mathcal{MATCHING}$ reaches a silent configuration in at most $(\Delta + 1)n + 2$ rounds.*

Proof: First, the number of *married* processes cannot decrease. Then, after the first round and until there is a maximal matching in the system, the number of *married* processes increases by at least 2 every $2\Delta + 2$ rounds by Lemma 6. Hence, there is a maximal matching into the networks after at most $(\Delta + 1)n + 1$ rounds. Once maximal matching is available in the network, one more round is necessary so that every *married* process p satisfies $M.p = true$ and every *ummarried* process p satisfies $PR.p = 0$. Hence, starting from any initial configuration, the system reaches a silent configuration in at most $(\Delta+1)n+2$ rounds. □

By Lemmas 4 and 7, follows:

Theorem 6 *$\mathcal{MATCHING}$ is a 1-efficient protocol that stabilizes to the* maximal matching predicate *in any locally-identified network.*

The following theorem shows a lower bound on the number of processes that are eventually "1-stable".

Theorem 7 *$\mathcal{MATCHING}$ is $\Diamond$-$(2\lceil \frac{m}{2\Delta-1} \rceil, 1)$-stable.*

Proof: From [15], we know that any maximal matching in a graph has a size at least $\lceil \frac{m}{2\Delta-1} \rceil$ edges. So, as a process belongs to at most one matched edge, we can conclude that at least $2\lceil \frac{m}{2\Delta-1} \rceil$ processes are eventually matched. As a consequence, $\mathcal{MATCHING}$ is $\Diamond$-$(2\lceil \frac{m}{2\Delta-1} \rceil, 1)$-*stable*. □

6. Concluding Remarks

We focused on improving communication efficiency of self-stabilizing protocols that eventually reach a global fixed point, and devised how much gain can be expected when implementing those protocols in a realistic model. Our results demonstrate the task difficulty, as most systematic improvements are impossible to get, yet also shows that some global improvement can be achieved over the least-overhead solutions known so far, the so-called *local checking* self-stabilizing protocols.

While we demonstrated the effectiveness of our scheme to reduce communication need on several local checking examples, the possibility of designing an efficient general transformer for protocols matching the local checking paradigm remains an open question. This transformer would allow to easily get more efficient communication in the stabilized phase or in absence of faults, but the effectiveness of the transformed protocol in the stabilizing phase is yet to be known.

References

[1] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control." *Commun. ACM*, vol. 17, no. 11, pp. 643–644, 1974.

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

[3] S. Katz and K. J. Perry, "Self-stabilizing extensions for message-passing systems." *Distributed Computing*, vol. 7, no. 1, pp. 17–26, 1993.

[4] J. Beauquier, S. Delaët, S. Dolev, and S. Tixeuil, "Transient fault detectors," *Distributed Computing*, vol. 20, no. 1, pp. 39–51, June 2007. [Online]. Available: http://www.springerlink.com/content/m267v22224127575/

[5] B. Awerbuch, B. Patt-Shamir, and G. Varghese, "Self-stabilization by local checking and correction (extended abstract)," in *FOCS*. IEEE, 1991, pp. 268–277.

[6] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev, "Self-stabilization by local checking and global reset (extended abstract)." in *Distributed Algorithms, 8th International Workshop, WDAG '94*, ser. Lecture Notes in Computer Science, G. Tel and P. M. B. Vitányi, Eds., vol. 857. Springer, 1994, pp. 326–339.

[7] S. Dolev, M. G. Gouda, and M. Schneider, "Memory requirements for silent stabilization," *Acta Inf.*, vol. 36, no. 6, pp. 447–462, 1999.

[8] M. Larrea, A. Fernández, and S. Arévalo, "Optimal implementation of the weakest failure detector for solving consensus." in *SRDS*, 2000, pp. 52–59.

[9] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "On implementing omega with weak reliability and synchrony assumptions," in *PODC*, 2003, pp. 306–314.

[10] M. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "Communication-efficient leader election and consensus with limited link synchrony." in *PODC*, 2004, pp. 328–337.

[11] C. Delporte-Gallet, S. Devismes, and H. Fauconnier, "Robust stabilizing leader election," in *SSS*, ser. Lecture Notes in Computer Science, T. Masuzawa and S. Tixeuil, Eds., vol. 4838. Springer, 2007, pp. 219–233.

[12] S. Dolev, A. Israeli, and S. Moran, "Resource bounds for self-stabilizing message-driven protocols," *SIAM J. Comput.*, vol. 26, no. 1, pp. 273–290, 1997.

[13] A. Israeli and M. Jalfon, "Token management schemes and random walks yield self-stabilizing mutual exclusion." in *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, 1990, pp. 119–131.

[14] F. Manne, M. Mjelde, L. Pilard, and S. Tixeuil, "A new self-stabilizing maximal matching algorithm," in *Proceedings of the 14th International Colloquium on Structural Information and Communication Complexity (Sirocco 2007)*, vol. 4474. Springer Verlag, June 2007, pp. 96–108. [Online]. Available: http://www.springerlink.com/content/u723x36620p80066/

[15] T. C. Biedl, E. D. Demaine, C. A. Duncan, R. Fleischer, and S. G. Kobourov, "Tight bounds on maximal and maximum matchings," *Discrete Mathematics*, vol. 285, no. 1-3, pp. 7–15, 2004.