A Distributed and Deterministic TDMA Algorithm for Write-All-With-Collision Model

Mahesh Arumugam

Cisco Systems, Inc.,
San Jose, CA 95134
`maarumug@cisco.com`

Abstract. Several self-stabilizing time division multiple access (TDMA) algorithms are proposed for sensor networks. Such algorithms enable the transformation of programs written in abstract models considered in distributed computing literature into a model consistent with sensor networks, i.e., write all with collision (WAC) model. Existing TDMA slot assignment algorithms have one or more of the following properties: (i) compute slots using a randomized algorithm, (ii) assume that the topology is known upfront, and/or (iii) assign slots sequentially. If these algorithms are used to transform abstract programs into programs in WAC model then the transformed programs are probabilistically correct, do not allow the addition of new sensors, and/or converge in a sequential fashion. In this paper, we propose a self-stabilizing deterministic TDMA algorithm where a sensor is aware of only its neighbors. We show that the slots are assigned to the sensors in a concurrent fashion and starting from arbitrary initial states, the algorithm converges to states where collision-free communication among the sensors is restored. Moreover, this algorithm facilitates the transformation of abstract programs into programs in WAC model that are deterministically correct.

1 Introduction

One of the important concerns in programming distributed computing platforms is the model of computation used to specify programs. Programs written for platforms such as sensor networks and embedded systems often have to deal with several low level challenges of the platform (e.g., communication, message collision, race conditions among different processes, etc). Therefore, to simplify programming, it is important to abstract such low level issues. In other words, the ability to specify programs in an abstract model and later transform them into a concrete model that is appropriate to the platform is crucial.

Transformation of programs has been studied extensively (e.g., [1–6]). These transformations cannot be applied for sensor networks as the model of computation in sensor networks is *write all with collision* (WAC) model. In WAC model, whenever a sensor executes an *action*, it writes the state of all its neighbors in one atomic step. However, if two neighbors j and k of a sensor (say i) try to execute their write actions at the same time then, due to collision, state of i will remain unchanged. The actions of j and k may update the state of their other neighbors successfully.

Recently, several approaches have been proposed to transform programs written in abstract models considered in distributed computing literature into programs in WAC model [7–10]. In [7], the author proposes a transformation to correctly simulate an abstract program in sensor networks. This algorithm uses carrier sensor multiple access (CSMA) to broadcast the state of a sensor and, hence, the transformed program is randomized. And, the algorithm in [9] uses time division multiple access (TDMA) that ensures collision-free write actions. In this approach, in WAC model, each sensor executes the *enabled* actions in the TDMA slots assigned to that sensor. And, the sensor writes the state of all its neighbors in its TDMA slots. If the TDMA algorithm in [11], a self-stabilizing and deterministic algorithm 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 randomized TDMA algorithms proposed in [8, 12] are used with [9] then the transformed program is probabilistically correct. Finally, the algorithm in [10], a self-stabilizing and deterministic TDMA algorithm for arbitrary topologies, allows one to obtain programs in WAC model that are deterministically correct for arbitrary topologies.

In this paper, we are interested in stabilization preserving deterministic transformation for WAC model. As mentioned above, a self-stabilizing deterministic TDMA algorithm enables such a transformation. One of the drawbacks of existing self-stabilizing deterministic TDMA algorithms (e.g., [10]) is that the recovery is sequential. Specifically, in [10], whenever the network is perturbed to arbitrary states (e.g., slots are not collision-free), a distinguished sensor (e.g., base station) initiates recovery and each sensor recomputes its slots one by one. However, it is desirable that the network self-stabilizes in a distributed and concurrent manner (without the assistance of distinguished sensors).

To redress this deficiency, in this paper, we propose a self-stabilizing deterministic TDMA algorithm that provides concurrent recovery. In this algorithm, whenever a sensor observes that the slots assigned to its neighbors are not collision-free, it initiates a recovery. As a result, its neighbors recover to legitimate states (i.e., the slots are collision-free) and the network as a whole self-stabilizes concurrently. We show that the algorithm supports addition or removal of sensors in the network. While a removal of a sensor does not affect the normal operation of the network, our algorithm ensures that the slots assigned to removed sensors are reused. And, our algorithm supports *controlled* addition of new sensors in the network.

Organization of the paper. The rest of the paper is organized as follows. In Section 2, we introduce the models of computation considered in distributed computing platforms and formally state the problem definition of TDMA. In Section 3, we present our distributed self-stabilizing TDMA slot assignment algorithm. And, in Section 4, we discuss extensions to our algorithm. Subsequently, in Section 5, we compare our algorithm with related work. Finally, in Section 6, we provide concluding remarks.

2 Preliminaries

In this section, we define the write all with collision model, formally state the problem, and list the assumptions made in this paper.

2.1 Write-All-With-Collision (WAC) Model and Collision Detectors

A computation model limits the variables that a program can read and write. Program actions are split into a set of processes (i.e., sensors). Each action is associated with one of the processes in the program.

In WAC model, each sensor consists of write actions (to be precise, write-all actions). In one atomic step, 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, WAC 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.

It is clear that WAC model does not provide any indication of collision. However, the physical layer of the communication stack may be required to expose state of the communication medium (e.g., collision information) to the higher layers of the stack. To enable such notifications, collision detectors are proposed in [13]. Collision detectors provide receiver-based notifications when message loss is detected. In [13], the authors identify 6 classes of collision detectors based on completeness and accuracy. In the context of this paper, we integrate *eventually accurate collision detector* to our model. In an eventually accurate collision detector, there exists a *frame*, say f_r, such that if k detects a collision in $f_{r'} \geq f_r$ then k does not receive some messages that were broadcast in $f_{r'}$.

2.2 Problem Statement

Distributed TDMA slot assignment. TDMA is the problem of assigning communication time slots to each sensor. Two sensors j and k cannot transmit in the same slot if their communication interferes with each other. In other words, j and k cannot transmit in the same slot if the communication distance between them is less than or equal to 2. To model this requirement, we consider the sensor network as a graph $G = (V, E)$ where V is the set of all sensors and E is the communication topology of the network. More precisely, if sensors $u \in V$ and $v \in V$ can communicate with each other then the edge $(u, v) \in E$. Finally, $distance_G(u, v)$ identifies the communication distance between u and v in G. The communication distance is the number of links in the shortest path between the two sensors. Thus, the problem statement of TDMA is shown in Figure 1.

Definition 1. *(TDMA frame) In TDMA, time is partitioned into fixed sized frames. Each TDMA frame is divided into fixed sized slots. In this paper, we ensure uniform bandwidth allocation among sensors. Therefore, each sensor is assigned one slot in every TDMA frame. A sensor is allowed to transmit in the slots assigned to it.*

Problem Statement: Distributed TDMA Slot Assignment
Consider the communication graph $G=(V,E)$; Given a sensor $j \in V$, assign time slots to j such that the following condition is satisfied:
$k \in V \wedge k \neq j \wedge \mathit{distance}_G(j,k) \leq 2 \Longrightarrow \mathit{slot}.j \cap \mathit{slot}.k = \emptyset$
where $\mathit{slot}.i$ identifies the slots assigned to sensor i.

Fig. 1. Problem statement of distributed TDMA slot assignment

Definition 2. *(TDMA period) The length of the TDMA frame is called the TDMA period. More specifically, it is the interval between the slots assigned to a sensor in consecutive frames.*

Distance 2 coloring. The problem statement of TDMA is similar to the problem of distance 2 coloring. Distance 2 coloring algorithm assigns colors to all the sensors in the network such that the colors assigned to distance 2 neighborhood of a sensor are unique. The color assigned to a sensor identifies the initial TDMA slot of that sensor. The sensor can compute its subsequent TDMA slots using TDMA period. Ideally, TDMA period $P = (d^2 + 1)$, where d is the maximum degree of the network. (We refer the reader to [10] for a proof that the number of colors required to obtain distance 2 coloring is at most $d^2 + 1$.) Thus, Figure 2 states the problem definition of distance 2 coloring.

Problem Statement: Distance 2 Coloring
Consider the communication graph $G=(V,E)$; Given a sensor $j \in V$, assign a color to j such that the following condition is satisfied:
$k \in V \wedge k \neq j \wedge \mathit{distance}_G(j,k) \leq 2 \Longrightarrow \mathit{color}.j \neq \mathit{color}.k$
where $\mathit{color}.i$ identifies the color assigned to sensor i.

Fig. 2. Problem statement of distance 2 coloring

Self-stabilization. An algorithm is self-stabilizing iff starting from an arbitrary state, it: (a) recovers to legitimate state and (b) upon recovery continues to be in legitimate states forever [14, 15]. Extending this definition, we have the problem statement of a self-stabilizing TDMA slot assignment algorithm as shown in Figure 3.

2.3 Assumptions

In this paper, we do not assume the presence of a base station. In our algorithm, the sensors collaborate among themselves to obtain distance 2 coloring and TDMA slot assignments. We assume that each sensor knows the IDs of the sensors that it can communicate with. This assumption is reasonable since the

Problem Statement: Self-Stabilizing TDMA Slot Assignment
Consider the communication graph $G = (V, E)$; A TDMA slot assignment algorithm is self-stabilizing iff starting from arbitrary initial states, the algorithm recovers to the following state:

$j \in V \wedge k \in V \wedge k \neq j \wedge \mathit{distance}_G(j,k) \leq 2 \Longrightarrow \mathit{slot.j} \cap \mathit{slot.k} = \emptyset$

and continues to remain in this state forever.

Fig. 3. Problem statement of self-stabilizing TDMA slot assignment

sensors collaborate among their neighbors when an event occurs. To simplify the presentation of the algorithm, we assume that frame numbers are not corruptible. However, we note that relaxing this assumption does not affect the correctness of the algorithm. Moreover, we can extend the algorithm to make frame numbers bounded. 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 the clocks of the sensors are synchronized. We can adopt the approach discussed in [10] to synchronize the clocks of the sensors.

3 TDMA Slot Assignment Algorithm

In this section, we present our distributed and deterministic TDMA algorithm. In Section 3.1, we give an outline of the algorithm. Then, in Section 3.2, we present the algorithm in detail. We discuss how the network self-stabilizes starting from arbitrary states to states where the slots are assigned as identified in Figure 3. Subsequently, in Section 3.3, we illustrate our algorithm with an example.

3.1 Outline of the Algorithm

Initially, the colors assigned to the sensors may be arbitrary. As a result, the communication among the sensors may not be collision-free. To achieve collision-free communication among the sensors, we adopt *distributed reset* (e.g., [16, 17]) approach. More specifically, whenever *collisions are detected* for a particular slot (i.e., color) for a threshold number of consecutive TDMA frames (say, at j), the algorithm resets the colors of appropriate sensor(s) in the neighborhood of j. In other words, a reset computation is used to update the colors assigned to the sensors such that the sensors in distance 2 neighborhood of j have unique colors and, thus, ensure that slots assigned to them are collision-free at j.

Towards this end, j schedules a reset computation in its current TDMA slots. It schedules the reset such that the following requirements are satisfied: (i) reset computations of others sensors in the distance 2 neighborhood of j do not interfere with each other and (ii) when j initiates reset, the sensors in the distance 3 neighborhood of j have stopped transmitting. The first requirement ensures that only one reset computation is active in a given neighborhood at any instant. Otherwise, simultaneous resets in a distance 2 neighborhood may result in collisions and/or sensors choosing conflicting colors. The second requirement ensures that the reset messages and update messages are communicated in a collision-free manner.

Whenever a sensor, say k, receives the reset message from j, first, it updates the color information it maintains about its distance 1 and distance 2 neighbors. Next, it checks if it has to change the color in response to the reset. If k needs to update its color, it chooses a non-conflicting color among the sensors in its distance 2 neighborhood. And, subsequently, k broadcasts change color message in its newly computed slots.

Now, whenever a sensor, say l, receives the change color message from k, first, it cancels any scheduled reset computations. Subsequently, l updates the color information it maintains about its distance 1 and distance 2 neighbors. When j receives change color message, it sends restart message to signal its distance 3 neighborhood to restart application communication. Thus, the algorithm resets the neighborhood of j to deal with a collision at j. However, note that one reset computation may not be sufficient to restore the state of the entire network.

3.2 Reset Computation and Slot/Color Assignment

In this section, we discuss the algorithm in detail. This is a 5-step algorithm: (1) observe collision and schedule reset computation, (2) send reset message, (3) update color, (4) notify color, and (5) restart communication. These steps may be repeated until the network self-stabilizes to legitimate states. (For reasons of space, we do not include a pseudo code for the proposed algorithm.)

Step 1: Observe collision and schedule reset computation. If a sensor, say j, observes collision at slot c_x (i.e., color c_x) for a threshold number of consecutive frames then it schedules a reset computation. Towards this end, first, j appends c_x to *collisions.j*, the list of collision slots it has observed so far. Also, it adds $(f_c.j, c_x)$ to *timestamp.j*, where $f_c.j$ is the frame in which j observed the collision at slot c_x. If j observed a collision for the first time then j determines the slot in which it can send a reset message. Sensor j schedules a reset computation such that requirements identified in Section 3.1 are met.

Requirement 1: Ensure only one active reset in the neighborhood. To satisfy this requirement, j schedules the reset computation in TDMA frame $f_{reset}.j = f_c.j + ID.j + D3_{timeout}$, where $ID.j$ is the ID of sensor j and $D3_{timeout}$ is defined below. This ensures that if two sensors observe a collision simultaneously, then their resets are scheduled in unique frames. On the other hand, if the sensors observe a collision in different frames, it is possible that their resets are scheduled in the same frame. However, before a sensor initiates a reset, requirement 2 ensures that the distance 3 neighborhood has stopped. As a result, the sensor that observed a collision earlier will be able to proceed.

Requirement 2: Ensure distance 3 neighborhood has stopped. Suppose j has scheduled reset in $f_{reset}.j$. Before j initiates reset, it has to wait until its distance 3 neighborhood stops transmitting messages. Towards this end, j stops transmitting for *at least* $D3_{timeout}$ frames before it fires the reset. $D3_{timeout}$ is the number of TDMA frames required for distance 3 neighborhood of j to stop transmitting messages. Specifically, when j stops, its neighbors will notice that j has stopped. As a result, distance 1 neighbors of j stop. Likewise, distance 2 and distance 3 neighbors of j stop. To prevent false positives, neighbor, say $l \in N.j$, stops only after it detects that j has stopped for a threshold number of consecutive frames, $stop_{timeout}$. Therefore, in order to ensure that distance 3 neighborhood of a sensor has stopped, $D3_{timeout} \geq 3 \times stop_{timeout}$.

Step 2: Send reset message. Each sensor, say j, maintains the state of its distance 2 neighborhood: *nbrClr.j* (contains the state of distance 1 neighbors of j) and *dist2Clr.j* (contains the state of distance 2 neighbors of j). Each entry in *nbrClr.j* contains color assignment and the last frame in which j or its neighbors received a message from the corresponding sensor. Likewise, each entry in *dist2Clr.j* contains color assignment and the last frame in which one of the neighbors of j received a message from the corresponding sensor. Initially, *nbrClr.j* and *dist2Clr.j* contain arbitrary color assignments that may not reflect the accurate state of its distance 2 neighborhood.

Notation. An entry in *nbrClr.j* is denoted as (k, c_k, f_k); this indicates that j last received a message from k in frame f_k and in slot (i.e., color) c_k. Entries in *dist2Clr.j* are denoted similarly. Additionally, we use "-" to wildcard or *ignore* a field in an entry. For example, $(-, c_x, -)$ indicates that we are interested in entries that have the color c_x. Additionally, we denote the current frame at j as $f_{current}.j$.

Sensor j initiates a reset in $f_{reset}.j$ only if it has not stopped transmitting in response to another reset. From Step 1, we note that j sends the reset message to its distance 1 neighbors in a collision-free manner. The reset message format is shown in Figure 4. This includes the state of distance 1 neighbors that j knows currently, list of collisions and their timestamps, the sensor that should update its color in response to this reset, and the initiator of the reset (i.e., j). Sensor j selects the sensor that should update its color based on IDs of the neighbors that j did not hear for a threshold number of consecutive frames.

$rm_j.neighborState$	*neighbor*	*color*	*lastReceived*
	j	*color.j*	$f_{current}.j$
	nbrClr.j		
$rm_j.collisionInfo$	*collisions.j*		
$rm_j.resetTimestamp$	*timestamp.j*		
$rm_j.sensorToChange$	l, where $l \in N.j$ is the sensor with lowest ID for which j did not hear any thing for a threshold number of frames		
$rm_j.initiator$	j		

Fig. 4. Reset message of j, rm_j

Theorem 1. *Reset computation initiated by any sensor executes in a collision-free manner.*

Proof. Suppose two reset computations execute simultaneously in a distance 2 neighborhood. Let k and l be two unique sensors that have initiated the reset such that $distance_G(k, l) \leq 2$. Both k and l should have observed a collision in the same frame and scheduled resets to start at the same frame. Otherwise, either one of them would have observed that the neighbors have stopped in response to a reset of the other and, hence, it would have stopped as well. Therefore, we have, $f_{reset}.k = f_{reset}.l$. In other words, $f_c.k + ID.k + D3_{timeout} = f_c.l + ID.l + D3_{timeout}$. Without loss of generality, assume that $ID.k < ID.l$. Now, we have $f_c.k > f_c.l$. More specifically, l observed the collision before k did. This is a contradiction. □

Step 3: Update color. Whenever a sensor, say k, receives the reset message rm_j, first, it cancels any scheduled reset. Next, it updates its neighbor state using the information in rm_j as shown in Figure 5. (Note that k updates an entry in *nbrClr.k* or *dist2Clr.k* only if the initiator j had received a message from the corresponding sensor most recently than that of k.)

```
if $(j = rm_j.initiator \wedge (j, c_j, -) \in rm_j.neighborState)$
    $nbrClr.k = \{nbrClr.k - (j, -, -)\} \cup (j, c_j, f_{current}.k)$
if $(p \in N.k \wedge (p, c_p, f_1) \in rm_j.neighborState \wedge (p, -, f_2) \in nbrClr.k \wedge f_2 < f_1)$
    $nbrClr.k = \{nbrClr.k - (p, -, -)\} \cup (p, c_p, f_1)$
else if $(p \notin N.k \wedge (p, c_p, f_1) \in rm_j.neighborState \wedge (p, -, f_2) \in dist2Clr.k \wedge f_2 < f_1)$
    $dist2Clr.k = \{dist2Clr.k - (p, -, -)\} \cup (p, c_p, f_1)$
// addition/removal of sensors are updated in nbrClr.k and dist2Clr.k as
    discussed in Section 4
```

Fig. 5. Updating *nbrClr.k* and *dist2Clr.k* of sensor k

Sensor k then checks if it has to update its color. If $k = rm_j.sensorToChange$ then j requires k to update its color. Sensor k updates its color as shown in Figure 6. Specifically, if *color.k* is in $rm_j.collisionInfo$, k chooses a color c from K (i.e., the set of all available colors) such that there is no collision in slot c at j and is unique among its distance 2 neighborhood.

Step 4: Notify color. If $k = rm_j.sensorToChange$, it sends *change color message* cm_k to all its neighbors as shown in Figure 7 (regardless of whether it changed its color or not). Specifically, k sends its color information, *nbrClr.k*, and the initiator of the reset. Whenever a sensor receives change color message, first, it cancels any scheduled resets. Next, it updates its *nbrClr* and *dist2Clr* similar to the discussion shown in Figure 5. Specifically, if l receives cm_k, it updates *nbrClr.l* with $(k, c_k, f_{current}.l)$, where $(k, c_k, -) \in cm_k.neighborState$. Similarly, l updates *nbrClr.l* and *dist2Clr.l* based on cm_k.

```
if ($k = rm_j.sensorToChange \wedge color.k \in rm_j.collisionInfo$) {
    $potentialColors = \{c | c \in K \wedge c \notin rm_j.collisionInfo \wedge (-, c, -) \notin nbrClr.k$
                        $\wedge (-, c, -) \notin dist2Clr.k\}$
    $color.k = min(potentialColors)$
}
```

Fig. 6. Updating color assignment of sensor k

$cm_k.neighborState$	*neighbor*	*color*	*lastReceived*
	k	$color.k$	$f_{current}.k$
	$nbrClr.k$		
$cm_k.initiator$	j		

Fig. 7. Change color message of k, cm_k

Theorem 2. *If a sensor updates its color in response to a reset then the change color message of that sensor is communicated in a collision-free manner.*

Proof. Let j be the initiator of the reset. And, $l \in N.j$ updates its color in response to the reset of j. When j initiates the reset (rm_j), distance 3 neighbors of j have stopped transmitting. Therefore, when l sends change color message cm_l, neighbors of l will receive it successfully. Hence, all neighbors of l will get the latest color assigned to l. □

Step 5: Restart communication. Whenever j initiates a reset, it expects to receive a change color message from $rm_j.sensorToChange$ before its next allotted slot in $f_{current}.j + 1$ frame. If j receives the change color message from the sensor that changed the color in response to reset of j, j cleans *collisions.j* and *timestamp.j*. Then, it signals its neighbors to restart application communication. Specifically, it sends restart message, sm_j; the format of sm_j is the same as change color message (cf. Figure 7). Once a sensor receives sm_j, it updates *nbrClr* and *dist2Clr* and starts application communication in its slots. Continuing in this fashion, the distance 3 neighborhood of j restarts. Note that the restart operation updates the color assignment of $l = rm_j.sensorToChange$ at distance 2 neighborhood of l, potentially causing collisions at some distance 2 neighbors of l. When a sensor hears a restart message or collision, it restarts communication.

On the other hand, if $l = rm_j.sensorToChange$ did not send change color message (possibly, due to failure of l) then j marks l as *potentially failed.* And, it cleans *collisions.j* and *timestamp.j*. Also, it sends a restart message. In future resets at j, j will not set l in $rm_j.sensorToChange$. If l has not failed, j will remove l from the list of potentially failed sensors when j hears from l.

Theorem 3. *If a sensor updates its color in response to a reset, eventually, the distance 2 neighborhood of that sensor learns the state of the sensor.*

Proof. Suppose $k \in N.j$ updates its color in response to a reset initiated by j. Distance 3 neighborhood of j have stopped transmitting in response to the reset of j. Therefore, we can conclude that sensors in distance 2 neighborhood of k have stopped transmitting. Now, when k sends change color message cm_k, distance 1 neighbors of k receive it successfully. When j sends restart message, distance 2 neighbors of k are updated. Note that it is possible that when distance 1 neighbors of k forward this restart, collisions may prevent some distance 2 neighbors of k to not receive the update. Future resets will restore the state of the neighborhood of k (cf. Figure 8 for illustration). Hence, eventually, state of k will be updated at all sensors in its distance 2 neighborhood. □

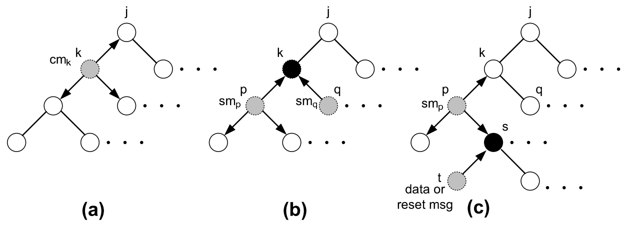


Fig. 8. Illustration of Theorem 3. (a) sensor k sends change color message cm_k to all its distance 1 neighbors. (b) sensor k forwards restart message sm_k to all its distance 1 neighbors. However, $p, q \in N.k$ may have the same color. As a result, when p and q forward sm_p and sm_q, some distance 2 neighbors of k may not be updated. This collision is detected by k and it will schedule a future reset. (c) sensor p forwards sm_p to its neighbors. However, sensor t such that $distance_G(p,t) = 2 \wedge distance_G(j,t) > 3$ may be assigned the same color as p. Future resets at s that detected this collision will restore the neighborhood. Note that (b) is a special case of (c).

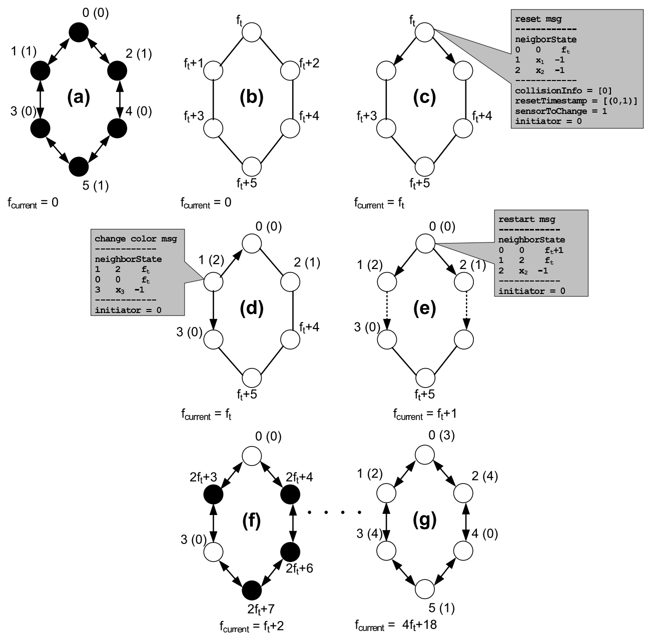
We note that in this algorithm at most one neighbor is recovered in any reset. Therefore, if j observes collisions at two or more colors/slots then j may observe collisions after this reset. Subsequent resets at j or at other sensors will eventually restore collision-free communication at j. Thus, we have

Theorem 4. *Eventually, the network self-stabilizes to the states where collision-free communication among the sensors is restored.* □

3.3 Illustration

Consider the topology shown in Figure 9(a). The color assignments of each sensor is specified along with its ID. For example, 2(1) denotes that sensor 2 is assigned color 1. Initially, we assume that $f_{current} = 0$ at all sensors. From Figure 9(a), we can note that every sensor observes a collision (shown with filled circles),

Each sensor, say j, determines the frame for reset: $f_{reset}.j = f_{current} + ID.j + f_t$, where $f_t = D3_{timeout}$ (cf. Figure 9(b)). Sensor 0 sets $rm_0.sensorToChange =$



```
reset msg
------------
neigborState
0   0    ft
1   x1  -1
2   x2  -1
------------
collisionInfo = [0]
resetTimestamp = [(0,1)]
sensorToChange = 1
initiator = 0
```

```
change color msg
------------
neighborState
1   2    ft
0   0    ft
3   x3  -1
------------
initiator = 0
```

```
restart msg
------------
neighborState
0   0    ft+1
1   2    ft
2   x2  -1
------------
initiator = 0
```

Fig. 9. Illustration of the TDMA slot assignment algorithm

1. As a result, sensor 1 changes its color to 2. Then, it sends a change color message, cm_1 (cf. Figure 9(d)). Once sensor 0 receives cm_1, it updates its state and sends restart message, sm_0 (cf. Figure 9(e)). Once sensors 1 and 2 receive sm_0, they restart their communication. Continuing in this fashion, distance 3 neighborhood of sensor 0 restart communication. As we can observe from Figure 9(f), message communication is still not collision free. Sensors then schedule subsequent resets and, finally, as shown in Figure 9(g), collision-free communication is restored. In this example, the network converges in $4f_t + 18$ frames. (Note that in this illustration all sensors are within distance 3 of each other.)

4 Extensions

In this section, we show how to extend the algorithm to deal with addition/removal of sensors. And, we present an approach to improve the bandwidth allocation of the sensors.

4.1 Dealing with Failure of Neighbors

In our algorithm, whenever a sensor (say j) hears a collision, it schedules a reset computation to restore collision-free communication. On the other hand, if j does not hear a message or observe a collision in a given slot, it could be because of the one of the following factors: (i) suppose $k \in N.j$ is the neighbor that is assigned the corresponding color; k may have failed, (ii) k may have stopped in response to a reset, or (iii) k does not have any data to send. If a sensor fails, the TDMA slots assigned to other sensors are still collision-free and, hence, normal operation of the network is not affected. However, the slots assigned to the failed sensors are wasted. In this section, we discuss an approach to reclaim slots assigned to failed sensors.

Towards this end, first, we introduce control message. Each sensor transmits a control message once in every $T_{control}$ frames. This message includes the color assignment of the sensor and its *nbrClr*. And, $T_{control}$ is determined when the network is deployed and is chosen based on how frequently the network changes. If topology changes are common, a smaller $T_{control}$ lets the sensors to quickly learn the state of their neighbors. On the other hand, a larger $T_{control}$ is more appropriate for a network that changes only occasionally.

To reclaim the slots, we proceed as follows. Sensor j concludes that $k \in N.j$ has failed if $f_{current}.j - lastReceived_k > T_{control}$, where $(k, -, lastReceived_k) \in nbrClr.j$. In other words, if j sees that it did not receive any message from k for more than $T_{control}$ frames, it concludes that k has failed.

When j concludes k has failed, it sets $(k, -, failed)$ in $nbrClr.j$. And, sends control message, $control_j$. Whenever a sensor observes that $(k, -, failed)$ is present in $control_j.neighborState$, it marks k as failed. The active neighbors of j remove $(k, -, -)$ from *nbrClr* or *dist2Clr*. This allows the sensors to reuse the color assigned to k to other sensors (in case of dynamic addition of new sensors or during reset computations). However, if k has not failed, it announces its presences in its current TDMA slots by sending $control_k$. When neighbors of k receive this message they update their *nbrClr* values. Subsequently, distance 2 neighbors of k also restore the state of k.

4.2 Dealing with Addition of Sensors

In this section, we discuss an approach to dynamically add new sensors in the network. This approach is similar to [10]. Suppose a sensor (say p) is added to the network such that the maximum degree of the network is not changed. Before p starts transmitting application messages, it listens to the message communication of its neighbors. To let p learn the colors used in its distance 2 neighborhood, we extend our algorithm as follows.

Sensor p waits for $T_{control}$ frames before it participates in the network. This allows p to learn distance 1 and distance 2 neighbors and their color assignments (from control messages of its neighbors). After $T_{control}$, p chooses a color. Next, p announces its presence to its neighbors by sending a control message in its newly computed slot. When a sensor receives a control message from p, it adds p to its neighbor list and updates *nbrClr*. Subsequently, distance 2 neighbors of p also learn its presence and update their *dist2Clr* values.

Thus, this approach allows the addition of new sensors in a neighborhood such that it does not violate the maximum degree assumption. However, if two or more sensors are added simultaneously, it is possible that they may choose the same color. Since our algorithm is self-stabilizing, the network will eventually self-stabilize to states where the color assignments are collision-free.

4.3 Improving the Bandwidth Allocation

In this section, we discuss an approach that allows the sensors to reduce the TDMA period and, hence, get better bandwidth allocation. The intuition behind this extension is that if c_x is the maximum color used in the network, the ideal TDMA period should be $c_x + 1$.

Each sensor (say j) maintains *maxColor.j* that denotes the maximum color used in its distance 2 neighborhood. It also maintains *controlMax.j* that denotes the maximum color used in the network. Note that j may not yet have the accurate information about the maximum color used in the network.

To improve the bandwidth allocation of the sensors, we extend the control message (discussed in Section 4.1) as follows. Any sensor in the network may decide to improve bandwidth allocation in the network. Let j decides to improve bandwidth allocation. It sends a control message, $control_j$ that includes $control_j.maxColorInfo = \max(controlMax.j, maxColor.j)$. Sensor j also indicates when the sensors can switch to new TDMA period, i.e., $control_j.switchOn = f_{switchOn}.j$, where $f_{switchOn}.j \geq f_{current}.j + 2 \times T_{control}$. (We discuss why this is necessary below.)

Whenever k receives $control_j$ with *maxColorInfo*, k sets *controlMax.k* and the frame in which it can switch to the new TDMA period as shown in Figure 10. Sensor k includes this information in its control messages. Thus, continuing in this fashion, each sensor will eventually learn the maximum color used in the network, i.e., *controlMax*. And, each sensor also knows the ideal TDMA period (i.e., $controlMax + 1$).

$$\boxed{\begin{array}{l} controlMax.k = \max(controlMax.k,\ control_j.maxColorInfo) \\ f_{switchOn}.k = \max(f_{switchOn}.k,\ control_j.switchOn) \end{array}}$$

Fig. 10. Receiving control message with *maxColorInfo*

Once the sensors have learned the maximum color used in the network, they can update their TDMA period. However, this operation should occur synchronously. In other words, all the sensors should update their TDMA period at the same time. Otherwise, collisions may occur. To address this issue, first, we note the following. If the TDMA slots assigned to the sensors are consistent then all the sensors learn the maximum color used in the network in at most $2 \times T_{control}$ frames, where $T_{control}$ is the period between two successive control messages (cf. Section 4.1). Since the initiator of this operation includes the frame in which new TDMA period is effective, each sensor knows exactly when to switch. Thus, the TDMA period can be updated to reflect the ideal value.

5 Related Work

Related work that deals with self-stabilizing deterministic slot assignment algorithms include [10, 11, 18]. In [11], Kulkarni and Arumugam proposed self-stabilizing TDMA (SS-TDMA). In this algorithm, the topology of the network is known upfront and remains static. Also, a base station is responsible for periodic diffusing computations to revalidate the slots. In [10], Arumugam and Kulkarni proposed self-stabilizing deterministic TDMA algorithm. Again, this algorithm assumes the presence of a base station that is responsible for token circulation. And, the slots are assigned in a sequential fashion.

In [18], 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. This problem has application in distance-k coloring, where k is the distance up to which a process cannot share CS. This algorithm requires each process p to maintain a tree rooted at itself that spans the processes with whom p cannot share CS.

Related work that deals with randomized algorithms for TDMA slot assignment include [8, 12]. In [8], Herman and Tixeuil proposed a probabilistic fast clustering technique for TDMA slot assignment. In this algorithm, first, a maximal independent set that identifies the leaders is computed. These leaders are then responsible for distance 2 coloring. In [12], Busch et al proposed a randomized algorithm for slot assignment. The algorithm operates in two phases: (1) to compute the slots and (2) to determine the ideal TDMA period. Both these phases are self-stabilizing and can be interleaved.

6 Conclusion

In this paper, we presented a self-stabilizing deterministic TDMA slot assignment algorithm for write all with collision (WAC) model. We showed that the algorithm allows sensors to recover concurrently and self-stabilize starting from arbitrary states. While the convergence time of the proposed algorithm is expected to be reasonable (since concurrent recoveries initiated by sensors that are sufficiently far apart are allowed), it can be improved further by integrating *neighborhood unique naming* scheme from [8] that assigns unique IDs for sensors within any distance 3 neighborhood.

Additionally, as discussed in [9], our algorithm is applicable in transforming existing programs in abstract models considered in distributed computing literature into programs in WAC model that are deterministically correct. This allows one to reuse existing solutions in distributed computing for problems such as routing, data dissemination, synchronization, and leader election in the context of sensor networks. Thus, the algorithm proposed in this paper allows one to transform such solutions and evaluate them in sensor networks. (We refer the reader to [19] for examples of such transformations, prototype implementations of the transformed programs, and their evaluations.)

References

1. Antonoiu, G., Srimani, P.K.: Mutual exclusion between neighboring nodes in an arbitrary system graph tree that stabilizes using read/write atomicity. In Europar'99 Parallel Processing, Springer-Verlag **LNCS:1685** (1999) 824–830
2. Gouda, M., Haddix, F.: The linear alternator. In Proceedings of the Third Workshop on Self-stabilizing Systems (1997) 31–47
3. Gouda, M., Haddix, F.: The alternator. In Proceedings of the Fourth Workshop on Self-stabilizing Systems (1999) 48–53
4. Ioannidou, K.: Transformations of self-stabilizing algorithms. In Proceedings of the 16th International Conference on Distributed Computing (DISC), Springer-Verlag **LNCS:2508** (October 2002) 103–117
5. Kakugawa, H., Yamashita, M.: 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) (2002) 202–211
6. Nesterenko, M., Arora, A.: Stabilization-preserving atomicity refinement. Journal of Parallel and Distributed Computing **62**(5) (2002) 766–791
7. Herman, T.: Models of self-stabilization and sensor networks. In Proceedings of the Fifth International Workshop on Distributed Computing (IWDC), Springer **LNCS:2918** (2003) 205–214
8. Herman, T., Tixeuil, S.: A distributed TDMA slot assignment algorithm for wireless sensor networks. In Proceedings of the Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors), Springer **LNCS:3121** (2004) 45–58
9. Kulkarni, S.S., Arumugam, M.: Transformations for write-all-with-collision model. Computer Communications **29**(2) (January 2006) 183–199
10. Arumugam, M., Kulkarni, S.S.: Self-stabilizing deterministic time division multiple access for sensor networks. AIAA Journal of Aerospace Computing, Information, and Communication (JACIC) **3** (August 2006) 403–419
11. Kulkarni, S.S., Arumugam, M.: SS-TDMA: A self-stabilizing mac for sensor networks. In: Sensor Network Operations. Wiley-IEEE Press (2006)
12. Busch, C., M-Ismail, M., Sivrikaya, F., Yener, B.: Contention-free MAC protocols for wireless sensor networks. In Proceedings of the 18th Annual Conference on Distributed Computing (DISC) (2004)
13. Chockler, G., Demirbas, M., Gilbert, S., Lynch, N., Newport, C., Nolte, T.: Consensus and collision detectors in wireless ad hoc networks. Distributed Computing (Springer) **21**(1) (2008) 55–84
14. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Communications of the ACM **17**(11) (1974)
15. Dolev, S.: Self-Stabilization. The MIT Press (2000)
16. Arora, A., Gouda, M.: Distributed reset. IEEE Transactions on Computers **43**(9) (1994) 1026–1038
17. Varghese, G., Arora, A., Gouda, M.G.: Self-stabilization by tree correction. Chicago Journal of Theoretical Computer Science **3** (1997)
18. Danturi, P., Nesterenko, M., Tixeuil, S.: Self-stabilizing philosophers with generic conflicts. In Proceedings of the Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems (November 2006)
19. Arumugam, M.: Rapid prototyping and quick deployment of sensor networks. PhD thesis, Michigan State University (2006)