Self-Stabilizing Deterministic Time Division Multiple Access for Sensor Networks*

Mahesh Arumugam and Sandeep S. Kulkarni

Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University, East Lansing MI 48824
Email: {arumugam,sandeep}@cse.msu.edu
Web: http://www.cse.msu.edu/~{arumugam,sandeep}

Abstract

An algorithm for time division multiple access (TDMA) is found to be applicable in converting existing distributed algorithms into a model that is consistent with sensor networks. Such a TDMA service needs to be self-stabilizing so that in the event of corruption of assigned slots and clock drift, it recovers to states from where TDMA slots are consistent. Previous self-stabilizing solutions for TDMA are either randomized or assume that the topology is known upfront and cannot change. Thus, the question of feasibility of self-stabilizing deterministic TDMA algorithm where the topology is unknown remains open.

In this paper, we present a self-stabilizing deterministic algorithm for TDMA in networks where a sensor is only aware of its neighbors. To our knowledge, this is the first such algorithm that achieves these properties. Moreover, this is the first algorithm that demonstrates the feasibility of stabilization-preserving deterministic transformation of a program in shared-memory model on an arbitrary topology into a program that is consistent with the sensor network model.

Keywords: self-stabilization, time division multiple access, deterministic distance 2 coloring, write-all-with-collision model, sensor networks

*A preliminary version of this paper appears in International Conference on Distributed Computing and Internet Technology, 2005.

Phone: +1-517-353-2387; Fax: +1-517-432-1061

This work was partially sponsored by NSF CAREER CCR-0092724, DARPA Grant OSURS01-C-1901, ONR Grant N00014-01-1-0744, NSF Equipment Grant EIA-0130724, and a grant from Michigan State University.

1 Introduction

The ability to write programs in an abstract model and then translate them into a concrete model is crucial in distributed computing. This ability permits one to write abstract programs where several low level issues such as communication and race conditions among different processes in a distributed system can be ignored. Also, since the abstract program omits these details, it is possible to thoroughly verify it by using techniques such as model checking and/or theorem proving. Now, if we want to utilize the verification of the abstract program to deduce the verification of the concrete program then the transformation from abstract program to concrete program must preserve those properties.

For this reason, the problem of transformation from abstract programs to concrete programs has been extensively considered in the literature [1–6]. These transformations have also focused on preserving the self-stabilization [7,8] property of the original program. The property of self-stabilization refers to the ability of a system to recover from an arbitrary state to a state from where the computation proceeds in accordance with its specification. Since self-stabilization ensures that in-spite of unexpected (transient) faults, the system will recover to legitimate states, it is highly desirable for distributed computing.

Unfortunately, the results from [1–6] cannot be applied to derive concrete programs for a sensor network, as the underlying model of computation in sensor networks is *write all with collision* (WAC) model [9]. In this model, the communication is (local) broadcast in nature and, hence, when a sensor executes an *action*, it can update the state of all its neighbors at once. This action can be thought of as a *write all* action. However, if two neighbors l and k of a sensor (say, j) try to execute their write all actions simultaneously then, due to collision, state of j remains unchanged. The actions of l and k may update the state of other sensors successfully.

To redress this deficiency, recently approaches [9, 10] have been proposed for generating programs in WAC model from programs written in abstract models considered in the distributed computing literature. Specifically, the transformation proposed in [9] takes any time division multiple access (TDMA) algorithm in WAC model (e.g., [11–13]) as input. If the algorithm in [11], which is self-stabilizing, deterministic and designed for grid based topologies, is used with [9] then the transformed program in WAC model is self-stabilizing and deterministically correct for grid based topologies. And, if the algorithms in [12,13], which are randomized, are used with [9] then the transformed program in WAC model is probabilistically correct. (Note that TDMA algorithm such as those in [14] cannot be used with the transformation algorithm in [9], as the algorithm is not correct under WAC model. Rather, in [14], the authors assume that when two writes collide

the result is an OR operation between them.) Likewise, since the transformation in [10] is randomized (using CSMA) and for arbitrary networks, it generates programs in WAC model that are probabilistically correct (even if the original program has been verified deterministically). Thus, if a self-stabilizing deterministic TDMA algorithm in WAC model were available then it would enable us to provide deterministic guarantees about the transformed program in WAC model. To the best of our knowledge, we are not aware of such self-stabilizing deterministic TDMA algorithm for arbitrary networks.

With this motivation, in this paper, we propose a self-stabilizing deterministic TDMA algorithm. This algorithm can be used to transform existing self-stabilizing abstract programs into programs in WAC model that are deterministically self-stabilizing. This feature is especially useful as there is a large class of self-stabilizing abstract programs in the literature (e.g., [7, 8, 15–17]) and there is a significant need for self-stabilization in sensor networks, where the environment is difficult to capture precisely and, hence, the ability to recover from unexpected transient faults is crucial. Moreover, if the network is deployed in inaccessible fields (e.g., [18, 19]) then self-stabilization is essential.

Organization of the paper. In Section 2, we precisely define the problem statement and the computational models. In Section 3, we present our self-stabilizing TDMA algorithm in shared-memory model that is traditionally considered in distributed computing. Programs written in this model are easy to understand and, hence, we discuss our algorithm first in this model. In this algorithm, we reuse existing graph traversal algorithms (e.g., [20–23]). Subsequently, in Section 4, we transform this algorithm into WAC model. Then, in Section 5, we show how stabilization can be added to the TDMA algorithm in WAC model. In Section 6, we discuss how sensors can request for additional bandwidth. In addition, we discuss some optimizations for addition of new sensors. In Section 7, we discuss some of the questions raised by this work and in Section 8, we discuss the related work. Finally, in Section 9, we make the concluding remarks.

2 Preliminaries

In this section, we formally state the problem, define the models of computation, and discuss the assumptions made in this paper.

Problem statement. TDMA is the problem of assigning timeslots to each sensor. Two sensors j and k can transmit in the same timeslot if j does not interfere with the communication of k and k does not interfere with the communication of j. In other words, j and k can transmit in the same slot if the communication

distance between j and k is greater than 2. Towards this end, we model the sensor network as a graph $G = (V, E)$, where V is the set of all sensors deployed in the field and E is the communication topology of the network. Specifically, if sensors j and k can communicate with each other then the edge $(j, k) \in E$. The function $distance_G(j, k)$ denotes the distance between j and k in G. Thus, the problem statement of TDMA is shown in Figure 1.

Problem statement: TDMA
Given a communication graph $G=(V, E)$; assign timeslots to V such that the following condition is satisfied:
If $j, k \in V$ are allowed to transmit at the same time, then $distance_G(j, k) > 2$

Figure 1: Problem statement of TDMA

Models of computation. We now precisely define shared-memory model and WAC model. The programs are specified in terms of guarded commands [24]; each guarded command (respectively, action) is of the form:

$$guard \quad \longrightarrow \quad statement,$$

where *guard* is a predicate over program variables, and *statement* updates program variables. An action $g \longrightarrow st$ is enabled when g evaluates to true and to execute that action, st is executed. A computation of this program consists of a sequence $s_0, s_1, \ldots$, where s_{j+1} is obtained from s_j $(j \geq 0)$ by executing actions (one or more, depending upon the semantics being used) in the program.

A computation model limits the variables that an action can read and write. Towards this end, we split the program actions into a set of processes. Each action is associated with one of the processes. We now describe how we model the restrictions imposed by shared-memory model and WAC model.

Shared-memory model. In this model, in one atomic step, a sensor can read its state as well as the state of its neighbors and write its own (public and private) variables.

Write all with collision (WAC) model. In this model, each sensor consists of write actions (to be precise, write-all actions). Specifically, in one atomic action, a sensor can update its own state and the state of all its neighbors. However, if two or more sensors simultaneously try to update the state of a sensor, say k, then the state of k remains unchanged. Thus, this model captures the fact that a message sent by a sensor is broadcast. But, if a sensor receives 2 messages simultaneously then they collide and both messages become incomprehensible.

Remark. In this paper, we use the terms process and sensor interchangeably.

Assumptions. We assume that there is a base station in the network that is responsible for graph

traversal/token circulation. Such a base station can be readily found in sensor network applications, where it is responsible for exfiltrating the data from the network to the outside world. For example, in the extreme scaling project [19], the network is split into multiple sections and each section has one or more higher-tier node(s) that is responsible for data gathering and network management. One of the higher-tier nodes in each section can be elected for token circulation in the corresponding section.

Next, we assume that each sensor knows the ID of the sensors that it can communicate with. This assumption is reasonable since the sensors collaborate among their neighbors when an event occurs. We assume that the maximum degree of the graph does not exceed a certain threshold, say, d. This can be ensured by having the deployment follow a certain geometric distribution or using a predetermined topology. Furthermore, we initially assume that the clocks are synchronized. Later, in Section 7, we discuss how sensors can synchronize their clocks.

3 Self-Stabilizing TDMA in Shared-Memory Model

In this section, we present our algorithm in shared-memory model. In Sections 4 and 5, we transform this algorithm into write all with collision (WAC) model that is consistent with sensor networks.

In this algorithm, we split the system architecture into 3 layers: (1) token circulation layer, (2) TDMA layer, and (3) application layer. The token circulation layer circulates a token in such a way that every sensor is visited at least once in every circulation. The TDMA layer is responsible for assigning timeslots to all the sensors. And, finally, the application layer is where the actual sensor network application resides. All application message communication goes through the TDMA layer. Now, we explain the functions of the first two layers in detail.

3.1 Token Circulation Layer

The token circulation layer is responsible for maintaining a spanning tree in the network and traversing the graph infinitely often. In this paper, we do not present a new algorithm for token circulation. Rather, we only identify the constraints that this layer needs to satisfy. The token circulation protocol should recover from token losses and presence of multiple tokens in the network. In other words, we require that the token circulation protocol be self-stabilizing. We note that graph traversal algorithms such as [20–23] satisfy these constraints. Hence, any of theses algorithms can be used.

Remark. Although TDMA slot assignment in shared-memory model is (expected to be) possible without a token circulation layer, we have used it to simplify the transformation to WAC model.

3.2 TDMA Layer

The TDMA layer uses a distance 2 coloring algorithm for determining the initial slots of the sensors. Hence, we present our algorithm in two parts: (1) distance 2 coloring and (2) TDMA slot assignment.

Distance 2 coloring. Given a communication graph $G = (V, E)$ for a sensor network, we compute E' such that two distinct sensors x and y in V are connected if the distance between them in G is at most 2. To obtain distance 2 coloring, we require that $(\forall (i, j) \in E' :: color.i \neq color.j)$, where $color.i$ is the color assigned to sensor i. Thus, the problem statement is defined in Figure 2.

Problem statement: Distance 2 coloring
Given a communication graph $G = (V, E)$; assign colors to V such that the following condition is satisfied:
$(\forall (i, j) \in E' :: color.i \neq color.j)$
where, $E' = \{(x, y) | (x \neq y) \wedge ((x, y) \in E \vee (\exists z \in V :: (x, z) \in E \wedge (z, y) \in E))\}$

Figure 2: Problem statement of distance 2 coloring

We use the token circulation protocol in designing a distance 2 coloring algorithm. In our algorithm, each sensor maintains two public variables: *color*, the color of the sensor and *nbrClr*, a vector consisting of $\langle id, c \rangle$ elements, where id is a neighbor of the sensor and c is the color assigned to corresponding sensor. Initially, *nbrClr* variable contains entries for all distance 1 neighbors of the sensor, where the corresponding color assignments are undefined. A sensor can choose its color from K, the set of colors. To obtain a distance 2 coloring, $d^2 + 1$ colors are sufficient, where d is the maximum degree in the graph (cf. Lemma 3.1). Hence, K contains $d^2 + 1$ colors.

Figure 3 shows the algorithm for distance 2 coloring. In this algorithm, whenever a sensor (say, j) receives the token from the token circulation layer, it executes actions A1 and A2 (in that order). Action A1 determines the colors used in the distance 2 neighborhood of j and chooses a non-conflicting color. Action A2 ensures that *color.j* is properly updated at its neighbors and subsequently forwards the token. We note that for simplicity of presentation, we represent action A2 separately from action A1. Whenever j receives the token, we require that action A2 is executed only after action A1 is executed at least once.

Action A1. First, j reads *nbrClr* of all its neighbors and updates its private variable *dist2Clr.j*. The variable *dist2Clr.j* is a vector similar to *nbrClr.j* and contains the colors assigned to the sensors at distance

2 of j. Next, j computes the set *used.j* which denotes the colors used in its distance 2 neighborhood. If $color.j \in used.j$, j chooses a color from K that is not used in its distance 2 neighborhood. Otherwise, j keeps its current color.

Action A2. Once j chooses its color, it requires that its neighbors read its current color. Specifically, j waits until all its distance 1 neighbors have copied *color.j*. Towards this end, sensor l will update *nbrClr.l* with $\langle j, color.j \rangle$ (using action A3) if j is a neighbor of l and *color.j* has changed. Once all the neighbors of j have updated *nbrClr* with *color.j*, j forwards the token (using the token circulation layer).

sensor j
const
  $N.j$ // neighbors of j
  K // set of colors
var
  public *color.j* // color of j
  public *nbrClr.j* // colors used by neighbors of j
  private *dist2Clr.j* // colors used at distance 2 of j
  private *used.j* // colors used within distance 2 of j
begin
A1: $token(j)$ $\longrightarrow$
  $dist2Clr.j := \{\langle id, c\rangle | \exists k \in N.j : (\langle id, c\rangle \in nbrClr.k) \wedge (id \neq j)\}$
  $used.j := \{c | \langle id, c\rangle \in nbrClr.j \vee \langle id', c\rangle \in dist2Clr.j\}$
  // choose an unused color from K, i.e., $\{K - used.j\}$.
  if($color.j \in used.j$) $color.j :=$ minimum color in $\{K - used.j\}$
A2: $token(j) \wedge (\forall l \in N.j : (\langle j, c\rangle \in nbrClr.l \wedge color.j = c))$ $\longrightarrow$
  forward token
A3: $(l \in N.j) \wedge (\langle l, c\rangle \in nbrClr.j) \wedge (color.l \neq c)$ $\longrightarrow$
  $nbrClr.j := nbrClr.j - \{\langle l, c\rangle\} \cup \{\langle l, color.l\rangle\}$
end

Figure 3: Algorithm for distance 2 coloring in shared-memory model

Now, we illustrate our distance 2 coloring algorithm with an example (cf. Figure 4). Let us assume that the token circulation layer maintains a depth first search (DFS) tree rooted at sensor r. Whenever a sensor receives a token, the TDMA layer computes the colors used in the distance 2 neighborhood and decides the color of the sensor. In Figure 4, let the colors assigned to sensors r, a, c and d be $0, 1, 2$ and 3 respectively. When sensor b receives the token, *nbrClr.b* contains $\{\langle c, 2\rangle\}$ and *dist2Clr.b* contains $\{\langle a, 1\rangle, \langle d, 3\rangle\}$. Thus, *used.b* contains $\{1, 2, 3\}$. Once this information is known, b determines its color. In this example, b sets its color to 0, the minimum color not used in its distance 2 neighborhood. Similarly, other sensors determine their colors.

Lemma 3.1 If d is the maximum degree of a graph then $d^2 + 1$ colors are sufficient to obtain distance 2 coloring.

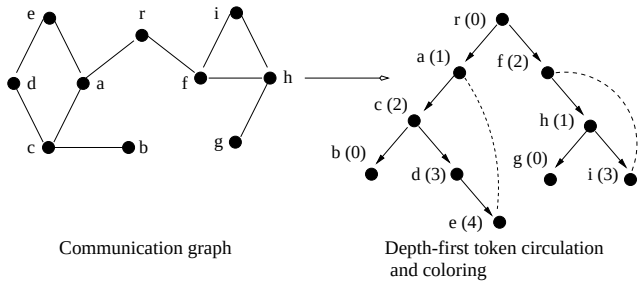


Figure 4: Color assignments using depth first search token circulation. The number associated with each sensor denotes the color assigned to that sensor. The dashed edges denote the back edges in the depth first search tree.

Proof. Based on the assumption about degree, given any vertex v, there exists at most d distance 1 neighbors, $d(d-1)$ distance 2 neighbors. Thus, at most d^2 vertices are within distance 2 of v. Now, we can arrange the vertices in some order and allow them to choose a color in such a way that the choice does not conflict with vertices that are considered earlier and within distance 2. When a vertex is about to choose a color, at most d^2 colors could be in its distance 2 neighborhood. Thus, a vertex can choose a color such that it does not overlap with the colors assigned to vertices in its distance 2 neighborhood. □

Corollary 3.2 For any sensor j, *used.j* contains at most d^2 colors. □

Theorem 3.3 The above algorithm satisfies the problem specification of distance 2 coloring. □

Theorem 3.4 Starting from arbitrary initial states, the above algorithm recovers to states from where the problem specification of distance 2 coloring is satisfied.

Proof. Based on the assumption in Section 3.1, the token circulation layer is self-stabilizing. The TDMA layer preserves the stabilization property of the token circulation layer since it eventually allows a sensor to forward the token. Thus, starting from arbitrary initial states, the token circulation algorithm self-stabilizes to states where only one token is present in the network. In the circulation of the token after stabilization, we show that the following conditions are satisfied.

- Given any sensor v_a that is visited by the token, color of v_a does not conflict with sensors that are within distance 2 of v_a and have been visited.
- Given any sensor v_a that is visited by the token, color of v_a is correctly captured in all its neighbors.

Let $v_1, v_2, \ldots, v_x$ be the path taken by the token after stabilization. It is straightforward to see that the above conditions are satisfied when the token is sent by v_1. Furthermore, based on the algorithm, these conditions are preserved when the token is passed. When the token circulation is complete, based on the

above conditions, it follows that the specification of distance 2 coloring is satisfied and the colors will be unchanged in subsequent token circulations. □

TDMA slot assignment. Once a sensor (say, j) decides its color, it can compute its TDMA slots. Specifically, *color.j* determines the initial TDMA slot of j. And, future slots are computed using the knowledge about the period between successive TDMA slots. Since the maximum number of colors used in any distance 2 neighborhood is d^2+1 (cf. Lemma 3.1), the period between successive TDMA slots, $P = d^2+1$, suffices. Once the TDMA slots are determined, the sensor forwards the token in its TDMA slot. And, the sensor can start transmitting application messages in its TDMA slots. Thus, the algorithm for TDMA slot assignment is shown in Figure 5.

```
const P = (d^2 + 1);
when j decides its color
    j can transmit at slots color.j + c * P, where c ≥ 0
```

Figure 5: TDMA slot assignment algorithm in shared-memory model

In Figure 4, the maximum degree of the graph is 3. Hence, the TDMA period is 10. However, since the number of colors assigned to sensors is 5, the desired TDMA period is 5. We note that while the number of colors used by our algorithm is small as the value of the d is expected to be small in sensor networks, identifying an optimal assignment is not possible. This is due to the fact that the problem of distance 2 coloring is NP-complete even in an offline setup [25]. In [26, 27], approximation algorithms for offline distance 2 coloring in specific graphs (e.g., planar graphs) are proposed. However, in this paper, we consider the problem of distributed distance 2 coloring where each sensor is only aware of its local neighborhood. In this case, given a sensor with degree d, the slots assigned to this sensor and its neighbors must be disjoint. Hence, at least $d+1$ colors are required. Thus, the number of colors used in our algorithm is within d times the optimal. We present an algorithm for computing the TDMA period depending on the local knowledge of the maximum difference in colors assigned to distance 2 neighborhood of each sensor in Section 6.1.1.

Theorem 3.5 The above algorithm ensures collision-free communication.

Proof. Consider two distinct sensors j and k such that the distance between j and k in the communication graph G is at most 2. The timeslots assigned to j and k are $color.j + c * P$ and $color.k + c * P$ respectively, where c is an integer and $P = (d^2+1)$. Suppose a collision occurs when j and k transmit a message at slots $color.j + c_1 * P$ and $color.k + c_2 * P$ respectively, where $c_1, c_2 > 0$. In other words, $color.j + c_1 * P = color.k + c_2 * P$. From Theorem 3.3, we know that $color.j \neq color.k$. Therefore, collision will occur iff $|color.j - color.k|$ is a

multiple of P. However, since the distance between j and k is at most 2, $|color.j - color.k|$ is at most d^2 (less than P). In other words, $|color.j - color.k| \leq d^2 < P$. Hence, if j and k transmit at the same time, then the distance between them is greater than 2. This is a contradiction. Thus, collisions cannot occur in this algorithm. □

Since the distance 2 coloring algorithm is self-stabilizing (cf. Theorem 3.4), starting from arbitrary initial states, the algorithm recovers to states from where the initial TDMA slots assigned to the sensors are collision-free. Once the initial TDMA slots are recovered, the sensors can determine the future TDMA slots. Thus, we have

Theorem 3.6 Starting from arbitrary initial states, the above algorithm recovers to states from where collision-free communication is restored. □

4 TDMA Algorithm in WAC Model

In this section, we transform the algorithm presented in Section 3 into a program in WAC model that achieves token circulation and distance 2 coloring upon appropriate initialization. (The issue of self-stabilization is handled in Section 5.) As discussed earlier, in shared-memory model, in each action, a sensor reads the state of its neighbors as well as writes its own state. However, in WAC model, there is no equivalent of a read action. Hence, the action by which sensor j reads the state of sensor k in shared-memory model is simulated by requiring k to write the appropriate value at j. Since simultaneous write actions by two or more sensors may result in a collision, we allow sensors to execute in such a way that simultaneous executions do not result in collisions.

Observe that if collision-freedom is provided then the actions of a program in shared-memory model can be trivially executed in WAC model. Specifically, the write all action of a sensor (say, j) in WAC model can be thought of as simultaneous read action by all neighbors of j. Our algorithm in this section uses this feature and ensures that collision-freedom is guaranteed. Thus, the effect of execution of a token circulation program in WAC model is similar to the case where it is executed in shared-memory model.

To obtain a program in WAC model, we proceed as follows. In this program, in the initial state, (a) sensors do not communicate among each other and (b) *nbrClr* and *dist2Clr* variables contain entries such that the color assignments are undefined. We present our algorithm in two parts: (1) distance 2 coloring,

and (2) TDMA slot assignment.

Distance 2 coloring. Initially, the base station (i.e., sensor r) circulates the token for obtaining distance 2 coloring. Whenever a sensor (say, j) receives the token (from the token circulation layer), it chooses its color. Towards this end, j first computes the set *used.j* which denotes the colors used in its distance 2 neighborhood. If *nbrClr.j* (or *dist2Clr.j*) contains $\langle l, undefined \rangle$, l did not receive the token yet and, hence, *color.l* is not assigned. Therefore, j ignores such neighbors. Afterwards, j chooses a color such that $color.j \notin used.j$. Subsequently, j reports its color to its neighbors within distance 2 using the primitive *report_distance_2_nbrs* (discussed later in this section) and forwards the token. Thus, the action by which k reads the colors used in its distance 2 neighborhood (in shared-memory model) is modeled as a write action where j reports its color to the sensors in its distance 2 neighborhood using the primitive *report_distance_2_nbrs*.

Note that the order in which the token is circulated is determined by the token circulation algorithm used in Section 3, which is correct under the shared-memory model (e.g., [20–23]). Since token circulation is the only activity in the initial state, it is straightforward to ensure collision-freedom. Specifically, to achieve collision-freedom, if j forwards the token to k in the algorithm used in Section 3, we require that the program variables corresponding to the token are updated at j and k without collision in WAC model. This can be achieved using the primitive *report_distance_2_nbrs*. Hence, the effect of executing the actions in WAC model will be one that is permitted in shared-memory model. Figure 6 shows the transformed algorithm in WAC model.

```
sensor j
const   N.j, K
var     color.j, nbrClr.j, dist2Clr.j, used.j
begin
token(j)  ⟶  used.j := {c | ⟨id, c⟩ ∈ nbrClr.j ∨ ⟨id', c⟩ ∈ dist2Clr.j}
              color.j := minimum color in K − used.j
              execute report_distance_2_nbrs
              forward token
end
```

Figure 6: Algorithm for distance 2 coloring in WAC model

Theorem 4.1 The above algorithm satisfies the problem specification of distance 2 coloring.

Proof. Observe that, the action by which a sensor (say, j) reads the colors assigned to sensors in its distance 2 neighborhood is simulated in this algorithm by requiring j to write its color at the sensors within distance 2 of j. Since there is no other communication before color assignment, token circulation will succeed. Hence, from Theorem 3.3, it follows that the above algorithm satisfies the problem specification of distance

2 coloring. □

TDMA slot assignment. Once a sensor determines its color, it can compute its TDMA slots. Similar to the discussion in Section 3, the color of the sensor determines the initial TDMA slot. Subsequent slots can be computed using the knowledge about the period between successive slots. If d is the maximum degree of the communication graph G, the TDMA period, $P = d^2 + 1$ suffices.

However, unlike the algorithm in Section 3 in shared-memory model, sensors do not start transmitting messages immediately. Otherwise, the token circulation may be interrupted due to collisions. Once the TDMA slots are determined, a sensor forwards the token in its TDMA slot. Hence, the token circulation does not collide with other TDMA slots. Next, a sensor waits until all the sensors in its distance 2 neighborhood have determined their TDMA slots before transmitting application messages in its TDMA slots. Thus, when a sensor starts transmitting application messages, all sensors in its distance 2 neighborhood have determined their TDMA slots and, hence, does not interfere with other TDMA slots and token circulation. Figure 7 shows the TDMA slot assignment algorithm.

const $P = (d^2 + 1)$;
if (j has decided its color) $\wedge$ (all sensors within distance 2 of j are colored)
  j can transmit application messages at slots: $color.j + c * P$, where $c \geq 0$

Figure 7: TDMA slot assignment algorithm in WAC model

Theorem 4.2 The above algorithm ensures collision-free communication. □

Implementation of *report_distance_2_nbrs*. In the above algorithm, we use the primitive *report_distance_2_nbrs*. In particular, whenever a sensor (say, j) decides its color, this primitive reports the color to its distance 2 neighborhood. Specifically, it updates the *nbrClr* value of its distance 1 neighbors and *dist2Clr* value of its distance 2 neighbors. We discuss its implementation, next.

Sensor j sends a broadcast message with its color and a schedule for its distance 1 neighbors. The sensors at distance 1 of j update their *nbrClr* values. Based on the schedule in the report message, each of the neighbors broadcast their *nbrClr* vectors. Specifically, if a distance 1 neighbor (say, l) of j is already colored, the schedule requires l to broadcast *nbrClr.l* in its TDMA slot. Otherwise, the schedule specifies the slot that l should use such that it does not interfere with the slots already assigned to j's distance 2 neighborhood. If there exists a sensor k such that $distance_G(l, k) \leq 2$, then k will not transmit in its TDMA slots, as l is not yet colored. (Recall that a sensor transmits application messages only if all its distance 2

neighbors have determined their TDMA slots.) Now, a sensor (say, m) updates *dist2Clr.m* with $\langle j, color.j \rangle$ iff $(m \neq j) \wedge (j \notin N.m)$. Thus, this schedule guarantees collision-free update of *color.j* at sensors within distance 2 of j. Furthermore, this primitive requires at most $d+1$ update messages.

5 Adding Stabilization in WAC Model

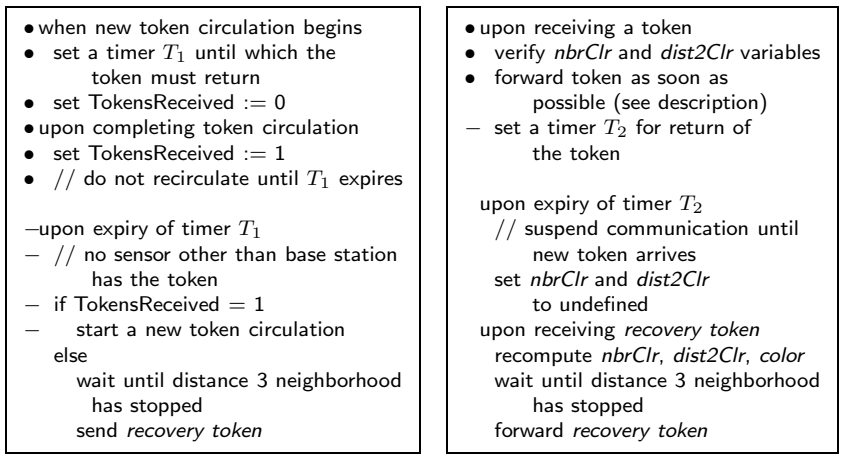
In the algorithm presented in Section 4, if the sensors are assigned correct slots then validating the slots is straightforward. Towards this end, we can use a simple diffusing computation to allow sensors to report their colors to distance 2 neighborhood and ensure that the slots are consistent. For simplicity of presentation, we assume that token circulation is used for revalidating TDMA slots. Now, in the algorithm presented in Section 4, we observe that in the absence of any faults, the token circulates the network successfully and, hence, slots are revalidated. However, in the presence of faults, the token may be lost due to a variety of reasons, such as, (1) slots assigned to sensors are not collision-free, (2) *nbrClr* values are corrupted, and/or (3) token message is corrupted. Or, due to transient faults, the token may circulate in a cycle or there may be several tokens.

To obtain self-stabilization, we use the *convergence-stair* approach proposed in [28]. First, we ensure that the token does not circulate in a cycle and if the system contains multiple tokens then it recovers to states where there is at most one token. Then, we ensure that the system recovers to states where there is a unique token (cf. Figure 8).

Step 1: Dealing with multiple tokens. During token circulation, there may be multiple tokens in the network or the tokens may circulate in a cycle. To deal with these problems, we add a time-to-live (TTL) field to the token message. Whenever the base station initiates a token circulation, it sets TTL to the number of hops the token traverses during one circulation. Since the token traverses an edge twice (once during visiting a sensor and once during backtracking), the base station sets TTL to $2 * |E_t|$, where $|E_t|$ is number of edges traversed by the token in one circulation. At each hop, the token decrements its TTL value. If this value is zero, the token circulation is terminated. Thus, this ensures that the token returns to the base station within $2 * |E_t|$ hops or it is lost.

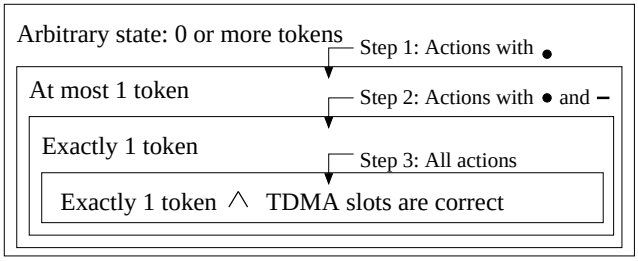
To deal with the case of multiple tokens, we ensure that any token in the network either returns to the base station within a predetermined time or it is lost. Towards this end, we ensure that a sensor forwards the token as soon as possible. To achieve this, whenever a sensor, say j, receives the token, j updates its color

at its neighbors in its TDMA slot. (This can be achieved within P slots, where P is the TDMA period.) Furthermore, in the subsequent slots, (a) the neighbors relay this information to distance 2 neighbors of j and (b) j forwards the token. (Both of these can be achieved within P slots.) Observe that if the TDMA slots are valid then any token will return in $2 * P * |E_t|$ slots to the base station. Otherwise, it may be lost.



(a) Actions at the base station

(b) Actions at the sensors



(c) Convergence to legitimate states

Figure 8: Adding stabilization

In order to revalidate the slots assigned to the sensors, the base station initiates a token circulation once every *token circulation period*, P_{tc} slots. The value of P_{tc} is chosen such that it is at least equal to the time taken for token circulation (i.e., $P_{tc} \geq 2 * P * |E_t|$). Thus, when the base station (i.e., r) initiates a token circulation, it expects to receive the token back within P_{tc} slots. Towards this end, the base station sets a timeout for P_{tc} duration whenever it forwards the token. Now, if the base station sends a token at time t and it does not send any additional token before time $t + P_{tc}$ then all tokens in the network at time t will return to the base station before time $t + P_{tc}$ or they will be lost. Hence, when the timeout expires,

there is no token in the network. If the base station does not receive any token before the timeout expires, it concludes that the token is lost. Similarly, whenever a sensor (say, $j \neq r$) forwards the token, it expects to receive the token in the subsequent round within P_{tc}. Otherwise, it sets the color values in *nbrClr.j* and *dist2Clr.j* to undefined. And, stops transmitting until it recomputes *color.j* and the sensors in its distance 2 neighborhood report their colors. Therefore, at most one token resides in the network at any instant. Thus, we have

Lemma 5.1 For any system configuration, if the base station initiates a token circulation at time t and does not initiate additional token circulation before time $t + P_{tc}$ then no sensor other than the base station may have a token at time $t + P_{tc}$. □

Steps 2 and 3: Recovery from lost token. Now, if the token is lost in the network, the base station initiates a recovery by sending a *recovery token*. Before the base station sends the *recovery token*, it waits until the sensors in its distance 3 neighborhood have stopped transmitting. This is to ensure that the primitive *report_distance_2_nbrs* can update the distance 2 neighbors of the base station successfully. Let T_{rt} be the time required for sensors in the distance 3 neighborhood of the base station to stop transmitting. Specifically, the value of T_{rt} should be chosen such that the sensors within distance 3 of the base station can detect the loss of the token within this interval. Although, the actual value of T_{rt} depends on the algorithm used for token circulation, it is bounded by P_{tc}. After waiting for T_{rt} amount of time, the base station recomputes its color. Furthermore, it reports its color to the sensors within distance 2 of it. As mentioned in Section 4, the primitive *report_distance_2_nbrs* ensures collision-free update since the sensors within distance 3 have stopped. Then, it forwards the *recovery token*.

Now, when a sensor (say, j) receives the *recovery token*, similar to the base station, it waits until the sensors in the distance 3 neighborhood of j have stopped transmitting. Then, j follows the algorithm in Section 4 to recompute its color. Once j decides its color, it uses the primitive *report_distance_2_nbrs* to update the sensors within distance 2 of j with *color.j*. Thus, we have

Lemma 5.2 Whenever a sensor (say, j) forwards the *recovery token*, sensors within distance 2 of j are updated with *color.j* without collision. □

The pseudo-code for stabilization and the illustration of how sensors converge to legitimate states are shown in Figure 8. Once a sensor recomputes its color, it can determine its TDMA slots using the algorithm

in Section 4. Thus, we have

Theorem 5.3 With the above modification, starting from arbitrary initial states, the TDMA algorithm in WAC model recovers to states from where collision-free communication is restored. □

Time complexity for recovery. Based on the above discussion, the value of T_{rt} depends on the algorithm used for token circulation. Suppose $T_{rt} = P_{tc}$, i.e., the base station waits for one token circulation period before forwarding the *recovery token*. Now, when the base station forwards the *recovery token*, all the sensors in the network would have stopped transmitting. Furthermore, whenever a sensor receives the token, it can report its color without waiting for additional time. To compute the time for recovery, observe that it takes (a) at most one token circulation time (i.e., P_{tc}) for the base station to detect token loss, (b) one token circulation for the sensors to stop and wait for recovery, and (c) at most one token circulation for the network to resume normal operation. Thus, the time required for the network to self-stabilize is at most $2 * P_{tc}+$ time taken for resuming normal operation. Since the time taken for resuming normal operation is bounded by P_{tc}, the time required for recovery is bounded by $3 * P_{tc}$.

We expect that depending on the token circulation algorithm, the recovery time can be reduced. Since this paper do not focus on a specific token circulation algorithm, we do not consider the issue of optimizing the recovery time. We refer the reader to Section 7 for a discussion on scalability and local recovery for small perturbations.

6 Extensions

In this section, we discuss mechanisms for improving the bandwidth utilization of sensors, propose techniques for improving the reliability of token circulation and recovery, and present optimizations for controlled topology changes.

6.1 Improving Bandwidth Utilization

In this section, first, we show how the TDMA period can be updated. Next, we show how the sensors can locally negotiate to request for additional bandwidth.

6.1.1 Dynamic Update of TDMA Period

In this extension, we focus on the problem of reducing the period between successive slots. This solution is based on the approach presented in [11]. Our solution involves three tasks: (1) allowing each sensor to compute the maximum difference in colors assigned to sensors within distance 2, (2) communicating the difference in the network, and (3) updating the TDMA period.

Task 1: Computing the desired local TDMA period. Regarding the first task, when a sensor (say, j) starts transmitting application messages, j has the knowledge about the colors assigned to sensors within distance 2 of j. Hence, j can compute the maximum difference ($LP.j$) among the colors assigned to the sensors in its distance 2 neighborhood. Specifically, $LP.j = max(\{\forall l, k : distance_G(l, j) \leq 2 \wedge distance_G(k, j) \leq 2 : |color.l - color.k|\}) + 1$. Since j maintains the colors assigned to sensors within distance 2 of j in $used.j$, $LP.j = max(\{\forall c_1, c_2 \in used.j \cup \{color.j\} : |c_1 - c_2|\}) + 1$. The variable $LP.j$ denotes the desired TDMA period for sensor j, since it reflects the maximum number of slots occupied in its distance 2 neighborhood.

Remark. In order to improve the TDMA period, we can ensure that a sensor chooses its color by locally minimizing the maximum difference in colors assigned to its distance 2 neighborhood. Specifically, whenever sensor j receives the token, it sets $color.j = c_j$ where c_j minimizes the quantity $max(\{\forall c_i \in used.j : |c_j - c_i|\})$. In other words, j chooses a color such that the maximum difference between its color and the colors assigned to its distance 2 neighborhood is minimized. Thus, this greedy approach minimizes the desired local TDMA period value of j.

Task 2: Computing the maximum local TDMA period. Regarding the second part, we use the token circulation algorithm to compute the maximum local TDMA period. Let $token.LP$ denote the maximum desired TDMA period determined so far. When the base station initiates token circulation, it sets $token.LP = LP.r$, where $LP.r$ denotes the maximum difference among the colors assigned to the sensors in the distance 2 neighborhood of the base station. Now, whenever a sensor (say, j) forwards the token, it sets $token.LP = max(token.LP, LP.j)$. It follows that when the base station receives the token back, it will obtain the maximum value of the desired TDMA period of all sensors in the network.

Task 3: Updating the TDMA period. Finally, regarding the third part, once the base station learns the new TDMA period value, it can include this when it initiates the next token circulation. Now, the sensors will learn the new TDMA period value. When the base station initiates the subsequent token circulation, the new TDMA period is used to determine the slots at which a sensor can send a message.

The above extension is intended to show that it is possible to dynamically update the TDMA period based on the colors assigned in the distance 2 neighborhood of all the sensors. However, we note that this approach may not improve the bandwidth utilization of the sensors if the number of colors used in a distance 2 neighborhood is equal to $|K|$ (i.e, all $d^2 + 1$ colors). In Section 6.1.2, we show how sensors can improve their bandwidth utilization by requesting for unused slots in its distance 2 neighborhood.

6.1.2 Local Negotiation Based Bandwidth Reservation

The algorithm in Section 5 allocates uniform bandwidth to all sensors. In this section, we consider an extension where a sensor can request for additional bandwidth, if available. This extension is based on the traditional mutual exclusion algorithms and it utilizes the fact that there is time synchronization and reliable timely delivery provided by TDMA.

In our TDMA algorithm, each sensor is aware of the slots used by the sensors in its distance 2 neighborhood. Hence, a sensor can determine the unused slots and if necessary request for the same. Whenever a sensor (say, j) requires additional bandwidth, it broadcasts a *request_slot* message in its TDMA slot. The message includes the slot requested by j and the time when j made the request. Since the message is broadcast, all distance 1 neighbors of j will receive the message. The distance 1 neighbors of j rebroadcast the message immediately to their neighbors in their earliest TDMA slots. If two or more *request_slot* messages are received before the communication slot assigned to a sensor, these messages are grouped and sent as a single request message.

Now, we show that if j transmitted its request in timeslot x_j and it did not receive any other request with timestamp ts such that $ts < x_j + P$ then j can access the requested timeslot without collisions. Towards this end, observe that if j transmits at slot x_j, all distance 1 neighbors of j can transmit at least once before j's next slot, $x_j + P$, where P is the TDMA period. Thus, if x_j is the slot when j requests unused slots, this request message is received by sensors in its distance 2 neighborhood within slot $x_j + P$. Likewise, if sensor l requests for the same slot such that $distance_G(j, l) \leq 2$, j will learn about l's request within time P of the request. Hence, if j does not receive a request with earlier timestamp before $x_j + P$ then j can use its requested slot without collisions. Furthermore, if j and l request for same slot then only one of them would succeed as the slots in which they request are different (due to collision-freedom of TDMA slots). Additionally, lease mechanisms [29] could be used to avoid starvation, where a sensor is required to renew the additional slots within a certain period of time.

Thus, sensors can request for unused slots when necessary using a simple local negotiation protocol. Furthermore, when a sensor requests unused slots, at most $d + 1$ request messages are transmitted, where d is the maximum degree of the communication graph. And, the sensor can determine whether or not it is allowed to use the requested slots within P slots.

6.2 Optimizations for Token Circulation and Recovery

In this section, we propose mechanisms that allows sensors to improve the reliability of token circulation. First, we note that in the algorithm in Section 5, whenever the token is lost, recovery is initiated by the base station. However, it is possible that the slots are still collision-free. This could happen if the token is lost due to message corruption or synchronization errors. To deal with this problem, the base station can choose to initiate recovery only if it misses the token for a threshold number of consecutive attempts.

Additionally, to ensure that the token is not lost due to message corruption, whenever a sensor (say, j) forwards the token, it expects its successor (say, $k \in N.j$) to forward the token within a certain interval. If j fails to receive such *implicit acknowledgment* from k, j retransmits the token (in its TDMA slots) a threshold number of times. If a sensor receives duplicate tokens, it ignores such messages. In [30], we have used implicit acknowledgments in the context of data dissemination across a large scale sensor network. Such data dissemination service is similar to a token circulation algorithm as a given message (respectively, token) is required to be disseminated reliably across the network. In [30], we show that the implicit acknowledgments improved the reliability of dissemination of messages by detecting message losses (for example, due to corruption) and recovered quickly from them, with the help of simulations and real-world experiments. Based on our experiences with the use of implicit acknowledgments, we expect that the reliability of token circulation can be improved with the help of such implicit acknowledgments.

6.3 Optimizations for Controlled Topology Changes

In our algorithm, controlled addition and removal of sensors do not affect the normal operation of the network. Let us first consider the removal/failure of sensors. Whenever a sensor is removed or fails, the TDMA slots assigned to other sensors are still collision-free and, hence, normal operation of the network is not interrupted. However, the slots assigned to the removed/failed sensor are wasted. We refer the reader to Section 6.1 for approaches on how to reclaim the wasted slots.

Suppose a sensor (say, q) is added to the network such that the assumption about the maximum degree is

not violated. Towards this end, we require that whenever a sensor forwards the token, it includes its color and the colors assigned to its distance 1 neighbors. Before q joins the network and starts transmitting application messages, we require q to learn the colors assigned to the sensors within its distance 2 neighborhood. One way to achieve this is by listening to token circulation of its distance 1 neighbors. Once q learns the colors assigned to sensors within distance 2, it can choose its color. Thus, q can determine the TDMA slots that it can use. Now, when q sends a message, its neighbors learn the presence of q and include it in the subsequent token circulations.

With this approach, if two or more sensors are added simultaneously then these new sensors may choose conflicting colors and, hence, collisions may occur. Since our algorithm is self-stabilizing, the network self-stabilizes to states where the colors assigned to all sensors are collision-free. Thus, new sensors can be added to the network. However, if adding new sensors violates the assumption about the maximum degree of the communication graph, slots may not be assigned to the sensors and/or collisions may occur.

7 Discussion

In this section, we discuss some of the questions raised by this work.

Scalability. One of the questions about the transformation is scalability. While the algorithm uses a token circulation approach for assigning initial colors (or recalculating them in the context of stabilization), we note that the algorithm provides acceptable performance in a typical scenario where sensor networks are deployed.

To illustrate the issue of scalability, we consider a network with 100 Mica-2 sensors. (Typically, networks with more than 100 sensors will be organized in sections to ensure that the path to a base station is within acceptable limits [19]. Then, our algorithm can be used independently for each section.) If such sensors are arranged in a 10x10 communication grid then five colors suffice [11]. The token circulation time (P_{tc}) in such networks is 0.99 minutes (where the timeslot interval is 30 ms = the time required to transmit one message in Mica-2 motes) and, hence the recovery time is $3 * P_{tc} = 2.97$ minutes.

Thus, the time required for the network to self-stabilize is small. Additionally, we expect that the number of colors required to obtain distance 2 coloring is small for random deployments. Therefore, our algorithm provides acceptable performance in such deployments.

Local recovery. It is possible to extend our algorithm so that sensors can locally correct the corrupted

slots when only a small number of sensors is corrupted. For example, if a sensor learns that its color overlaps with its neighbor within distance 2, it can change its color locally. Alternatively, if only a small set of sensors are corrupted then we could combine our algorithm with that in [12]. Specifically, whenever a sensor detects that the slots are corrupted, initially, it could use the algorithm in [12] to locally correct the slots. Thus, for the case where only a small subset of sensors are corrupted, the slots will be quickly restored. However, if it fails to assign slots in a fixed interval then the *recovery token* from the base station will restore the slots. Local recovery is especially useful if the base station tries multiple tokens before initiating recovery. Specifically, in this case, small perturbations are corrected locally. However, if the corruption is excessive then our algorithm will ensure recovery in a deterministic interval.

Edge coloring vs. vertex coloring. Our solution is based on vertex coloring where timeslots are assigned to each sensor. An alternative approach is edge coloring where timeslots are assigned to each edge. Formally, the problem of edge coloring is stated as follows: Let $f(a, b)$ be the color assigned to edge (a, b); then $\forall (x, y) \in E : (f(x, y) \notin (\{f(j, x) | j \text{ is a neighbor of } x\} \cup \{f(l, y), f(y, l) | (x \neq l) \wedge (l \text{ is a neighbor of } y)\})$. Now, a sensor (say, x) can send messages at slots $\exists y : y$ is a neighbor of $x : f(x, y)$. Moreover, x can send messages at slots $f(x, y) + c * K$, where $c \geq 0$ and K (the TDMA period) is the number of colors used in the network. Based on the color assignments, whenever x sends a message in the slot $f(x, y) + c * K$, sensor y receives the message successfully (although it may cause collision elsewhere). Hence, in order to broadcast a given message m with this approach, a sensor has to transmit m up to d times, where d is the maximum degree of the communication graph. Thus, edge coloring is not energy-efficient. By contrast, with vertex coloring, a sensor has to transmit only once in order to broadcast m to all its neighbors.

Time synchronization. We assume that all the sensors have identical clocks. Time synchronization can be achieved as follows: whenever a sensor receives the token, it synchronizes its clock with respect to its parent (i.e., the sensor from which it receives the token for the first time). Thus, sensors can deal with clock drifts and ensure that the slots are collision-free. Furthermore, in the case where TDMA slots are consistent, we can use time synchronization algorithms proposed in the literature for sensor networks. For example, we can integrate a time synchronization service (e.g., [31–33]) with the TDMA algorithm proposed in this paper. The time synchronization service synchronizes the clocks of the sensors within a few microseconds. Moreover, we expect that the performance of the time synchronization service will be improved as TDMA

will ensure that the time synchronization messages are transmitted successfully.

Violation of maximum degree assumption. As discussed in Section 5, whenever a sensor is added to the network, as long as the assumption about the maximum degree, d, of the communication graph is not violated, the normal operation of the network is not affected. However, if this assumption is violated then slots may not be assigned to the new sensors and/or collisions may occur. To deal with this problem, whenever a sensor is added such that the maximum degree of the communication graph is increased, we can use the approach proposed in Section 6.1.1 to increase the period between successive TDMA slots of the sensors. Towards this end, the base station can update the TDMA period while circulating the token.

Additionally, if the base station is not aware of the violation of the maximum degree, during stabilization, the sensors adjacent to the added sensors learn that the maximum degree has changed. Now, the sensors can use the algorithm in Section 6.1.1 to change the TDMA period accordingly. Thus, if sensors are added to the network in small numbers and in a controlled fashion, normal operation of the network will not be affected.

Variability in degree. If the communication topology of the network is such that the degree of sensors varies considerably in different parts of the network then bandwidth is underutilized in some parts of the network. To address this problem, in Section 6.1.1, we proposed a mechanism by which a sensor can calculate the ideal TDMA period. Specifically, during token circulation, we can compute the maximum difference in colors assigned in distance 2 neighborhood of all sensors, and update the period accordingly. If a sensor requires additional bandwidth, it can request for more slots using the local negotiation protocol proposed in Section 6.1.2.

8 Related Work

In this section, we compare and contrast the proposed algorithm with the related work [11–14, 34–38].

Self-stabilizing deterministic TDMA algorithms. Related work that deals with self-stabilizing deterministic TDMA algorithms include [11, 14, 34].

SS-TDMA. In [11], Kulkarni and Arumugam propose a self-stabilizing TDMA (SS-TDMA) algorithm where the topology is known and cannot change. However, in our algorithm, we allow addition/removal of sensors. Additionally, in our solution, we require that the sensors are only aware of their local neighborhood.

Self-stabilizing philosophers. In [34], Danturi et al proposed a self-stabilizing solution to dining philosophers

problem where a process cannot share the critical section (CS) with non-neighboring processes also. Such generalized dining philosophers problem has application in distance-k coloring, where k is the distance up to which a process cannot share CS. In [34], each process p is assumed to maintain a tree (rooted at p) that spans the processes with whom p cannot share CS using algorithms from the literature. However, existing tree construction and maintenance algorithms are not written for WAC model. On the contrary, in our algorithm, we show how a token circulation algorithm can be used in WAC model in order to obtain distance 2 coloring. And, on the other hand, unlike our algorithm, the approach in [34] allows concurrent coloring of processes.

BitMAC. In [14], the authors propose BitMAC, a deterministic, collision-free MAC protocol for sensor networks. One of the important assumptions in this paper is that when two writes collide the result is an OR operation between them. Moreover, the algorithm in [14] is not self-stabilizing. Unlike [14], our algorithm is written for WAC model and is also self-stabilizing.

Self-stabilizing randomized TDMA algorithms. In [12], Herman and Tixeuil propose a randomized TDMA slot assignment algorithm where a probabilistic fast clustering technique is used. In their algorithm, first, a maximal independent set is computed. This set identifies the leaders that are responsible for obtaining distance 2 coloring. Further, addition/removal of nodes in their algorithm can cause local collisions (and the effects are contained within distance 3 neighborhood). By contrast, our approach uses a deterministic algorithm to assign timeslots.

In [13], Busch et al propose a randomized TDMA algorithm for sensor networks. In their approach, initially, a randomized algorithm is used to determine the slots. Later, the sensors enter another phase where the TDMA period is reduced. Both these phases are self-stabilizing and are interleaved. By contrast, we propose a deterministic TDMA solution, where the sensors identify their timeslots without any collisions.

Other TDMA algorithms. Other TDMA algorithms include [35–38]. In [35], whenever a collision occurs during startup (synchronization phase), exponential backoff is used for determining the time to transmit next. One of the important assumption in [35] is that each node has a unique message length. By contrast, we do not make any such assumption in our TDMA algorithm.

In [36], Sohrabi and Pottie propose a network self-organization protocol, where nodes identify the presence of other nodes and form a multi-hop network. In [37], Arisha et al propose a clustering scheme to allot timeslots to different sensors. Each cluster has a gateway node that informs the sensors in its cluster about the timeslots in which the sensors can transmit. And, in [38], Heinzelman et al propose a clustering

algorithm. In these papers, initially, nodes are in random access mode and TDMA slots are assigned during network organization. By contrast, in our solution, we use a deterministic algorithm to assign timeslots. Unlike the algorithms proposed in [35–38], our algorithm is self-stabilizing.

9 Conclusion

In this paper, we presented a self-stabilizing deterministic time division multiple access (TDMA) algorithm for sensor networks. As discussed in [9], such algorithms suffice in transforming existing programs in shared-memory model into programs in write all with collision (WAC) model. This is especially useful since many of the problems considered in sensor networks (e.g., routing, data diffusion, synchronization, leader election) have been extensively studied in distributed computing. Thus, this algorithm can allow us to transform existing distributed programs and evaluate them in sensor networks. It follows that we can rapidly prototype a sensor network application with such transformations.

In [39], we evaluated the performance of the TDMA based transformation in reusing existing algorithms in the context of sensor networks. Specifically, we used *ProSe* [39], a programming tool for sensor networks to (i) specify a program in an abstract model considered in distributed computing literature (e.g., shared-memory model, read/write model) while hiding low-level details such as message collisions, message losses, resource limitations, and sensor failures, (ii) automatically transform it into WAC model using the TDMA based transformation, and (iii) generate and deploy code. We generated sensor network binaries for balanced routing program [16], tracking program [40], and distributed reset program [15]. We showed that the performance of transformed program is close to the performance of the programs generated manually for sensor networks, where the designer has to deal with low-level details in addition to the functionality of the program. Thus, the transformation algorithm reduces the development time of a typical sensor network application.

To our knowledge, this is the first algorithm that demonstrates the feasibility of deterministic transformation of a program in shared-memory model into a program in WAC model while preserving the property of self-stabilization on an arbitrary topology (where maximum degree of a node is known). By contrast, previous TDMA algorithms [11–13, 35–38] are limited to certain topologies (e.g., grid) or generate programs that are probabilistically correct.

There are several possible future directions for this work. While this algorithm demonstrates the feasibility of a self-stabilizing deterministic transformation for arbitrary topology and the recovery time for

the algorithm is expected to be acceptable for a typical deployment (cf. Section 7), one future direction is to develop a TDMA algorithm that (in addition to being deterministic and self-stabilizing) provides concurrency during recovery. Also, while our experience in [39] demonstrates that the efficiency of the program obtained by transformation is close to the program designed manually, another future direction is to quantify the efficiency of the transformed programs.

References

[1] M. Gouda and F. Haddix. The linear alternator. *In Proceedings of the Third Workshop on Self-Stabilizing Systems*, pages 31–47, 1997.

[2] M. Gouda and F. Haddix. The alternator. *In Proceedings of the Fourth Workshop on Self-Stabilizing Systems*, pages 48–53, 1999.

[3] G. Antonoiu and P. K. Srimani. Mutual exclusion between neighboring nodes in an arbitrary system graph tree that stabilizies using read/write atomicity. *In Proceedings of Euro-par'99 Parallel Processing*, 1999.

[4] M. Nesterenko and A. Arora. Self-stabilization preserving atomicity refinements. *Journal of Parallel and Distributed Computing*, 62(5):766–791, 2002.

[5] H. Kakugawa and M. Yamashita. Self-stabilizing local mutual exclusion on networks in which process identifiers are not distinct. *In Proceedings of the 21st Symposium on Reliable Distributed Systems (SRDS)*, pages 202–211, 2002.

[6] K. Ioannidou. Transformations of self-stabilizing algorithms. *In Proceedings of the 16th International Conference on Distributed Computing (DISC), Springer-Verlag*, LNCS:2508:103–117, October 2002.

[7] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11), 1974.

[8] S. Dolev. *Self-Stabilization*. The MIT Press, 2000.

[9] S. S. Kulkarni and M. Arumugam. Transformations for write-all-with-collision model. *Computer Communications (Elsevier), Special Issue on Dependable Wireless Sensor Networks (DWSN)*, 2005, to appear. Available at: `http://www.cse.msu.edu/~sandeep/publications/ka05COMCOM/`.

[10] T. Herman. Models of self-stabilization and sensor networks. *In Proceedings of the International Workshop on Distributed Computing (IWDC)*, 2003.

[11] S. S. Kulkarni and M. Arumugam. SS-TDMA: A self-stabilizing MAC for sensor networks. In *Sensor Network Operations*. Wiley-IEEE Press, March 2006, to appear. `http://www.cse.msu.edu/~sandeep/publications/ka05IEEEPress/`.

[12] T. Herman and S. Tixeuil. A distributed TDMA slot assignment algorithm for wireless sensor networks. *In Proceedings of the Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors), Springer-Verlag*, LNCS:3121:45–58, 2004.

[13] C. Busch, M. M-Ismail, F. Sivrikaya, and B. Yener. Contention-free MAC protocols for wireless sensor networks. *In Proceedings of the 18th Conference on Distributed Computing (DISC)*, 2004.

[14] M. Ringwald and K. Römer. BitMAC: A deterministic, collision-free, and robust MAC protcol for sensor networks. *In Proceedings of the European Workshop on Sensor Networks (EWSN)*, 2005.

[15] A. Arora and M. Gouda. Distributed reset. *IEEE Transactions on Computers*, 43(9):1026–1038, 1994.

[16] J. A. Cobb and M. G. Gouda. Balanced routing. *In Proceedings of the International Conference on Network Protocols (ICNP)*, 1997.

[17] T. Herman. A comprehensive bibliography on self-stabilization - a working paper. `http://www.cs.uiowa.edu/ftp/selfstab/bibliography`.

[18] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y-R. Choi, T. Herman, S. S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 46(5):605–634, 2004.

[19] A. Arora et al. ExScal: Elements of an extreme scale wireless sensor network. *In Proceedings of the International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, August 2005.

[20] C. Johnen, G. Alari, J. Beauquier, and A. K. Datta. Self-stabilizing depth-first token passing on rooted networks. *In Proceedings of the Workshop on Distributed Algorithms*, 1997.

[21] F. Petit and V. Villain. Color optimal self-stabilizing depth-first token circulation. *In Proceedings of the Symposium on Parallel Architectures, Algorithms, and Networks*, 1997.

[22] A. K. Datta, C. Johnen, F. Petit, and V. Villain. Self-stabilizing depth-first token circulation in arbitrary rooted networks. *Distributed Computing*, 13:207–218, 2000.

[23] F. Petit. Fast self-stabilizing depth-first token circulation. *In Proceedings of the Workshop on Self-Stabilizing Systems, Springer*, LNCS:2194:200–215, 2001.

[24] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1997.

[25] E. L. Lloyd and S. Ramanathan. On the complexity of distance-2 coloring. *In Proceedings of the International Conference on Computing and Information*, 1992.

[26] S. Ramanathan and E. L. Lloyd. Scheduling algorithms for multihop radio networks. *IEEE/ACM Transactions on Networking*, 1(2):166–177, April 1993.

[27] S. O. Krumke, M. V. Marathe, and S. S. Ravi. Models and approximation algorithms for channel assignment in radio networks. *Wireless networks*, 7(6):575–584, November 2001.

[28] M. G. Gouda and N. J. Multari. Stabilizing communication protocols. *IEEE Transactions on Computers*, 40(4):448–458, 1991.

[29] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, March 2003.

[30] S. S. Kulkarni and M. Arumugam. Infuse: A TDMA based reliable data dissemination protocol for sensor networks. *International Journal on Distributed Sensor Networks (IJDSN)*, 2006, to appear.

[31] S. Ganeriwal, R. Kumar, and M. Srivastava. Timing sync protocol for sensor networks. *In Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[32] J. van Greunen and J. Rabaey. Lightweight time synchronization for sensor networks. *In Proceedings of the Workshop on Wireless Sensor Networks and Applications (WSNA)*, September 2003.

[33] T. Herman. NestArch: Prototype time synchronization service. `http://www.ai.mit.edu/people/sombrero/nestwiki/index/ComponentTimeSync`, 2003.

[34] P. Danturi, M. Nesterenko, and S. Tixeuil. Self-stabilizing philosophers with generic conflicts. Technical Report TR-KSU-CSE-2005-05, Kent State University, August 2005.

[35] V. Claesson, H. Lönn, and N. Suri. Efficient TDMA synchronization for distributed embedded systems. *In Proceedings of the IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 198–201, October 2001.

[36] K. Sohrabi and G. J. Pottie. Performance of a novel self-organization protocol for wireless ad-hoc sensor networks. *In Proceedings of the IEEE Vehicular Technology Conference*, pages 1222–1226, 1999.

[37] K. Arisha, M. Youssef, and M. Younis. Energy-aware TDMA-based MAC for sensor networks. *In Proceedings of the IEEE Workshop on Integrated Management of Power Aware Communications, Computing and Networking (IMPACCT)*, May 2002.

[38] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, October 2002.

[39] M. Arumugam and S. S. Kulkarni. Programming sensor networks made easy. Technical Report MSU-CSE-05-25, Department of Computer Science, Michigan State University, September 2005.

[40] M. Demirbas, A. Arora, and M. Gouda. A pursuer-evader game for sensor networks. *In Proceedings of the Symposium on Self-Stabilizing Systems (SSS)*, June 2003.