Self-Stabilizing Deterministic TDMA 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 topology is unknown remains open.
In this paper, we present a self-stabilizing, deterministic algorithm for TDMA in networks where a sensor is aware of only its neighbors. 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 shared memory distributed program on an arbitrary topology into a program that is consistent with the sensor network model.

1 Introduction

The ability to write programs in an abstract model and then transform 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 can be ignored. Also, it is possible to thoroughly verify the abstract program 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 must preserve those properties.

For this reason, the problem of transformation from abstract programs to concrete programs has been studied in the literature [1–4]. These transformations have also focused on preserving the *self-stabilization* [5,6] property of the original

* 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.

program. 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 such a system recovers to legitimate states in-spite of unexpected (transient) faults, it is highly desirable for distributed computing.

Unfortunately, the results from [1–4] cannot be applied to deriving concrete programs for a sensor network, as the underlying model of computation in sensor networks is *write all with collision* (WAC) model [7]. 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. However, if two neighbors of a sensor try to execute their actions simultaneously then a collision occurs and none of the actions are successful.

To redress this deficiency, recently approaches [7, 8] have been proposed for transforming programs written in abstract models into WAC model. The transformation proposed in [7] takes any time division multiple access (TDMA) algorithm in WAC model (e.g., [9–12]) as input. If the algorithm in [9], which is self-stabilizing, deterministic and designed for grid based topologies, is used with [7] then the transformed program in WAC model is self-stabilizing and deterministically correct for grid based topologies. And, if the algorithms in [10–12], which are randomized, are used with [7] then the transformed program in WAC model is probabilistically correct. (Note that TDMA algorithm such as those in [13] cannot be used with [7], as the algorithm is not correct under WAC model. Rather, in [13], the authors assume that when two writes collide the result is an OR operation between them.) Likewise, since the transformation in [8] is randomized, it generates programs in WAC model that are probabilistically correct. 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 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 deterministically self-stabilizing programs in WAC model. This feature is useful as there is a large class of self-stabilizing programs in the literature (e.g., [5, 6, 14]) and there is a significant need for self-stabilization in sensor networks, where the environment is difficult to capture and, hence, ability to recover from unexpected transient faults is crucial.

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. Subsequently, we transform this algorithm into WAC model in Section 4 and add stabilization in Section 5. Finally, in Section 6, we make the concluding remarks.

2 Preliminaries

Problem statement. TDMA is the problem of assigning time slots to each sensor. Two sensors j and k can transmit in the same time slot 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 and E is the communication topology. 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 time slots to V such that the following condition is satisfied:
If $j, k \in V$ are allowed to transmit at the same time, then $distance_G(j, k) > 2$

Fig. 1. Problem statement of TDMA

Models of computation. Programs are specified in terms of guarded commands; each guarded command is of the form, $g \longrightarrow st$, where g is a predicate over program variables, and st 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 consists of a sequence $s_0, s_1, \ldots$, where s_{j+1} is obtained from s_j by executing actions in the program. A computation model limits the variables that an action can read and write. We split the 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 the shared-memory and the WAC models.

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 update its private variables) and write its own variables using its own 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 multiple messages are sent to a sensor simultaneously then, due to collision, it receives none.

Assumptions. We assume that there is a base station that is responsible for token circulation. Such a base station can be readily found in sensor network applications, where it is responsible for exfiltrating the data to the outside world (e.g., in the extreme scaling project [15], the network is split into multiple sections and each section has at least one base station for data-gathering and network management). 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. Finally, we assume that time

synchronization can be achieved during token circulation. Whenever a sensor receives the token, it may synchronize its clock with respect to its parent. Also, we can integrate the algorithms proposed in literature (e.g., [16]).

3 Self-Stabilizing TDMA in Shared-Memory Model

In this section, we present our algorithm in shared-memory model. 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. In this paper, we do not present a new algorithm for token circulation. Rather, we only identify the constraints that this layer should satisfy. Specifically, this layer should recover from token losses and presence of multiple tokens. In other words, we require that this layer be self-stabilizing. We note that graph traversal algorithms such as [17–20] satisfy these constraints. Hence, any of these algorithms can be used. The TDMA layer is responsible for assigning time slots 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 TDMA layer in detail.

3.1 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, $E' = \{(x,y) | (x \neq y) \wedge ((x,y) \in E \vee (\exists z \in V :: (x,z) \in E \wedge (z,y) \in E))\}$

Fig. 2. Problem statement of distance 2 coloring

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 colors 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.

Whenever a sensor (say, j) receives the token from the token circulation layer, 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 *used.j* which contains the colors used in its distance 2 neighborhood. If $color.j \in used.j$, j chooses a color from $K - used.j$. Otherwise, j keeps its current color. Once j chooses its color, it waits until all its distance 1 neighbors have copied *color.j*. Towards this end, sensor l will update *nbrClr.l* with $\langle j, color.j \rangle$ 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. Thus, the algorithm for distance 2 coloring is shown in Figure 3. (For simplicity of presentation, in Figure 3, we represent action A3, where j forwards the token after all its neighbors have updated their *nbrClr* values with *color.j*, separately. Whenever j receives the token, we require that action A3 is executed only after action A2 is executed at least once.)

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: $(l \in N.j) \wedge (\langle l, c \rangle \in nbrClr.j) \wedge (color.l \neq c)$ $\longrightarrow$
  $nbrClr.j := nbrClr.j - \{\langle l, c \rangle\} \cup \{\langle l, color.l \rangle\}$
A2: $token(j)$ $\longrightarrow$
  $dist2Clr.j := \{\langle id, c \rangle | \exists k \in N.j : (\langle id, c \rangle \in nbrClr.k) \wedge (id \neq j)\}$
  $used.j := \{c | \langle id, c \rangle \in nbrClr.j \vee \langle id, c \rangle \in dist2Clr.j\}$
  if($color.j \in used.j$) $color.j :=$ minimum color in $K - used.j$
A3: $token(j) \wedge (\forall l \in N.j : (\langle j, c \rangle \in nbrClr.l \wedge color.j = c))$ $\longrightarrow$
  forward token
end

Fig. 3. Algorithm for distance 2 coloring in shared-memory model

Lemma 3.1 If d is the maximum degree of a graph then $d^2 + 1$ colors are sufficient for distance 2 coloring. (cf. [21] for proofs of the theorems.) □

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 distance 2 coloring is achieved. □

TDMA slot assignment. In our algorithm, *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.

We note that identifying an optimal assignment is not possible as the problem of distance 2 coloring is NP-complete even in an offline setup [22]. In [23,24], approximation algorithms for offline distance 2 coloring in specific graphs (e.g., planar) 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.

Theorem 3.5 The above algorithm ensures collision-free communication. □

Since the distance 2 coloring algorithm is self-stabilizing (cf. Theorem 3.4), once the initial TDMA slots are recovered starting from arbitrary initial states, the sensors can determine the future TDMA slots.

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 in Section 3 into WAC model that achieves token circulation and distance 2 coloring upon appropriate initialization. (The issue of self-stabilization is handled in Section 5.) 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. Our algorithm in this section uses this feature and ensures that collision-freedom is guaranteed. In this algorithm, in the initial state, (a) sensors do not communicate and (b) *nbrClr* and *dist2Clr* variables contain entries such that the colors are undefined.

Distance 2 coloring. Whenever a sensor (say, j) receives the token, j computes *used.j* which contains 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 its neighbors (in shared memory model) is modeled as a write action where j reports its color using the primitive *report_distance_2_nbrs*. Figure 4 shows the transformed algorithm in WAC model.

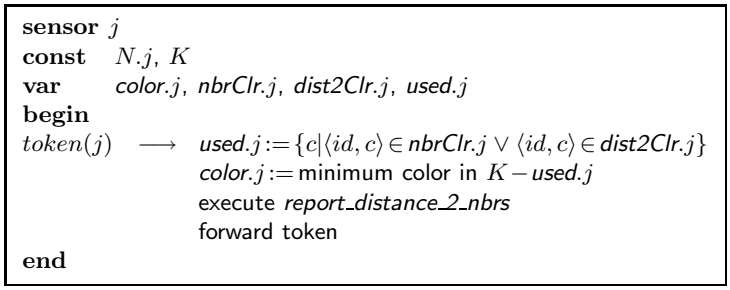


Fig. 4. Algorithm for distance 2 coloring in WAC model

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., [17–20]). 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.

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

TDMA slot assignment. 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 TDMA period. 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 as the TDMA message communication may interfere with the token circulation or the primitive *report_distance_2_nbrs*. 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 messages in its TDMA slots. A sensor learns this information when the sensors in its distance 2 neighborhood report their colors using the primitive *report_distance_2_nbrs*. 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 the primitive *report_distance_2_nbrs*.

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

Implementation of *report_distance_2_nbrs*. Whenever a sensor (say, j) decides its color, this primitive reports the color to its distance 2 neighborhood. It updates the *nbrClr* value of its distance 1 neighbors and *dist2Clr* value of its distance 2 neighbors. Towards this end, 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.

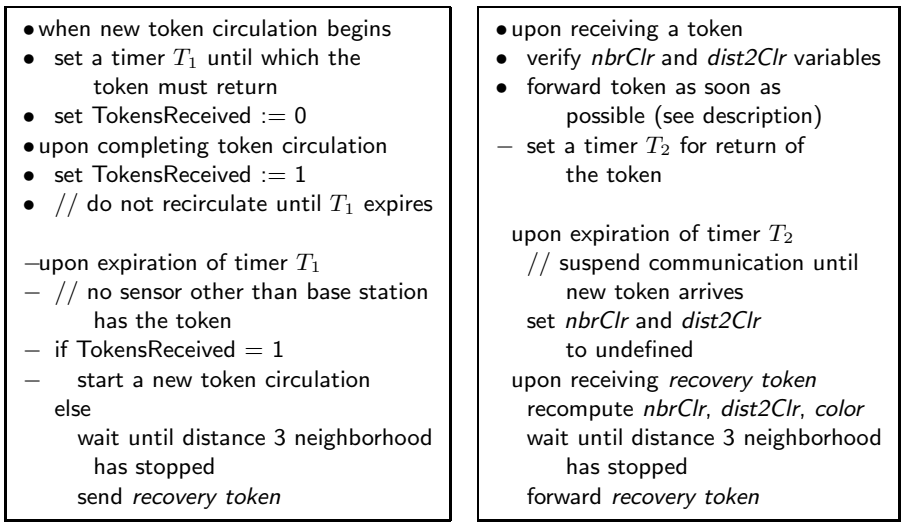
5 Adding Stabilization in WAC Model

In 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 absence of faults, the token circulates 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) TDMA slots are not collision-free, (2) *nbrClr* values are corrupted, and/or (3) token is corrupted. Or, due to transient faults, there may be several tokens.

To obtain self-stabilization, we use the *convergence-stair* approach proposed in [25]. First, we ensure that 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 5).

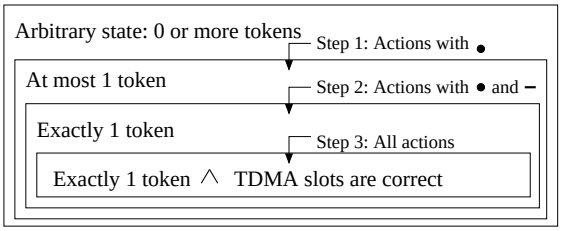
Step 1: Dealing with multiple tokens. In this step, we ensure that any token either returns to the base station within a predetermined time or it is lost. Towards this end, we ensure that a sensor forwards the token as soon as possible. To achieve this, whenever a sensor, say j, receives the token, j 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.) If the TDMA slots are valid then any token will return in $2 * P * |E_t|$ slots to the base station, where $|E_t|$ is number of edges traversed by the token. Otherwise, it may be lost.

In order to revalidate the slots, the base station initiates a token circulation once every *token circulation period*, P_{tc} slots. This value 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}. Towards this end, the base station sets a timeout for P_{tc} whenever it forwards the token. Now, if the base station sends a token at time t and it does not send additional token(s) before time $t + P_{tc}$ then all tokens at time t will return to the base station before time $t+P_{tc}$ or they will be lost. Hence,



(a) Actions at the base station

(b) Actions at the sensors



(c) Convergence to legitimate states

Fig. 5. Adding stabilization

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.

Lemma 5.1 For any configuration, if the base station initiates a token circulation at time t and does not circulate additional tokens 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, the base station initiates a recovery by sending a *recovery token*. Before it 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. 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 token circulation algorithm, it is bounded by P_{tc}. After T_{rt} amount of time, the base station 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 2 have stopped. Then, it forwards the *recovery token*.

When a sensor (say, j) receives the *recovery token*, it waits until the sensors in the distance 3 neighborhood of j have stopped. Then, j follows the algorithm in Section 4 to compute its color and report it to its distance 2 neighborhood.

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 and illustration for stabilization are shown Figure 5. Once a sensor recomputes its color, it can determine its TDMA slots (cf. Section 4).

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. Suppose $T_{rt} = P_{tc}$, i.e., the base station waits for one token circulation before forwarding the *recovery token*. Now, when the base station forwards the *recovery token*, all the sensors in the network would have stopped transmitting. Further, 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. However, the issue of optimizing the recovery time is outside the scope of this paper.

Optimizations for token circulation and recovery. Whenever the token is lost, 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. Thus, the reliability of token circulation can be improved.

Optimizations for controlled topology changes. Whenever a sensor is removed or fails, the 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 sensors are wasted. We refer the reader to [21] on how the sensors can reclaim these slots.

Suppose a sensor (say, q) is added such that the maximum degree assumption 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 starts transmitting application messages, we require q to learn the colors assigned to 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. Now, when q sends a message, its neighbors learn q's presence and include it in subsequent token circulations. 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.

6 Conclusion

In this paper, we presented a self-stabilizing deterministic TDMA algorithm for sensor networks. Such algorithm suffice in transforming existing programs in shared memory model into WAC model. This is useful since many of the problems in sensor networks (e.g., routing, data diffusion, synchronization, leader election) have been extensively studied in distributed computing. Thus, this algorithm helps in quickly prototyping a sensor network application.

To our knowledge, this is the first algorithm that demonstrates the feasibility of deterministic transformation of shared memory distributed programs 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 algorithms [9–12] are limited to certain topologies (e.g., grid) or generate programs that are probabilistically correct.

There are several possible future directions for this work. One future direction is to develop a TDMA algorithm that (in addition to being deterministic and self-stabilizing) provides concurrency during recovery. Another future direction is to quantify the efficiency of the transformed program in WAC model using the TDMA algorithm proposed in this paper.

References

1. M. Gouda and F. Haddix. The alternator. *Workshop on Self-Stabilizing Systems*, 1999.

2. G. Antonoiu and P. K. Srimani. Mutual exclusion between neighboring nodes in an arbitrary system graph tree that stabilizies using read/write atomicity. *Euro-par'99 Parallel Processing*, 1999.
3. M. Nesterenko and A. Arora. Self-stabilization preserving atomicity refinements. *Journal of Parallel and Distributed Computing*, 62(5):766–791, 2002.
4. K. Ioannidou. Transformations of self-stabilizing algorithms. *Conference on Distributed Computing*, 2002.
5. E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 1974.
6. S. Dolev. *Self-Stabilization*. The MIT Press, 2000.
7. S. S. Kulkarni and M. Arumugam. Transformations for write-all-with-collision model. *Computer Communications (Elsevier)*, 2005, to appear.
8. T. Herman. Models of self-stabilization and sensor networks. *In Proceedings of the International Workshop on Distributed Computing (IWDC)*, 2003.
9. S. S. Kulkarni and M. Arumugam. SS-TDMA: A self-stabilizing MAC for sensor networks. In *Sensor Network Operations*. IEEE Press, 2005, to appear.
10. T. Herman and S. Tixeuil. A distributed TDMA slot assignment algorithm for wireless sensor networks. *Algorithmic Aspects of Wireless Sensor Networks*, 2004.
11. C. Busch, M. M-Ismail, F. Sivrikaya, and B. Yener. Contention-free MAC protocols for wireless sensor networks. *18th Conference on Distributed Computing*, 2004.
12. V. Claesson, H. Lönn, and N. Suri. Efficient TDMA synchronization for distributed embedded systems. *IEEE Symposium on Reliable Distributed Systems*, 2001.
13. M. Ringwald and K. Römer. BitMAC: A deterministic, collision-free, and robust MAC protcol for sensor networks. *European Workshop on Sensor Networks*, 2005.
14. T. Herman. A comprehensive bibliography on self-stabilization - a working paper. `http://www.cs.uiowa.edu/ftp/selfstab/bibiography`.
15. A. Arora et al. ExScal: Elements of an extreme scale wireless sensor network. *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2005.
16. T. Herman. NestArch: Prototype time synchronization service. `http://www.ai.mit.edu/people/sombrero/nestwiki/index/ComponentTimeSync`, 2003.
17. C. Johnen, G. Alari, J. Beauquier, and A. K. Datta. Self-stabilizing depth-first token passing on rooted networks. *Workshop on Distributed Algorithms*, 1997.
18. F. Petit and V. Villain. Color optimal self-stabilizing depth-first token circulation. *Symposium on Parallel Architectures, Algorithms, and Networks*, 1997.
19. 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.
20. F. Petit. Fast self-stabilizing depth-first token circulation. *In Proceedings of the Workshop on Self-Stabilizing Systems, Springer*, LNCS:2194:200–215, 2001.
21. M. Arumugam and S. S. Kulkarni. Self-stabilizing deterministic TDMA for sensor networks. Technical Report MSU-CSE-05-19, Michigan State University, 2005.
22. E. L. Lloyd and S. Ramanathan. On the complexity of distance-2 coloring. *International Conference on Computing and Information*, 1992.
23. S. Ramanathan and E. L. Lloyd. Scheduling algorithms for multihop radio networks. *IEEE/ACM Transactions on Networking*, 1(2):166–177, April 1993.
24. S. O. Krumke, M. V. Marathe, and S. S. Ravi. Models and approximation algorithms for channel assignment in radio networks. *Wireless networks*, 2001.
25. M. G. Gouda and N. J. Multari. Stabilizing communication protocols. *IEEE Transactions on Computers*, 40(4):448–458, 1991.