On Bootstrapping Topology Knowledge in Anonymous Networks

TOSHIMITSU MASUZAWA
Osaka University
and
SÉBASTIEN TIXEUIL
Université Pierre et Marie Curie

In this article, we quantify the amount of "practical" information (i.e., views obtained from the neighbors, colors attributed to the nodes and links) to obtain "theoretical" information (i.e., the local topology of the network up to distance k) in anonymous networks. In more detail, we show that a coloring at distance $2k + 1$ is necessary and sufficient to obtain the local topology at distance k that includes outgoing links. This bound drops to $2k$ when outgoing links are not needed. A second contribution of this article deals with color bootstrapping (from which local topology can be obtained using the aforementioned mechanisms). On the negative side, we show that (*i*) with a distributed daemon, it is impossible to achieve deterministic color bootstrap, even if the whole network topology can be instantaneously obtained, and (*ii*) with a central daemon, it is impossible to achieve distance m when instantaneous topology knowledge is limited to $m - 1$. On the positive side, we show that (*i*) under the k-central daemon, deterministic self-stabilizing bootstrap of colors up to distance k is possible provided that k-local topology can be instantaneously obtained, and (*ii*) under the distributed daemon, probabilistic self-stabilizing bootstrap is possible for any range.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Verification*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; D.1.1 [**Programming Techniques**]: General; D.1.3 [**Programming Techniques**]: Concurrent Programming; D.2.5 [**Software Engineering**]: Design—*Methodologies*

General Terms: Algorithm, Design, Theory

Additional Key Words and Phrases: Topology, anonymous networks, stabilization, daemon

An earlier version of this article was presented at the 8th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'06).
T. Masuzawa is supported in part by Global COE (Center of Excellence) Program of MEXT and Grant-in-Aid for Scientific Research ((B)19300017) of JSPS. S. Tixeuil was partially supported by the FRAGILE and SOGEA projects of the ACI "Sécurité et Informatique" of the French Ministry of Research.
Part of this work was done while S. Tixeuil was visiting Osaka University.
Authors' addresses: T. Masuzawa, Osaka University, Japan; S. Tixeuil, Université Pierre et Marie Curie, Paris 6, France.

ACM Reference Format:
Masuzawa, T., and Tixeuil, S. 2008. On bootstrapping topology knowledge in anonymous networks. ACM Trans. Autonom. Adapt. Syst. 4, 1, Article 8 (January 2009), 27 pages. DOI = 10.1145/1462187.1462195 http://doi.acm.org/10.1145/1462187.1462195

1. INTRODUCTION

Topology update is an essential problem in distributed computing [Spinelli and Gallager 1989]. It has direct applicability in practical systems. For example, link-state based routing protocols such as OSPF (Open Shortest Path First [Moy 1998]) use topology discovery mechanisms to compute the routing tables. Recently, the problem came to the fore with the introduction of ad hoc wireless sensor networks, such as the Berkeley mote network [Hill and Culler 2002], where topology discovery is essential for routing decisions.

Self-stabilization is now considered to be the most general technique to design a system to tolerate arbitrary transient faults. A self-stabilizing system guarantees that starting from an arbitrary state, the system converges to a legal configuration in a finite number of steps and remains in a legal state until another fault occurs (see also Dolev [2000]). Intuitively, a self-stabilizing topology update algorithm guarantees that, even if the system is started from a global state where the topology information is erroneous, within a finite number of steps, correct topology information is maintained at every node.

1.1 Bootstrapping Topology

In this article, we investigate the problem of distributed topology update in an arbitrary anonymous network. Each node is only aware of its neighboring peers and it needs to learn the topology of the network up to some finite distance k. "Bootstrapping" topology knowledge refers to the fact that each node is required to construct topology at distance k with only topology knowledge up to distance $(k-1)$. While this task can be performed in identifier-based networks (where each node has a unique identifier) [Dolev and Herman 1997], to our knowledge no solution exists in *anonymous* networks, where nodes have no identifier whatsoever. While most modern networks have identifiers for nodes in the network (e.g., the address of the network card), it is also likely (with the advent of very heterogeneous systems including RFIDs, computers, smartphones) that there are identifier clashes (either unintentional due to conflicting addressing schemes, or intentional due to a reconfiguration of the network card). An algorithm that is able to perform in anonymous networks will also behave correctly in a network with identifiers, but the converse is not true.

1.2 Related Works

While most distributed algorithms dealing with topology information are self-stabilizing [Dolev and Herman 1997; Delaët and Tixeuil 2002; Masuzawa 1995], they only deal with networks where nodes have unique identifiers. For the particular case of *detecting* transient inconsistencies locally, Beauquier et al.

[2007] investigate the amount of local knowledge that is necessary for various kinds of problems. Note that the local knowledge that is used includes the local topology of a particular node.

In the context of anonymous networks, self-stabilizing solutions either run on networks where topology information is known (ring, tree, etc), or consider problems that can be solved without topology information [Boldi and Vigna 2002]. For classical algorithms, Sakamoto [2000] provides a classification of problems according to how much asymmetry is initially provided in the system (e.g., a unique leader vs. a set of k leaders), and Fraigniaud et al. [2001] provide a scheme for unique naming when a distinguished process is present. Also, Yamashita and Kameda [1999] study the feasibility of leader election when the processor identity numbers are not distinct, and use techniques based on (infinite) colored views obtained from the neighbors as well as global knowledge (the size of the network) and a synchronous setting.

1.3 Our Contribution

In this article, we quantify the amount of "practical" information (i.e., views obtained from the neighbors, colors attributed to the nodes and links) to obtain "theoretical" information (i.e., the local topology of the network up to distance k) in anonymous networks. In more detail, we show that a coloring at distance $2k + 1$ is necessary and sufficient to obtain the local topology at distance k that includes outgoing links. This bound drops to $2k$ when outgoing links are not needed. A second contribution of this article deals with color bootstrapping (from which local topology can be obtained using the aforementioned mechanisms). On the negative side, we show that (i) with a distributed daemon, it is impossible to achieve deterministic color bootstrap, even if the whole network topology can be instantaneously obtained, and (ii) with a central daemon, it is impossible to achieve distance m when instantaneous topology knowledge is limited to $m - 1$. On the positive side, we show that (i) under the k-central daemon, deterministic bootstrap of colors up to distance k is possible provided that k-local topology can be instantaneously obtained, and (ii) under the distributed daemon, probabilistic bootstrap is possible for any range.

2. MODEL

2.1 Distributed Systems

A *distributed system* $S = (P, L)$ consists of a set $P = \{v_1, v_2, \ldots, v_n\}$ of processes and a set L of bidirectional communication links (simply called links). A link is an unordered pair of distinct processes. A distributed system S can be regarded as a graph whose vertex set is P and whose link set is L, so we use some graph terminology to describe a distributed system S. The *girth* of a graph is the length of a shortest (simple) cycle in the graph; and the *circumference* is the length of a longest (simple) cycle. The girth and circumference of an acyclic graph are defined to be infinity (∞). The *k-th power* G^k of a graph G is a supergraph formed by adding an edge between all pairs of vertices of G with distance at most k.

Processes u and v are called *neighbors* if $(u, v) \in L$. The set of neighbors of a process v is denoted by N_v, and its cardinality (the *degree* of v) is denoted by $\Delta_v (= |N_v|)$. The degree Δ of a distributed system $S = (P, L)$ is defined as $\Delta = \max\{\Delta_v \mid v \in P\}$. We do not assume existence of a unique identifier of each process. Instead we assume each process can distinguish its neighbors from each other by locally arranging them in some arbitrary order: the k-th neighbor of a process v is denoted by $N_v(k)$ $(1 \le k \le \Delta_v)$.

Processes can communicate with their neighbors through link registers. For each pair of neighboring processes u and v, there are two link registers $r_{u,v}$ and $r_{v,u}$. Message transmission from u to v is realized as follows: u writes a message to link register $r_{u,v}$ and then v reads it from $r_{u,v}$. The link register $r_{u,v}$ is called an *output register* of u and is called an *input register* of v. The set of all output (resp. input) registers of u is denoted by Out_u (resp. In_u), that is, $Out_u = \{r_{u,v} \mid v \in N_u\}$ and $In_u = \{r_{v,u} \mid v \in N_u\}$. The variables that are maintained by process denote their states. Similarly, the values of the variables stored in each register denote the state of these registers. The algorithm executed by each process is described by a finite set of guarded actions of the form $\langle\text{guard}\rangle \longrightarrow \langle\text{statement}\rangle$. Each guard of process p is a Boolean expression involving the variables of p and its input registers. When there is no explicit register communication described in the algorithm, it is implicitly assumed that the state of the process is copied in each output register after the execution of any action.

A global state of a distributed system is called a *configuration* and is specified by a product of states of all processes. We define C to be the set of all possible configurations of a distributed system S. For a process set $R \subseteq P$ and two configurations ρ and ρ', we denote $\rho \overset{R}{\mapsto} \rho'$ when ρ changes to ρ' by executing an action of each process in R simultaneously. A *schedule* of a distributed system is an infinite sequence of process sets. Let $Q = R^1, R^2, \ldots$ be a schedule, where $R^i \subseteq P$ holds for each i $(i \ge 1)$. An infinite sequence of configurations $e = \rho_0, \rho_1, \ldots$ is called an *execution* from an initial configuration ρ_0 by a schedule Q, if e satisfies $\rho_{i-1} \overset{R^i}{\mapsto} \rho_i$ for each i $(i \ge 1)$.

Process actions are executed atomically, and we consider in this paper three kinds of scheduling possibilities:

(1) the *distributed daemon* schedules the actions of processes, in such a way that any subset of processes can simultaneously execute their actions,
(2) the *central daemon* schedules the actions of processes such that exactly one process executes its actions at a given time,
(3) the *k-central daemon* schedules the actions of processes such that no two processes that are k hops away or less execute their actions at the same time.

Of course, the central scheduler is a special case of the k-central scheduler, which in turn is a special case of the distributed scheduler. The most realistic scheduler is the distributed scheduler but the other two can be emulated using a mutual exclusion protocol (for the central scheduler [Dolev 2000]) or a k-local mutual exclusion protocol (for the k-central scheduler [Danturi et al. 2006]), with an additional overhead. Note however that the k-central emulation known so far requires unique identifiers.

The set of all possible executions from $\rho_0 \in C$ is denoted by E_{ρ_0}. The set of all possible executions is denoted by E, that is, $E = \bigcup_{\rho \in C} E_\rho$. We consider *asynchronous* distributed systems where we can make no assumption on schedules except that any schedule is *weakly fair*: every process is contained in infinite number of subsets appearing in any schedule. For complexity results, we use the concept of *round* as the minimum portion of an execution in which every process has the opportunity to execute at least one action.

In this context, a protocol is self-stabilizing for some specification $\mathcal{S}$ if there exists a global predicate $\mathcal{P}$ on configurations, such that the two following conditions are satisfied: (i) any execution starting from a configuration satisfying $\mathcal{P}$ satisfies the problem $\mathcal{S}$ (correctness), and (ii), any computation reaches a configuration that satisfies $\mathcal{P}$ (convergence). In this article, we also consider probabilistic protocols, for which the convergence is only achieved with probability 1 (see Beauquier et al. [1999] for more details). A class of self-stabilizing protocols is that of *silent* protocols [Dolev et al. 1999]: a self-stabilizing protocol is silent if there exists a point in every execution after which the communication registers of any process p (i.e., the set of registers that p's neighbors can read from) have their values *fixed* for the remaining of the execution.

2.2 Local View and Topology

We now define notions that are central in this paper, and related to local views and local topology.

Definition 1 (*Local Topology*). The local topology at distance k of a node p is the subgraph T_p^k of the communication graph G that contains nodes and edges of G up to distance k from p.[1]

A slightly different definition of local topology is given by Perlman [2000]:

Definition 2 (*P-Local Topology*). The P-local topology at distance k of a node p is the subgraph $P\text{-}T_p^k$ of the communication graph G that contains nodes of G up to distance k from p and edges of G up to distance $k-1$ from p.

The significant difference between the local topology and the P-local topology at distance k is that the former requires exact recognition of the links between processes at distance k but the latter does not. Figure 1 presents an example of 2-local topology vs. 2-P-local topology of node v: the former includes *outgoing pending edges* from nodes at distance 2, and if a link exists between nodes at distance exactly two (e.g. f and c), it is correctly identified. This property does not hold for P-local topology. Note that grayed identifiers are *not* known in the local topology.

Definition 3 (*Local View*). The local view at distance 0 of a node p is a tree V_p^0 rooted at p with its set of adjacent edges. The local view at distance 1 of a

[1]The distance from a node (say p) to an edge (say $e = (q, r)$) is defined as $\min\{dist(p, q), dist(p, r)\}$.

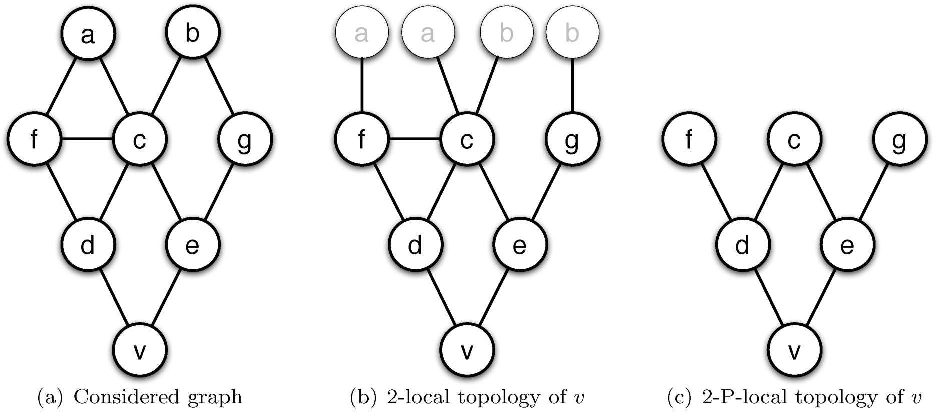


Fig. 1. Local topology vs. P-local topology at distance 2.

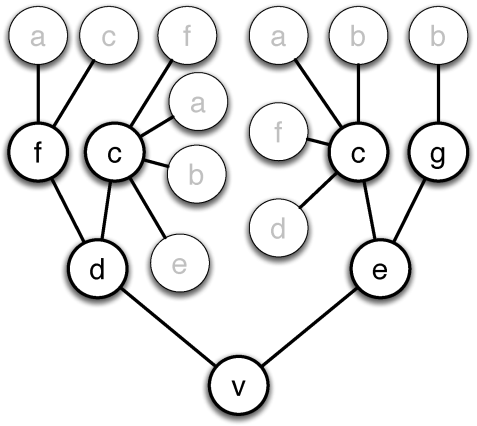


Fig. 2. Local view of v at distance 2 for the same considered graph as in Figure 1.

node p is a tree V_p^1 of height 1 rooted at p that contains one leaf V_q^0 for every neighbor q of p. The local view at distance k of a node p is a tree V_p^k of height k that contains one local view V_q^{k-1} as subtree of p for each neighbor q of p.

Informally, the local view is the knowledge about the network that a node can collect by getting information from its neighbors. In contrast, the local topology is the exact knowledge. An example of a view at distance 2 for process v (for the same graph as in Figure 1) is presented in Figure 2.

It is obvious that the (P-)local topology coincides with the local view when the (P-)local topology is acyclic. Thus, the following two observations hold.

Observation 1. Let k be a strictly positive integer. In any network of girth $2k+2$ or more, $T_p^k = V_p^k$ holds for any process p.

Observation 2. Let k be a strictly positive integer. In any network of girth $2k+1$ or more, $P\text{-}T_p^k \subset V_p^k$ holds for any process p.

Overall the only difference between local topology and P-local topology is whether the outgoing pending edges are included or not.

3. LOCAL VIEW VS. LOCAL TOPOLOGY

In this section, we show that there exists a relation between views (resp. P-views) and local topology. In more detail, when sufficient node or link coloring is provided, it is possible to construct the local topology from the local view (resp. P-view) if all the cycles in a network are sufficiently large or small.

Definition 4 (*k-local Node Coloring*). A coloring of the nodes is k-local in graph G, if any two nodes that are at distance k or less in G have different colors.

Definition 5 (*k-local Link Coloring*). A coloring of the links is k-local in graph G, if any two nodes that are at distance k or less in G' (the dual graph of G) have different colors.

Observation 3. To provide k-local node coloring (resp. k-local link coloring), at least $\min(n, k+1)$ (resp. $\min(m, k+1)$) colors are required, where n (resp. m) is the number of nodes (resp. links) in the network.

The following lemma shows that when every cycle is sufficiently large, there is no problem to identify the local topology and P-local topology from the local view. Note also that the lemma holds independently of the existence of a coloring.

LEMMA 3.1 (GIRTH). *Given a local view at distance k for each process, it is possible to construct a local (resp. P-local) topology at distance k at every node for any network of girth $2k+2$ (resp. $2k+1$) or more.*

PROOF. The lemma immediately follows from Observations 1 and 2. □

In contrast to Lemma 3.1, the following lemma shows that when every cycle is sufficiently small, the local topology and P-local topology can be constructed from the local view with the help of a node or a link coloring.

LEMMA 3.2 (CIRCUMFERENCE). *Given a local view at distance k for each process, and a node or a link coloring at distance $2k$ (resp. $2k-1$), it is possible to construct a local (resp. P-local) topology at distance k at every node for any network of circumference $4k+1$ (resp. $4k-1$) or less.*

PROOF. We first consider the case of a local topology with a $2k$-local node coloring. Algorithm 3.1 constructs T_v^k from V_v^k at each process v with the help of a $2k$-local node coloring. The fundamental underlying idea of the algorithm is as follows.

Algorithm 3.1. Local topology construction (distance k)

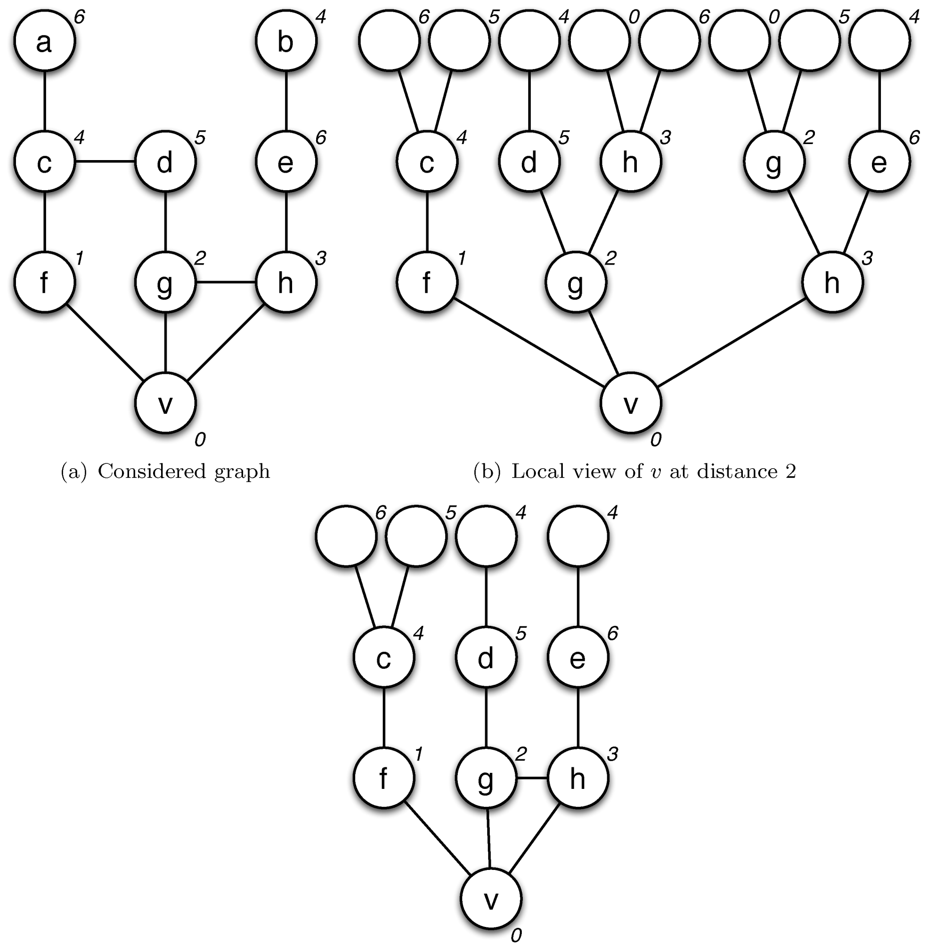
```
Obtain tentative $T_v^k$ from $V_v^k$ by unifying nodes with the same color;
for each outgoing pending edge $(p, q)$ in $T_v^k$ (let $p$ be a node in $V_v^k$)
  get color of $q$ (from $V_u^k$ of a neighbor $u$ of $v$);
  if no process in $T_v^k$ has the same color as $q$;
    $(p, q)$ is assumed to be an outgoing edge of $T_v^k$;
  if a process $r$ in $T_v^k$ has the same color as $q$;
    {only a single process can have the same color as $q$}
    if $dist(q, r) \leq 2k$
      $(p, q)$ is assumed to be edge $(p, r)$;
      {$q$ is assumed to be identical to $r$}
    if $dist(q, r) = 2k + 1$
      $(p, q)$ is tentatively assumed to be edge $(p, r)$;
      {$q$ is tentatively assumed to be $r$}
      construct a cycle $\mathcal{C}$ of length $2k + 1$ from
        a shortest path from $v$ to $p$, edge $(p, r)$, and a shortest path from $r$ to $v$;
      repeat
        extend local view of $v$ along a repeated sequence of colors in $\mathcal{C}$;
      until the extension becomes impossible;
      $(p, q)$ is assumed to be an outgoing edge of $T_v^k$;
```

In a node coloring at distance $2k$, all nodes with the same color in V_v^k represent a same process. Thus, each process v can precisely recognize a local topology at distance k except for links between processes at distance k; v constructs a tentative T_v^k from V_v^k by identifying nodes with the same color as a common process. Figure 3 shows an example of the tentative T_v^k, where the integer associated to each node denotes its color.

Let p be any node at distance k from v and (p, q) be an outgoing pending edge in the tentative T_v^k. For construction of T_v^k, it is sufficient to recognize whether (p, q) is an internal edge or an outgoing pending edge of T_v^k. Process v can get the color of q from the local view of a neighbor of v. When there is no node at distance k or $k - 1$ from v with the same color as q, q is a process at distance $k + 1$ from v and (p, q) is an outgoing pending edge of T_v^k. When there is a node r at distance k or $k - 1$ with the same color as q and $dist(q, r) \leq 2k$ holds, q is identical with r and (p, q) is an internal edge (p, r) of T_v^k. However, when there is a node r with the same color as q and $dist(q, r) = 2k + 1$ holds,[2] q may or may not be identical to r. Note that the $2k$-local node coloring guarantees that there is only a single process with the same color as q at distance k or $k - 1$ from v.

For the node r at distance k from v with the same color as q, process v tentatively considers that q is identical with r until it discovers evidence that q is distinct from r. In the following, we present the strategy for finding the evidence.

[2]$dist(q, r) = 2k + 1$ implies that r is at distance k from v, and thus $dist(q, r)$ is never greater than $2k + 1$.

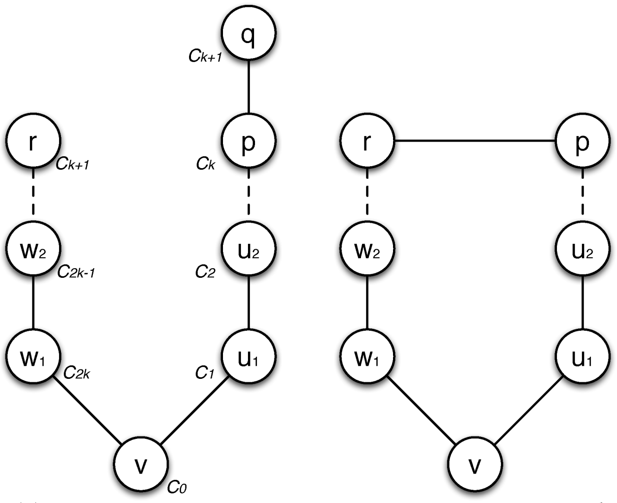


(a) Considered graph (b) Local view of v at distance 2

(c) Tentative 2-local topology of v

Fig. 3. Example of tentative T_v^k obtained from V_v^k ($k = 2$).

Let $v(= u_0), u_1, \ldots, p(= u_k)$ and $v(= w_0), w_1, \ldots r(= w_k)$ be any shortest paths in the local topology of v (see Figure 4). Since $dist(q, r) = 2k + 1$ holds, these two paths can have only v in common, and each process in the paths has a color distinct from any other process in the paths. If $q = r$ holds, it is obvious that the infinite view of v contains an infinitely repeated path of colors $c_0, c_1, \ldots, c_{2k}, c_0, c_1, \ldots,$ where c_i $(0 \leq i \leq k)$ is the color of u_i and c_i $(k + 1 \leq i \leq 2k)$ is the color of w_{2k+1-i}. On the other hand, from the following reason, the infinite view of v contains an infinitely repeated path of colors $c_0, c_1, \ldots, c_{2k}, c_0, c_1, \ldots,$ only if $q = r$ holds. Since the path length is infinite but the number of processes is finite, the path contains a cycle of processes; let the i-th process (or color) in the infinitely long path be the first one that is identical with a previously appearing process (say, the j-th process $(j < i)$). First we can show that $j = 1$, that is, the i-th process is identical with the first process v in



(a) Two considered paths (b) A tentative cycle in T_v^k

Fig. 4. Example of a tentative cycle of length $2k + 1$.

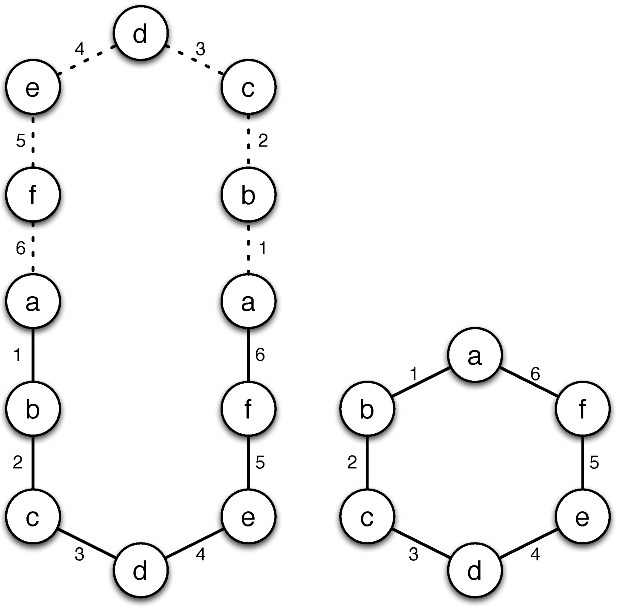
the path; otherwise the $(j - 1)$-th process and the $(i - 1)$-th process have the same color,[3] but their distance is two,[4] which contradicts a node coloring at distance $2k$. Now, by showing that $i = 2k + 2$, we can show that the cycle contained in the path is $v(= u_0), u_1, \ldots p(= u_k), r(= w_k), w_{k-1}, \ldots, v(= w_0)$. Otherwise the length of the cycle is $4k + 2$ or more since the path is a repeated sequence of $(2k + 1)$ colors, which contradicts the assumption that the circumference is $4k + 1$ or less.

Thus, we can eventually determine as follows whether an outgoing pending edge (p, q) is a link between processes at distance k or not: each process tentatively considers that q is identical to process r at distance k in its local topology and extends its local view along the path of the repeated cycles of colors described above. When q is identical with r, it can extend the path infinitely and keeps the tentative local topology. On the other hand, when q represents another process than r, it eventually fails to extend the path (that is evidence of $q \neq r$) and changes the edge (p, q) to an outgoing pending edge.

For the case of a local topology with a link coloring, we can prove the lemma by an similar argument. In this case, we have to determine whether outgoing pending links with a same color in V_v^k are identical or not. The only difference from the above case of a node coloring is that more than two outgoing pending links may have a same color but only a single pair of links can be identical. However, we can identify the pair (if exists) by extending a local view in a similar way to the case of a node coloring.

By discussion similar to that for a local topology, we can prove the lemma for a P-local topology. □

[3]From the definition of the i-th process, the two processes are distinct.
[4]The successors of these processes in the paths are a same process.



(a) A $4k$ sized $2k-1$ node and link colored ring

(b) A $2k$ sized $2k-1$ node and link colored ring

Fig. 5. Example with $k = 3$ and even cycle of size $2k$.

Lemmas 3.1 and 3.2 present sufficient conditions on cycle length for (P-)local topology construction. Lemma 3.5 (for local topology) and Corollary 3.4 (for P-local topology) show that all presented bounds on cycle length in Lemmas 3.1 and 3.2 are tight.

LEMMA 3.3 (EVEN CYCLES). *Given a local view at distance k for each process, a node coloring at distance $2k-1$ and a link coloring at distance $2k-1$, there exists a pair of networks containing an even cycle of size $2k$ and $4k$ where it is impossible to construct a local topology at distance k for every node p in either ring.*

PROOF. Consider a $2k$ sized ring network where processes (numbered for the purpose of this proof from p_1 to p_{2k}) up to distance $2k-1$ apart have different colors, and links (numbered for the purpose of this proof from l_1 to l_{2k}) up to distance $2k-1$ apart have different colors. As there are from Observation 3 at least $2k$ available node colors and at least $2k$ available link colors, each node and each link has a unique color in this network.

Now suppose that there exists one process p in this ring that is able to construct the local topology at distance k. In the $2k$ sized ring, the local topology at distance k includes the full ring. So, there exists a j such that p_j is able to construct the full $2k$ ring. This is exemplified as Figure 5(b) for the particular case of $k = 3$. Then, consider a $4k$ sized ring where processes (numbered from p'_1 to p'_{4k}) and links (numbered from l'_1 to l'_{4k}) are colored as follows:

—for any i in $\{1 \ldots 2k\}$, p'_i and p'_{2k+i} are colored as p_i,
—for any i in $\{1 \ldots 2k\}$, l'_i and l'_{2k+i} are colored as l_i,

This is exemplified as Figure 5(a) for the particular case of $k = 3$. This new $4k$ sized ring is also $2k - 1$-local node and link colored. We can show for any i in $\{1 \ldots 2k\}$ that p'_i and p'_{2k+i} in the $4k$ sized ring can behave exactly in the same way as p_i in the $2k$ sized ring. This implies that p'_j and p'_{2k+j} construct the full $2k$ ring, which is incorrect.

Then, consider the complementary case, where p'_j is able to correctly construct the local topology at distance k in the $4k$ sized ring. Since p_j in the $2k$ sized ring can behave exactly in the same way as p'_j in the $4k$ sized ring. This implies that p_j constructs a local topology at distance k which is a tree, which is incorrect.

So, every process is unable to construct a local topology at distance k in either of the two networks. □

Since the P-local topology of each process is the same as the local topology in networks considered in the above proof, a similar impossibility result holds for the P-local topology.

COROLLARY 3.4. *Given a local view at distance k for each process, a node coloring at distance $2k - 1$ and a link coloring at distance $2k - 1$, there exists a pair of networks containing an even cycle of size $2k$ and $4k$ where it is impossible to construct a P-local topology at distance k for every node p in either ring.*

LEMMA 3.5 (ODD CYCLES). *Given a local view at distance k for each process, a node coloring at distance $2k$ and a link coloring at distance $2k$, there exists a pair of networks containing an odd cycle of size $2k+1$ and $4k+2$ where it is impossible to construct a local topology at distance k for every node p in either ring.*

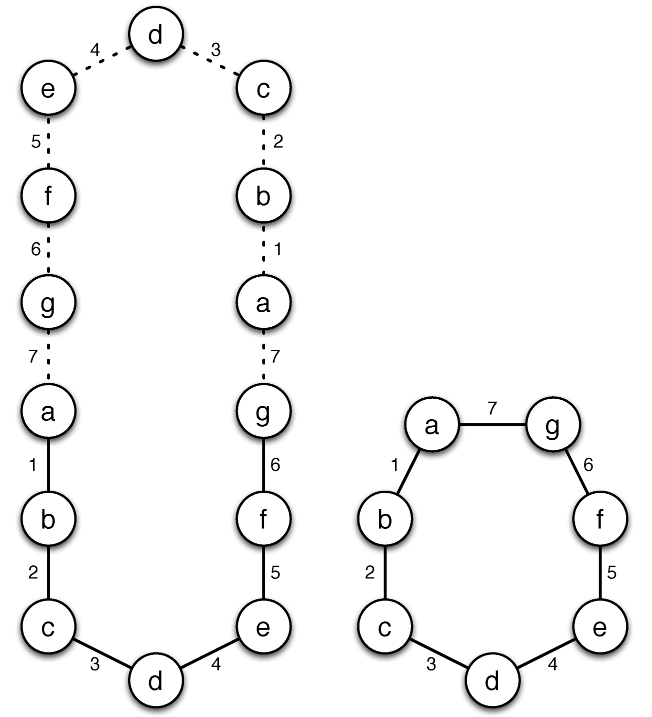
PROOF. Consider a $2k + 1$ sized ring network where processes (numbered for the purpose of this proof from p_1 to p_{2k+1}) up to distance $2k$ apart have different colors, and links (numbered for the purpose of this proof from l_1 to l_{2k+1}) up to distance $2k$ apart have different colors. As there are from Observation 3 at least $2k + 1$ available node colors and at least $2k + 1$ available link colors, each node and each link has a unique color in this network.

Now suppose that there exists one process p in this ring that is able to construct the local topology at distance k. In the $2k + 1$ sized ring, the local topology at distance k includes the full ring. So, there exists a j such that p_j is able to construct the full $2k + 1$ ring. This is exemplified as Figure 6(b) for the particular case of $k = 3$.

Then, consider a $4k + 2$ sized ring where processes (numbered from p'_1 to p'_{4k+2}) and links (numbered from l'_1 to l'_{4k+2}) are colored as follows:

—for any i in $\{1 \ldots 2k + 1\}$, p'_i and p'_{2k+i+1} are colored as p_i,
—for any i in $\{1 \ldots 2k + 1\}$, l'_i and l'_{2k+i+1} are colored as l_i,

This is exemplified as Figure 6(a) for the particular case of $k = 3$. This new $4k + 2$ sized ring is also $2k$-local node and link colored. We can show for any i in $\{1 \ldots 2k + 1\}$ that p'_i and p'_{2k+i+1} in the $4k + 2$ sized ring can behave exactly in the same way as p_i in the $2k + 1$ sized ring. This implies that p'_j and p'_{2k+j+1} construct the full $2k + 1$ ring, which is incorrect.



(a) A $4k+2$ sized $2k$ node and link colored ring

(b) A $2k+1$ sized $2k$ node and link colored ring

Fig. 6. Example with $k = 3$ and odd cycle of size $2k + 1$.

Then, consider the complementary case, where p'_j is able to correctly construct the local topology at distance k in the $4k + 2$ sized ring. Since p_j in the $2k + 1$ sized ring can behave exactly in the same way as p'_j in the $4k + 2$ sized ring. This implies that p_j constructs a local topology at distance k that is a tree, which is incorrect.

So, every process is unable to construct a local topology at distance k in either of the two networks. □

Notice that the similar impossibility result does not hold for the P-local topology. Actually, from Observation 2, the P-local topology and the local view of distance k coincide, and thus, can be obtained from each other.

We now present the main result of the section.

THEOREM 3.6. *Given a local view at distance k for each process, a node coloring at distance* $2k+1$ *(resp.* $2k$*)* or *a link coloring at distance* $2k+1$ *(resp.* $2k$*), it is possible to construct a local (resp. P-local) topology at distance k for every node. The bounds on the coloring distance are tight.*

PROOF. Assume that we wish to obtain the local topology at distance k. Now, this local topology is a subgraph of the global graph G. The cycles of size $2k + 2$ or more cause no problem to the topology discovery from the proof of Lemma 3.1.

Similarly, the cycles of size $2k$ or less cause no problem either from the proof of Lemma 3.2. The tightness of the bounds is presented in Lemma 3.5. The results for the P-view result from Corollary 3.4. □

4. BOOTSTRAPPING TOPOLOGY KNOWLEDGE

In this section, we discuss the possibility to bootstrap topology knowledge in anonymous networks. Bootstrapping topology knowledge is the process of learning local topology at distance $k + 1$ assuming local topology at distance k is known. As proved in Section 3, this is tantamount to requiring that node or link coloring can be bootstrapped, that is, being able to compute a node (link, respectively) coloring at distance $k + 1$ given a node (link respectively) coloring at distance k.

4.1 Negative Results

On the negative side, Theorems 4.1 and 4.2 show that (i) with a distributed daemon, it is impossible to achieve deterministic color bootstrap, even if the whole network topology can be instantaneously obtained, and (ii) with a central daemon, it is impossible to achieve distance m when instantaneous topology knowledge is limited to $m - 1$.

THEOREM 4.1. *Given a m-local node and link coloring, the ability for each process to instantaneously get a colored topology of the whole network, and a distributed daemon, it is impossible to get either a node coloring at distance $m + 1$ or a link coloring at distance $m + 1$ with a deterministic algorithm.*

PROOF. The proof for the node coloring at distance $m + 1$ is implied by the proof of Theorem 5.4 in Sakamoto [2000]. The basic argument is to consider a ring of size $m \times l$, where the node colors repeat along the cycle in the same order. If the schedule is synchronous (which can occur with a distributed scheduler), all nodes that were in the same state had the same view of the system. Since their code is deterministic, they all reach the same new state. This is true for all nodes, so we do not get more asymmetry in the network, and $m + 1$ colors can not be generated.

Now, for the case of $m + 1$-local link coloring. We simply consider the fact that the color of a link is determined by the state of its two incident processes. From the previous argument, pairs of processes (p_j, p_{m+j+1}), for $j \in \{1 \ldots m + 1\}$, remain in the same state infinitely often, so pairs of pairs of processes $((p_j, p_{m+j+1}), (p_{j+1}, p_{m+j+2}))$, for $j \in \{1 \ldots m + 1\}$ remain in the same state. As a result, the colors of links l_j and l_{m+j+1} remain the same forever, so a $m + 1$-local link coloring is never achieved. □

THEOREM 4.2. *Let k and m be two integers such that $m \geq k \geq 2$. Given an m-local node and link coloring, the ability of each process to instantaneously get a colored local topology at distance k, and a central daemon, it is impossible to get either a node coloring at distance $m + 1$ or a link coloring at distance $m + 1$ with a deterministic algorithm.*

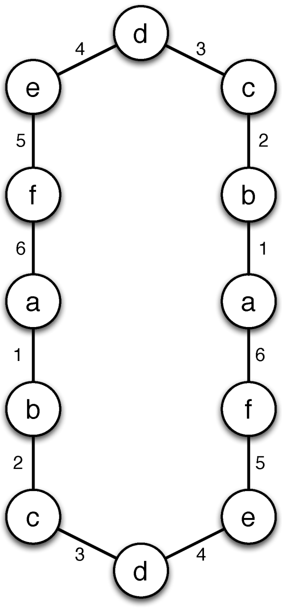


Fig. 7. Example with $m = 5$.

Proof. Consider a $2m + 2$ sized ring whose processes are numbered from p_1 to p_{2m+2} and whose links are numbered from l_1 to l_{2m+2}, with l_i corresponding to the link between p_i and p_{i+1} (if $i < 2m + 2$) or p_1 (if $i = 2m + 2$). Since a m-local node and link coloring is assumed, we have $m + 1$ possible colors for both nodes and links. We consider that for every $i \in \{1 \dots m + 1\}$, processes p_i and p_{i+m+1} share the same color, and that links l_i and l_{i+m+1} share the same color. This case is exemplified in Figure 7 when $m = k = 5$.

As a result, for any $k \leq m$, p_i and p_{i+m+1} share the same local topology at distance k. Now, the central scheduler indefinitely activates the processes as follows:

—for $j \in \{1..m + 1\}$, p_j then p_{j+m+1} are successively activated.

Now consider a starting configuration where all pairs of processes (p_j, p_{j+m+1}), for $j \in \{1 \dots m + 1\}$, are in the same state. Then, after all processes have been activated exactly once by the central scheduler, all pairs of processes (p_j, p_{j+m+1}), for $j \in \{1 \dots m + 1\}$ are *still* in the same state, because they run a deterministic algorithm that is fed with the same input. Also, this process may repeat so that all pairs of processes (p_j, p_{j+m+1}), for $j \in \{1 \dots m+1\}$ remain in the same state forever. As a result, no symmetry breaking at distance more than m can occur, and nodes at distance $m + 1$ from each other keep the same color, so a $m + 1$-local node coloring is never achieved.

Now, for the case of $m + 1$-local link coloring. We simply consider the fact that the color of a link is determined by the state of its two incident processes. From the previous argument, pairs of processes (p_j, p_{j+m+1}), for $j \in \{1 \dots m + 1\}$, remain in the same state infinitely often, so pairs of pairs

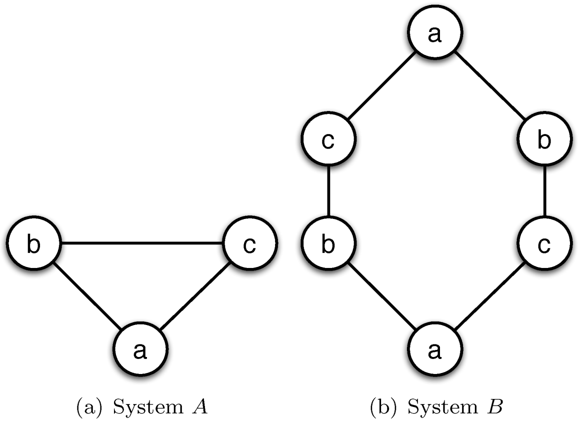


Fig. 8. Impossibility of silent self-stabilizing 2-coloring.

of processes $((p_j, p_{j+m+1}), (p_{j+1}, p_{j+m+2}))$, for $j \in \{1 \ldots m+1\}$ remain in the same state. As a result, the colors of links l_j and l_{j+m+1} remain the same forever, so a $m+1$ local link coloring is never achieved. □

THEOREM 4.3. *For $k \geq 3$, there exists no silent (probabilistic or deterministic) self-stabilizing k-coloring algorithms for anonymous networks where topology is unknown.*

PROOF. For $k = 3$ consider the two systems A and B that are presented in Figure 8. Suppose there exists a silent self-stabilizing 3-coloring algorithm that can perform on unknown graphs. Let this algorithm perform on system A and terminate. The final states of the processes (in the terminal configuration) is depicted in Figure 8(a). Now construct system B with the same states as depicted in Figure 8(b). The two nodes with state a must have the same color, since they have the same states, and their neighbors also have the same state. However, those two nodes are at distance 3 from one another. As a result, the coloring is not a proper 3-coloring. Since all nodes are in a terminal state, the system remains in the same configuration forever. As a result, the algorithm is not a self-stabilizing 3-coloring algorithm for unknown graphs. Hence a contradiction.

The same argument can be used for the case of $k > 3$. Hence the theorem. □

4.2 Positive Results

On the positive side, Theorems 4.4 and 4.6 show that (i) under the k-central daemon, deterministic bootstrap of colors up to distance k is possible provided that k-local topology can be instantaneously obtained, and (ii) under the distributed daemon, probabilistic bootstrap is possible for any range.

4.2.1 *A Condition for Deterministic Self-Stabilizing Bootstrap.* We first prove that under a k-local central daemon, color bootstrapping is essentially feasible. The case of bootstrapping is presented in Section 4.2.3.

Algorithm 4.1. View construction (distance k)

```
inputs:
  $k$: integer parameter
  $V_p^0$: arbitrary typed variable
variables:
  $V_p^k$: view of $p$ at distance $k$
    //$V_p^{k-1}|j$ denotes the part of $V_p^k$ related to neighbor $j$
    //$V_p^0$ denotes the monitored variable of $p$ and is considered an input
actions::
  { If a node has an update on its local topology, update the current view }
  $\mathcal{R}_1 :: k > 0 \wedge \exists j \in N_p, V_j^{k-1} \neq V_p^{k-1}|j$
    $\rightarrow \forall j \in N_p, V_p^{k-1}|j := V_j^{k-1}$
```

THEOREM 4.4. *Let k, m be two integers such that $k > m \geq 2$. Given the ability of each process to instantaneously get a colored local topology at distance k, and a k-local central daemon, there exists a deterministic self-stabilizing algorithm that permits to get a $m + 1$ local node coloring.*

PROOF. In fact, we prove that knowing the local topology at distance k permits to node and link color the network so that the coloring is k-local. Each process p knows the local topology at distance k, so each process p is able to construct the graph G^k. By the k-local central daemon hypothesis, no two neighbors in G^k are activated at the same time. Then each process executes, for example, Algorithm [Gradinariu and Tixeuil 2000] (that performs under the 1-local central daemon) to node color G^k. Of course, a 1-local node coloring of G^k is a k-local node coloring in G. □

4.2.2 *Probabilistic Self-Stabilizing Bootstrap.* The positive results that we provide in this section are constructive, and we present probabilistic self-stabilizing algorithms that permit to color nodes at some arbitrary distance.

The first component is described as Algorithm 4.1. The dynamics of the algorithm essentialy consists in propagating local variables of each node to neighboring nodes up to distance k. This algorithm only assumes that a node is able to locally distinguish its neighbors, so that it can update parts of its own view accordingly to updates provided by its neighbors. This scheme is essentially the same as the one used in Dolev and Herman [1997] (with the notable exception that here nodes do not have unique identifiers) and Boldi and Vigna [2002].

The overall dynamics of Algorithm 4.1 are as follows. When $k = 0$, a node p simply outputs its monitored variable V_p^0 to its neighbors. When $k = 1$, a node p verifies (Rule $\mathcal{R}_1$) whether a neighbor j has inconsistent monitored variable (i.e., $V_j^0 \neq V_p^0|j$), and updates the cached value $V_p^0 \mid j$ with the value that is effectively held at j, that is V_j^0. For any $k > 1$, the same process is repeated (Rule $\mathcal{R}_1$) in order that eventually all views are consistent.

Algorithm 4.2. Probabilistic unstable node coloring (distance k)

```
inputs:
  $V_p^k$: view at distance $k$
variables:
  $c_p$: color of node $p$ (in domain $\Gamma$)
functions:
  $colors(V)$: returns the set of colors contained in view $V$
  $random(S)$: returns a random element of set $S$
actions::
  { If a conflict is detected, randomly choose a fresh color }
  $\mathcal{R}_1 :: \exists j \in V_p^k, c_j = c_p$
    $\rightarrow c_p := random\left(\Gamma \setminus colors(V_p^k)\right)$
```

Lemma 4.5. *Algorithm* 4.1 *is self-stabilizing for the distance k view construction and stabilizes in k rounds.*

Proof. The proof is by induction on k. For $k = 0$, all monitored variables are inputs and are thus correct. Now suppose that Algorithm 4.1 constructs a distance $k-1$ view that stabilizes in $k-1$ rounds. After one more round, every node executes (if enabled) Rule $\mathcal{R}_1$ and constructs a view at distance k from the view at distance $k-1$ of each of its neighbors. Hence the result. □

Since a view at distance k for a limited set of variables can be achieved using Algorithm 4.1, a first attempt for coloring the nodes would be to run Algorithm 4.2 in parallel (using the fair composition of Gouda and Herman [1991]). The algorithm is the same as the neighborhood coloring protocol of [Herman and Tixeuil 2004; Mitton et al. 2005, 2006], with the notable exception that here the local topology is not known, and the local view is used instead. Informally, the algorithm runs as follows: each node checks whether its color conflicts with another color visible in its view at distance k (excluding itself). If a conflict is found, the node randomly draws a new color, in the set of available colors (i.e., the set of colors that do not appear in its view at distance k).

Of course, it is still possible that a node actually detects a conflict with itself if it gets a view at distance k in a cycle of size exactly k. In that case, Algorithm 4.2 would result in having every such node changing colors forever; the color constraint would not be broken (neighbors would have distinct colors), but some colors would keep changing forever. To resolve this issue, we present a different approach: a node p detecting a conflict uses a *probe* mechanism. The node p first extracts the path (using local labeling on edges stored in its view) towards and backwards from the conflicting node. Then p draws a big random number, stores it in a $probe_p$ variable, and sends the random number along the path. When the random variable arrives at the destination q, q checks whether its last calculated $probe_q$ matches the received one. If no, a "fail" response is sent using the backward path. If yes, a "success" response is sent using the backward path. Now, when p receives a "fail" response, it randomly chooses a new color, as in the original algorithm. If p receives a "success," it draws a new random number and sends a new probe. This probing mechanism is described in more detail in the following.

A *probe* is a 4-tuple $(n_i, p_d, f_{i,d}, f_{d,i})$, where n_i denotes the random number that was drawn by the initiator of the probe, p_d denotes the current path toward the destination of the probe (as a list of local port numbers), $f_{i,d}$ denotes the full path from the initiator i to the destination d (as a list of local port numbers), $f_{d,i}$ denotes the full path from the destination d back to the initiator i (as a list of local port numbers). A *response probe* is a 5-tuple $(n_i, p_i, f_{i,d}, f_{d,i}, b_d)$, where n_i denotes the random number that was drawn by the initiator of the probe, p_i denotes the current path toward the initiator of the probe—and destination of the probe response (as a list of local port numbers), $f_{i,d}$ denotes the full path from the initiator i to the destination d of the initial probe (as a list of local port numbers), $f_{d,i}$ denotes the full path from the destination d back to the initiator i of the initial probe (as a list of local port numbers), and b_d denotes the destination's response (i.e. b_d is true if the destination has the same random number as the initiator). In order to manage lists (of local port numbers), nodes are allowed to use *head*(l) (that returns the first item in list l), *tail* (l) (that returns list l minus its first element). The constant () denotes the empty list.

In order to deal with multiple probes, we assume that each register is able to hold several such probes and probe responses. In the following, we use the notation $r \cup = p$ to add the probe message p (or response) to register r. Similarly, we use the notation $r \setminus = p$ to remove the probe message p (or response) from the register r. The dynamics of the algorithm presented as Algorithm 4.3 are as follows:

(1) When a node notices that at least one node in its view has the same color as his own color, it initiates a probe to each such node (Rule $\mathcal{R}_1$),
(2) When a node that previously initiated k probes has received k responses, all stating that the color was the same at the destination, the node randomly selects a new probe number (Rule $\mathcal{R}_2$),
(3) When a node that previously initiated k probes has received k responses, at least one stating that the color was *not* the same at the destination, the node randomly selects a new color not already seen in its view (Rule $\mathcal{R}_3$),
(4) When a node notices that it should forward a probe, it does so (Rule $\mathcal{R}_4$),
(5) When a node notices that it is the destination of a probe, it sends a probe response (Rule $\mathcal{R}_5$),
(6) When a node notices that it should forward a probe response, it does so (Rule $\mathcal{R}_6$),
(7) Whenever a node notices that some cleanup is necessary, it does so (Rule $\mathcal{R}_7$).

We assume that each node p maintains two local variables that are accessible to all neighbors: c_p denotes the color of node p (in domain Γ), and $probe_p$ denotes the current integer probe number of p. Also, each node p is able to compute the following local functions:

—*to*(j, V) returns the path from p to a particular node j in view V.

—*from*(j, V): returns the path from a particular node j to p in view V.

—*probeSent*(j, V) returns the contents of the probe the current node p should send to j in view V. Formally:

$$probeSent(j) :: (probe_p, tail(to(j, V)), to(j, V), from(j, V))$$

—*probeForwarded*$(n_i, p_d, f_{i,d}, f_{d,i})$ returns the contents of the probe that should have been forwarded if probe $(n_i, p_d, f_{i,d}, f_{d,i})$ was received. Formally:

$$probeForwarded(n_i, p_d, f_{i,d}, f_{d,i}) :: (n_i, tail(p_d), f_{i,d}, f_{d,i})$$

—*probeResponseSent*$(n_i, (), f_{i,d}, f_{d,i})$ returns the contents of the probe response that should be sent back to the sender if probe $(n_i, (), f_{i,d}, f_{d,i})$ was received. Formally:

$$probeResponseSent(n_i, (), f_{i,d}, f_{d,i}) :: (n_i, tail(f_{d,i}), f_{i,d}, f_{d,i}, probe_p = n_i)$$

—*probeResponseForwarded*$(n_i, p_i, f_{i,d}, f_{d,i}, b_d)$ returns the contents of the probe response that should have been forwarded upon receiving probe response $(n_i, p_i, f_{i,d}, f_{d,i}, b_d)$. Formally:

$$probeResponseForwarded(n_i, p_i, f_{i,d}, f_{d,i}, b_d) :: (n_i, tail(p_i), f_{i,d}, f_{d,i}, b_d)$$

—*probeResponseReceived*(j, x) returns the probe response contents that should have been received by an initiator sending a probe to node j in view V and expecting response x. Formally:

$$probeResponseReceived(j, V, x) :: (probe_p, to(j, V), (), to(j, V), from(j, V), x)$$

—*cleanupNeeded* returns true if and only if some cleanup is necessary. The $f_{i.d}$ and $f_{d,i}$ parts of each probe and each probe response are used to detect that: (i) every probe in some output register is consistent with respect to path information, (ii) every probe in some output register corresponds to some initiated probe, or results from another probe in some input register, (iii) every probe response in some output register corresponds either to some initiated probe response, or results from some probe response in some input register, and (iv) every probe response in some output register corresponds to a matching probe (a response may only be initiated by a probe).

—*cleanup* removes any probe that is not needed any more. That is, any discrepancy in some output register found by *cleanupNeeded* should be erased.

THEOREM 4.6. *Let k, m be two integers such that $k > m \geq 2$. Given an m-local node coloring, the ability of each process to get a colored local view at distance k, and a distributed daemon, there exists a probabilistic self-stabilizing algorithm that permits to get a $m + 1$-local node coloring in $O(k)$ rounds.*

PROOF. The proof is by providing such a self-stabilizing probabilistic algorithm. The core components of this protocol are described as Algorithms 4.1 and 4.3.

By Lemma 4.5, Algorithm 4.1 is able to produce a view at distance k, for any arbitrary k, in a self-stabilizing way, within k rounds.

Second, for Algorithm 4.3, we first observe that the cleanup rule (Rule $\mathcal{R}_7$) ensures that after at most k rounds, there is no pending fake probe or fake probe response. Then, initiated probes are forwarded up to their destination, and

Algorithm 4.3. Probabilistic stable node coloring with probe (distance k)

```
variables:
  $c_p$: color of node $p$ (in domain $\Gamma$)
  $probe_p$: probe integer of node $p$ (in domain $I$)
functions:
  $colors(V)$: returns the set of colors contained in view $V$
  $random(S)$: returns a random element of set $S$
actions::
  { Initiate new probe }
  $\mathcal{R}_1 :: \neg\, cleanupNeeded$
  $\wedge\ \exists j \in \cup_{i \in N_p} V_p^k|i,\ c_j = c_p \wedge probeSent(j,V) \notin r_{p,head(to(j,V))}$
    $\rightarrow r_{i,head(to(j,V))} \cup= probeSent(j,V)$
  { All probe responses returned and match, try new probe }
  $\mathcal{R}_2 :: \neg\, cleanupNeeded$
  $\wedge\ \exists j \in \cup_{i \in N_p} V_p^k|i,\ c_j = c_p \wedge probeSent(j,V) \notin r_{p,head(to(j,V))}$
  $\wedge\ \forall l \in \cup_{i \in N_p} V_p^k|i,\ c_l = c_p, probeResponseReceived(l, true) \in r_{head(to(l,V)),p}$
    $\rightarrow probe_p := random(\{1, \ldots N\});$
  { All probe responses returned and at least one mismatch, try new color }
  $\mathcal{R}_3 :: \neg\, cleanupNeeded$
  $\wedge\ \exists j \in \cup_{i \in N_p} V_p^k|i,\ c_j = c_p \wedge probeSent(j,V) \notin r_{p,head(to(j,V))}$
  $\wedge\ \forall l \in \cup_{i \in N_p} V_p^k|i,\ c_l = c_p, probeResponseReceived(l, x) \in r_{head(to(l,V)),p}$
  $\wedge\ \exists v \in \cup_{i \in N_p} V_p^k|i,\ c_v = c_p, probeResponseReceived(v, false) \in r_{head(to(v,V)),p}$
    $\rightarrow c_p := random\left(\Gamma \setminus colors(\cup_{i \in N_p} V_p^k|i)\right)$
  { Forward Probe }
  $\mathcal{R}_4 :: \neg\, cleanupNeeded$
  $\wedge\ \exists j \in N_p, (n_i, p_d, f_{i,d}, f_{d,i}) \in r_{j,p} \wedge p_d \neq ()$
  $\wedge\ probeForwarded(n_i, p_d, f_{i,d}, f_{d,i}) \notin r_{p,head(p_d)}$
    $\rightarrow r_{j,head(p_d)} \cup= probeForwarded(n_i, p_d, f_{i,d}, f_{d,i})$
  { Initiate new probe response }
  $\mathcal{R}_5 :: \neg\, cleanupNeeded$
  $\wedge\ \exists j \in N_p, (n_i, (), f_{i,d}, f_{d,i}) \in r_{j,p}$
  $\wedge\ probeResponseSent(n_i, (), f_{i,d}, f_{d,i}) \notin r_{p,head(f_{d,i})}$
    $\rightarrow r_{j,head(f_{d,i})} \cup= probeResponseSent(n_i, (), f_{i,d}, f_{d,i})$
  { Forward probe response }
  $\mathcal{R}_6 :: \neg\, cleanupNeeded$
  $\wedge\ \exists j \in N_p, (n_i, p_d, f_{i,d}, f_{d,i}, b_d) \in r_{j,p} \wedge p_d \neq ()$
  $\wedge\ probeResponseForwarded(n_i, p_i, f_{i,d}, f_{d,i}, b_d) \notin r_{p,head(p_i)}$
    $\rightarrow r_{j,head(p_i)} \cup= probeResponseForwarded(n_i, p_i, f_{i,d}, f_{d,i}, b_d)$
  { Perform cleanup }
  $\mathcal{R}_7 :: cleanupNeeded$
    $\rightarrow cleanup$
```

responses backtrack to the source, with either a positive or a negative feedback. Moreover, a probe reaches its destination within at most k rounds, and returns to its destination within at most k rounds. If the feedback is negative, then the node that initiated the probe chooses a new color *not* in the neighborhood at

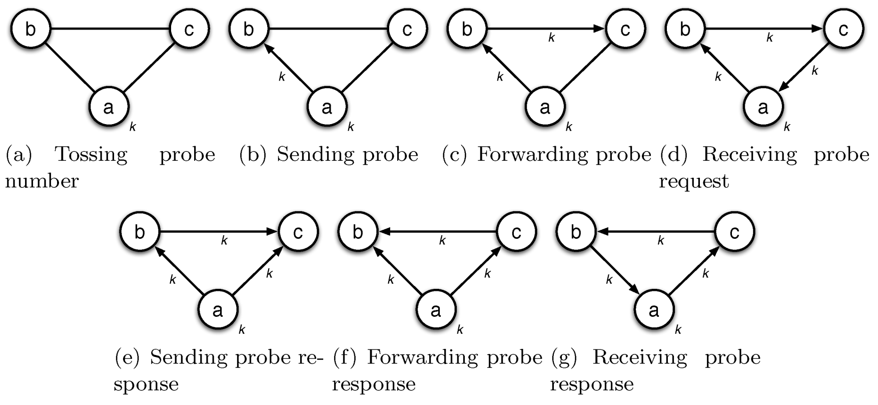

b
c
a
k
(a) Tossing probe number
(b) Sending probe
(c) Forwarding probe
(d) Receiving probe request
(e) Sending probe response
(f) Forwarding probe response
(g) Receiving probe response

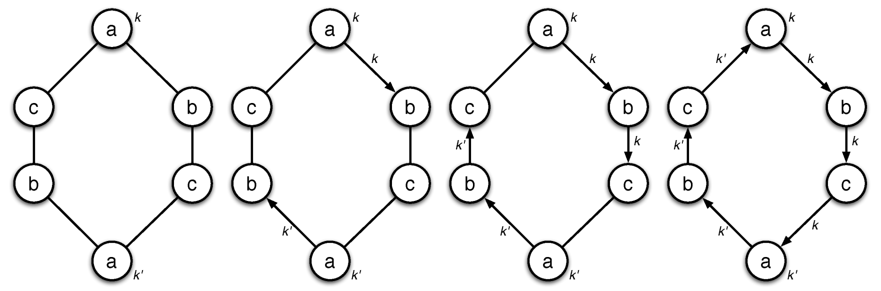
Fig. 9. Probing oneself always returns the same probe number.

distance k. If the feedback is positive, the initiator sends a new probe with a different random number. Now, if the initiator and the probed nodes are the same, the same random probe number will be used forever, and the color of the initiator will not change (see Figure 9). If the two nodes are indeed different, then there is a strictly positive probability of $1 - \frac{1}{|I|^2}$ that two different nodes draw different probe numbers, so within expected $\frac{1}{1-\frac{1}{|I|^2}}$ random number choices, each node changes its color only when it implies a conflict with a different node (see Figure 10). Now, if two nodes with the same color choose a new random color and there are sufficiently many colors in Γ to properly color nodes up to distance k, each trial gives a node a strictly positive probability to draw a unique color (see Herman and Tixeuil [2004] for details), and the expected number of such trials is also bounded by a constant for each node.

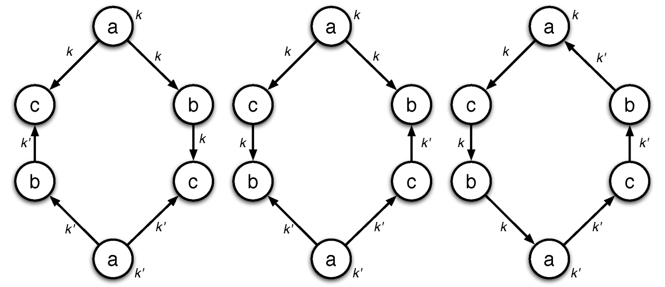
After the initial $2k$ rounds to get rid of the initial discrepancies in views and fake probes and responses, every node sends an expected constant number of probes (and responses) before getting a unique color at distance k. Each probe requiring $2k$ rounds for a round trip. Overall, Algorithm 4.3 has a local self-stabilizing time of $O(k)$ rounds, before providing coloring at distance k, for any arbitrary k. □

4.2.3 *From Node Coloring to Link Coloring.* In this section, we present a link coloring algorithm (Algorithm 4.4) assuming a node coloring is already available. The protocol is then extended to provide a distance-2 link coloring (Algorithm 4.5).

Distance 1 node coloring (Algorithm 4.3) is used to define *domination* between neighboring nodes as follows: the node whose color is larger dominates the other node. In the link coloring protocol Algorithm 4.4, the dominating node is responsible for setting the color of a particular edge (since the network is node colored, such a dominator always exists). The algorithm then runs as follows. First, each node p collects in a shared variable $report_p$ the colors of all adjacent edges (Rule $\mathcal{R}_1$). Second, a node p detecting a conflict between an edge it dominates (to another node j) and either:



(a) Tossing probe number (b) Sending probe (c) Forwarding probe (d) Receiving probe request



(e) Sending probe response (f) Forwarding probe response (g) Receiving probe response

Fig. 10. Probing a different node returns a different probe number whenever $k \neq k'$.

(1) another adjacent edge of p (Rule $\mathcal{R}_2$), or
(2) another adjacent edge of j (Rule $\mathcal{R}_3$)

randomly chooses a new color for this particular link it dominates among the set of available link colors.

LEMMA 4.7. *Assuming a distance-*1 *node coloring, Algorithm* 4.4 *is probabilistically self-stabilizing to a distance-*1 *link coloring.*

PROOF. Algorithm 4.4 assumes that a node coloring at distance 1 is achieved, and provides a distance 1 link coloring of the same graph. Self-stabilization of the protocol can be proved as follows. Assume there is a conflict of link colors (without loss of generality, assume that those links are (k, i) and (i, j)), then three cases may occur:

(1) i dominates both k and j, then i randomly chooses a new color for one of the links, and the conflict disappears,
(2) k dominates i but i dominates j, then either k (by Rule $\mathcal{R}_3$), i (by Rule $\mathcal{R}_2$), or both randomly choose a new color. In the first two cases, the conflict is solved. In the last case, there is a strictly positive probability that the conflict is solved (if k and i draw different colors).
(3) i is dominated by both k and j, then either k (by Rule $\mathcal{R}_3$), j (by Rule $\mathcal{R}_3$), or both randomly choose a new color. In the first two cases, the conflict is solved. In the last case, there is a strictly positive probability that the conflict is solved (if k and j draw different colors).

Algorithm 4.4. Randomized link coloring (distance 1)

```
variables:
    $l_{p\to j}$: color of link $(p, j)$, with $c_p > c_j$ (in domain $\Gamma'$)
    $report_p$: multiset of colors
functions:
    $conflict(M, l)$: returns true if color $l$ is present more than once in color multiset $M$
actions::
    { Build multiset of neighboring link colors }
    $\mathcal{R}_1 :: report_p \neq \left(\cup_{k\in N_p, c_p > c_k} l_{p\to k}\right) \cup \left(\cup_{k'\in N_p, c_{k'} > c_p} l_{k'\to p}\right)$
        $\to report_p := \left(\cup_{k\in N_p, c_p > c_k} l_{p\to k}\right) \cup \left(\cup_{k'\in N_p, c_{k'} > c_p} l_{k'\to p}\right)$
    { Take care of local conflict on dominated link }
    $\mathcal{R}_2 :: \exists j \in N_p, conflict(report_p, l_{p\to j})$
        $\to l_{p\to j} := random(\Gamma' \setminus \{report_p \cup report_j\})$
    { Take care of distance 1 conflict on dominated link }
    $\mathcal{R}_3 :: \exists j \in N_p, conflict(report_j, l_{p\to j})$
        $\to l_{p\to j} := random(\Gamma' \setminus \{report_p \cup report_j\})$
```

So at each step, there is at least a positive constant probability to reduce the number of conflicting colors for links. As a result, Algorithm 4.4 is a probabilistic self-stabilizing algorithm for distance 1 link coloring assuming distance 1 node coloring. □

The distance-2 link coloring algorithm (Algorithm 4.5) uses the same basic scheme, except that a third rule is added (and implemented by $\mathcal{R}_4$, $\mathcal{R}_5$, and $\mathcal{R}_6$ in Algorithm 4.5): if a node finds out that a link it dominates is in conflict with *any* of its neighbors' adjacent nodes, it randomly chooses a new color. Of course, the domain of edge colors Γ'' has to be larger than the number of distance-1 edge colors Γ'. Since the technique is essentially greedy, we assume that Γ' contains at least $2\Delta - 1$ colors (where Δ denotes the maximum degree of the graph), and that Γ'' contains at least Δ^2 colors.

LEMMA 4.8. *Assuming a distance-*1 *node coloring, Algorithm* 4.5 *is probabilistically self-stabilizing to a distance-*2 *link coloring.*

PROOF. Algorithm 4.5 assumes that a node coloring at distance 1 is achieved, and provides a distance-2 link coloring of the same graph. Self-stabilization of the protocol can be proved as follows.

First, the first three rules guarantee that link coloring at distance 1 is eventually achieved, since the new rules are the same except that the color domain is larger. Now, assume that there is a conflict of link colors at distance 2 (without loss of generality, assume that those links are (i, j) and (k, l), with j and k being neighbors. Four cases may occur:

(1) i dominates j and k dominates l, then either k notices through $report_j$ that there is a conflict with a link it dominates, and executes rule $\mathcal{R}_4$ and randomly chooses a new color, or j reports the conflict into the $conflicts_j$ variables (by Rule $\mathcal{R}_5$), and both k and i are now aware of the conflict, and can execute Rule $\mathcal{R}_6$ and choose a random new color.

Algorithm 4.5. Randomized link coloring (distance 2)

```
variables:
    $l_{p\to j}$: color of link $(p, j)$, with $c_p > c_j$ (in domain $\Gamma''$)
    $report_p$: multiset of colors
    $conflicts_p$: set of colors
functions:
    $conflict(M, l)$: returns true if color $l$ is present more than once in color multiset $M$
actions::
    { Build multiset of neighboring link colors }
    $\mathcal{R}_1 :: report_p \neq \left(\cup_{k\in N_p, c_p>c_k} l_{p\to k}\right) \cup \left(\cup_{k'\in N_p, c_{k'}>c_p} l_{k'\to p}\right)$
        $\to report_p := \left(\cup_{k\in N_p, c_p>c_k} l_{p\to k}\right) \cup \left(\cup_{k'\in N_p, c_{k'}>c_p} l_{k'\to p}\right)$
    { Take care of local conflict on dominated link }
    $\mathcal{R}_2 :: \exists j \in N_p, conflict(report_p, l_{p\to j})$
        $\to l_{p\to j} := random(\Gamma'' \setminus \{report_p \cup (\cup_{k\in N_p} report_k)\})$
    { Take care of distance 1 conflict on dominated link }
    $\mathcal{R}_3 :: \exists j \in N_p, conflict(report_j, l_{p\to j})$
        $\to l_{p\to j} := random(\Gamma'' \setminus \{report_p \cup (\cup_{k\in N_p} report_k)\})$
    { Take care of distance 2 conflict on dominated link }
    $\mathcal{R}_4 :: \exists j, j' \in N_p, conflict(report_j, l_{p\to j'})$
        $\to l_{p\to j'} := random(\Gamma'' \setminus \{report_p \cup (\cup_{k\in N_p} report_k)\})$
    { Build list of conflicting colors at distance 2 }
    $\mathcal{R}_5 :: conflicts_p \neq \{l_{j'\to p} | \exists j, j' \in N_p, conflict(report_j, l_{j'\to p})\}$
        $\to conflicts_p := \{l_{j'\to p} | \exists j, j' \in N_p, conflict(report_j, l_{j'\to p})\}$
    { Take care of distance 2 reported conflict on dominated link }
    $\mathcal{R}_6 :: \exists j \in N_p, conflict(conflicts_j, l_{p\to j})$
        $\to l_{p\to j} := random(\Gamma'' \setminus \{report_p \cup (\cup_{k\in N_p} report_k)\})$
```

(2) i dominates j and k does not dominate l, then both j and k can execute Rule $\mathcal{R}_5$ and report the conflict, then both i and l are aware of the conflict, and can execute Rule $\mathcal{R}_6$ and choose a random new color.
(3) i does not dominate j and k dominates l, then either j or k may execute Rule $\mathcal{R}_4$ and randomly choose a new color for the conflicting links.
(4) i does not dominate j and k does not dominates l, then this case is symmetric to case number 1.

So at each step, there is at least a positive constant probability to reduce the number of conflicting colors at distance 2 for links. As a result, Algorithm 4.5 is a probabilistic self-stabilizing algorithm for distance 2 link coloring assuming distance 1 node coloring. □

THEOREM 4.9. *Let k, m be two integers such that $k > m \geq 2$. Given an m-local node coloring or an m-local link coloring, the ability of each process to get a colored local view at distance k, and a distributed daemon, there exists a probabilistic self-stabilizing algorithm that permits to get an $m + 1$-local link coloring.*

Proof. The proof technique is similar to that of Theorem 4.6, except that we need to identify a particular edge instead of a particular node. Instead of sending a single probe to identify the node, we send two probes to both end nodes of the edge. Each of these probes collects information about the two adjacent nodes. Now, if both probes return with the same two numbers, the edge is assumed to be the same, and is identified. If probes return with different numbers, the edges must be different, and thus are to be recolored. □

5. CONCLUSION

We provided evidence that graph coloring (either node or link) is related to topology knowledge as $2k + 1$ coloring is necessary and sufficient for topology knowledge up to distance k (if outgoing edges are necessary). Also, we proved that deterministic algorithms are of no practical help for coloring at distance k, as they require hypotheses that are either unrealistic (topology knowledge at distance k, which is strictly stronger than distance k coloring, the output of the algorithm), or have no known solutions in anonymous networks (to our knowledge, the only deterministic solution for implementing a k-central scheduler was presented in Danturi et al. [2006] and required unique node identifiers). The probabilistic solution we provided practically solves the whole problem: the network is anonymous, topology is not known, and the scheduling daemon is distributed. The distance one and two probabilistic self-stabilizing link coloring algorithms that we presented as a part of the solution may be of particular interest, as these are the first self-stabilizing solutions to this problem that perform under the distributed scheduler.

REFERENCES

Beauquier, J., Delaët, S., Dolev, S., and Tixeuil, S. 2007. Transient fault detectors. *Distrib. Comput. 20,* 1, 39–51.

Beauquier, J., Gradinariu, M., and Johnen, C. 1999. Memory space requirements for self-stabilizing leader election protocols. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing (PODC'99)*. 199–208.

Boldi, P. and Vigna, S. 2002. Universal dynamic synchronous self-stabilization. *Distrib. Comput. 15,* 3, 137–153.

Danturi, P., Nesterenko, M., and Tixeuil, S. 2006. Self-stabilizing philosophers with generic conflicts. In *proceedings of the 8th International Symposium on Stebilization, Safety, and Security on Distributed Systems (SSS'06)*, A. K. Datta and M. Gradinariu, Eds. Lecture Notes in Computer Science, vol. 4280. Springer-Verlag, 214–230.

Delaët, S. and Tixeuil, S. 2002. Tolerating transient and intermittent failures. *J. Parall. Distrib. Comput. 62,* 5, 961–981.

Dolev, S. 2000. *Self-Stabilization*. MIT Press.

Dolev, S., Gouda, M. G., and Schneider, M. 1999. Memory requirements for silent stabilization. *Acta Inf. 36*, 6, 447–462.

Dolev, S. and Herman, T. 1997. Superstabilizing protocols for dynamic distributed systems. *Chicago J. Theor. Comput. Sci.*

Fraigniaud, P., Pelc, A., Peleg, D., and Perennes, S. 2001. Assigning labels in an unknown anonymous network with a leader. *Distrib. Comput. 14*, 3, 163–183.

Gouda, M. G. and Herman, T. 1991. Adaptive programming. *IEEE Trans. Softw. Engin. 17*, 9, 911–921.

Gradinariu, M. and Tixeuil, S. 2000. Self-stabilizing vertex coloring of arbitrary graphs. In *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS'00)*, 55–70.

Herman, T. and Tixeuil, S. 2004. A distributed TDMA slot assignment algorithm for wireless sensor networks. In *Proceedings of the 1st Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors'04)*. Lecture Notes in Computer Science, vol. 3121, Springer-Verlag, 45–58.

Hill, J. L. and Culler, D. E. 2002. Mica: A wireless platform for deeply embedded networks. *IEEE Micro 22*, 6, 12–24.

Masuzawa, T. 1995. A fault-tolerant and self-stabilizing protocol for the topology problem. In *Proceedings of the 2nd Workshop on Self-Stabilizing Systems*, 1.1–1.15.

Mitton, N., Fleury, E., Guérin-Lassous, I., Séricola, B., and Tixeuil, S. 2006. On fast randomized colorings in sensor networks. In *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS'06)*. IEEE Press, 31–38.

Mitton, N., Fleury, E., Guérin-Lassous, I., and Tixeuil, S. 2005. Self-stabilization in self-organized wireless multihop networks. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (WWAN'05)*. IEEE Press, 909–915.

Moy, J. T. 1998. *OSPF: Anatomy of an Internet Routing Protocol*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

Perlman, R. 2000. *Interconnexion Networks*. Addison-Wesley.

Sakamoto, N. 2000. Structure of initial conditions for distributed algorithms. *IEICE Trans. Inform. Syst. E83-D*, 12, 2029–2038.

Spinelli, J. and Gallager, R. 1989. Event driven topology broadcast without sequence numbers. *IEEE Trans. Comm. 37*, 468–474.

Yamashita, M. and Kameda, T. 1999. Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Trans. Paral. Distrib. Syst. 10*, 9, 878–887.

Received March 2007; revised September 2008; accepted September 2008