A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks[*]

Ted Herman
University of Iowa
ted-herman@uiowa.edu

Sébastien Tixeuil[†]
LRI – CNRS UMR 8623 & INRIA Grand Large
tixeuil@lri.fr

March 8, 2006

Abstract

Wireless sensor networks benefit from communication protocols that reduce power requirements by avoiding frame collision. Time Division Media Access methods schedule transmission in slots to avoid collision, however these methods often lack scalability when implemented in *ad hoc* networks subject to node failures and dynamic topology. This paper reports a distributed algorithm for TDMA slot assignment that is self-stabilizing to transient faults and dynamic topology change. The expected local convergence time is $O(1)$ for any size network satisfying a constant bound on the size of a node neighborhood.

Moreover, the bandwidth that is allocated to a node depends on its local topology rather than on a global network parameter. This permits nodes in locally sparse areas to get more bandwidth share than nodes in locally dense areas.

Finally, in this scheme, nodes need not be able to detect collisions, be those induced by simultaneous transmission and reception at a single node, or by the well known hidden terminal problem.

Keywords: TDMA, distributed algorithms, self-stabilization, sensor networks, adaptativity.

1 Introduction

Collision management and avoidance are fundamental issues in wireless network protocols. Networks now being imagined for sensors [28] and small devices [4] require energy conservation, scalability, tolerance to transient faults, and adaptivity to topology change. Time Division Media Access (TDMA) is a reasonable technique for managing wireless media access, however the priorities of scalability and fault tolerance are not emphasized by

[*]A preliminary abstract of this paper appears in [13]

[†]Contact Author. Address: LRI Bâtiment 490, Université Paris Sud, 91405 Orsay cedex, FRANCE. Tel: +33 1691 54239. Fax: +33 1691 56586

most previous research. Recent analysis [11] of radio transmission characteristics typical of sensor networks shows that TDMA may not always substantially improve bandwidth when compared to randomized collision avoidance protocols, however fairness and energy conservation considerations remain important motivations. In applications with predictable communication patterns, a sensor may even power down the radio receiver during TDMA slots where no messages are expected; such timed approaches to power management are typical of the sensor regime.

Emerging models of *ad hoc* sensor networks are more constrained than general models of distributed systems, especially with respect to computational and communication resources. These constraints tend to favor simple algorithms that use limited memory. A few constraints of some sensor networks can be helpful: sensors may have access to geographic coordinates and a time base (such as GPS provides), and the density of sensors in an area can have a known, fixed upper bound. The question we ask in this paper is how systems can distributively obtain a TDMA assignment of slots to nodes, given the assumptions of synchronized clocks and a bounded density (where density is interpreted to be a fixed upper bound on the number of immediate neighbors in the communication range of any node). In practice, such a limit on the number of neighbors in range of a node has been achieved by dynamically attenuating transmission power on radios. Our answers to the question of distributively obtaining a TDMA schedule are partial: our results are not necessarily optimum, and although the algorithms we present are self-stabilizing, they are not optimally designed for all cases of minor disruptions or changes to a stabilized sensor network.

Before presenting our results, it may be helpful for the reader to consider the relation between problems of TDMA scheduling and graph coloring. Algorithmic research on TDMA relates the problem of timeslot assignment to minimal graph coloring, where the coloring constraint is typically that of ensuring that no two nodes within distance two have the same color. The constraint of distance two is motivated by the well known hidden terminal problem in wireless networks. This simple reduction of TDMA timeslot assignment neglects some opportunities for time division: even a solution to minimum coloring does not necessarily give the best result for TDMA slot assignment. Consider the two colorings shown in Figure 1, which are minimum distance-two colorings of the same network. We can count, for each node p, the size of the set of colors used within its distance-two neighborhood (where this set includes p's color); this is illustrated in Figure 2 for the respective colorings of Figure 1. We see that some of the nodes find more colors in their distance-two neighborhoods in the second coloring of Figure 1. The method of slot allocation in Section 6 allocates larger bandwidth share when the number of colors in distance-two neighborhoods is smaller. Intuitively, if some node p sees $k < \lambda$ colors in its distance-two neighborhood, then it should have at least a $1/(k+1)$ share of bandwidth, which is superior to assigning a $1/(\lambda+1)$ share to each color. Thus the problem of optimum TDMA slot assignment is, in some sense, harder than optimizing the global number of colors.

Contributions. The main issues for our research are dynamic network configurations, transient fault tolerance and scalability of TDMA slot assignment algorithms. Our approach to both dynamic network change and transient fault events is to use the paradigm of self-stabilization, which ensures that the system converges to a valid TDMA assignment after

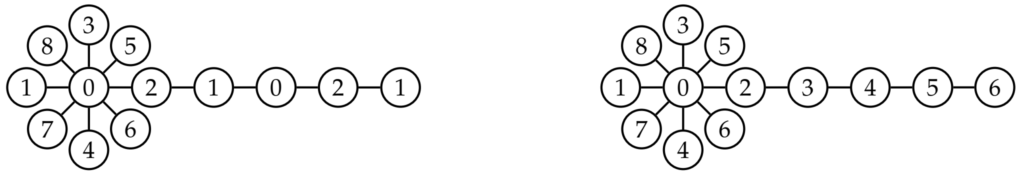


Figure 1: two solutions to distance-two coloring

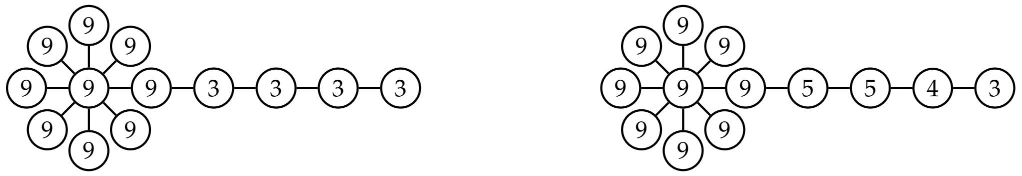


Figure 2: number of colors used within distance two

any transient fault or topology change event. Our approach to scalability is to propose a randomized slot assignment algorithm with $O(1)$ expected *local* convergence time. The basis for our algorithm is, in essence, a probabilistically fast clustering technique (which could be exploited for other problems of sensor networks). The expected time for *all* nodes to have a valid TDMA assignment is not $O(1)$; our view is that stabilization over the entire network is an unreasonable metric for sensor network applications; we discuss this further in the paper's conclusion.

Related Work. The idea of self-stabilizing TDMA has been developed in [17, 18] for a grid topology with a base station such that each node has knowledge of its location in the grid. The solutions of [17, 18] are deterministic and optimized for certain types of application communication patterns. Our work investigates the more general problem of unknown topology and unknown communication pattern. In [3], a distributed self-stabilizing algorithm for TDMA slot assignment is presented for general topologies, however that algorithm requires collision detection mechanisms not assumed by our model; the algorithm also has a "softer" form of self-stabilization than we use in this paper (in [3], certain initial state values are assumed for new nodes joining the network). A refined type of TDMA, for link scheduling, is considered in [7], which is a slot assignment problem when directed transmission is possible in the communication model, which not part of the model of this paper.

Algorithms for allocating TDMA time slots and FDMA frequencies are formulated as vertex coloring problems in a graph [21]. Let the set of vertex colors be the integers from the range $0..\lambda$. For FDMA the colors (f_v, f_w) of neighboring vertices (v, w) should satisfy $|f_v - f_w| > 1$ to avoid interference. A useful notation for this constraint is $L(\ell_1, \ell_2)$: for any pair of vertices at distance $i \in \{1, 2\}$, the colors differ by at least ℓ_i. The coloring problem for TDMA is: let $L'(\ell_1, \ell_2)$ be the constraint that for any pair of vertices at distance $i \in \{1, 2\}$, the colors differ by at least ℓ_i mod $(\lambda + 1)$. (This constraint represents the fact that time slots wrap around, unlike frequencies.) The coloring constraint for TDMA is $L'(1, 1)$. Coloring problems with constraints $L(1,0)$, $L(0,1)$, $L(1,1)$, and $L(2,1)$ have been well-studied not only for general graphs but for many special types of graphs [2, 15, 22]; many such problems are NP-complete and although approximation algorithms have been proposed, such algorithms are typically not distributed. (The related problem finding a minimum dominating set has been shown to have a distributed approximation using constant time [16], though it is unclear if the techniques apply to self-stabilizing coloring.) Self-stabilizing algorithms for $L(1,0)$ have been studied in [8, 25, 23, 24, 10], and for $L(1,1)$ in [9]. Our algorithms borrow from techniques of self-stabilizing coloring and renaming [9, 10], which use techniques well-known in the literature of parallel algorithms on PRAM models [19]. To the extent that the sensor network model is synchronous, some of these techniques can be adapted; however working out details when messages collide, and the initial state is unknown, is not an entirely trivial task. This paper is novel in the sense that it composes self-stabilizing algorithms for renaming and coloring for a base model that has only probabilistically correct communication, due to the possibility of collisions at the media access layer. Also, our coloring uses a constant number of colors for the $L(1,1)$ problem, while the previous self-stabilizing solution to this problem uses n^2 colors.

The rest of the paper is organized as follows. In Section 2, we present the underlying model that we consider and discuss implementation issues relative to the probabilistic nature of communications when there is no collision detection mechanism. In Sections 3, 4, and 5, we successively present the three layers of a minimal distance two coloring algorithm. A possible way to convert this coloring into a TDMA schedule along with bandwith guarantees is presented in Section 6. Section 7 states our main results, while we discuss open issues in Section 8. An appendix of independent interest provides guidance for emulating higher level models using our scheme.

2 Wireless Network, Program Notation

The system is comprised of a set V of nodes in an *ad hoc* wireless network, and each node has a unique identifier. Communication between nodes uses a low-power radio. Each node p can communicate with a subset $N_p \subseteq V$ of nodes determined by the range of the radio signal; N_p is called the neighborhood of node p. In the wireless model, transmission is omnidirectional: each message sent by p is effectively broadcast to all nodes in N_p. We also assume that communication capability is bidirectional: $q \in N_p$ iff $p \in N_q$. Define $N_p^1 = N_p$ and for $i > 1$, $N_p^i \;=\; N_p^{i-1} \;\cup\; \{\, r \mid (\exists q : \; q \in N_p^{i-1} \; : \; r \in N_q) \,\}$ (call N_p^i the distance-i neighborhood of p). Similarly, we denote by Γ_p the set of edges that are incident to p. We define $\Gamma_p^1 = \Gamma_p$ and for $i > 1$, $\Gamma_p^i \;=\; \Gamma_p^{i-1} \;\cup\; \{\, (q,r) \mid (\exists q : \; q \in N_p^{i-1}, r \in N_q) \,\}$ (call Γ_p^i the distance-i topology of p). Distribution of nodes is sparse: there is some known constant δ such that for any node p, $|N_p| \leq \delta$. (Sensor networks can control density by powering off nodes in areas that are too dense, which is one aim of topology control algorithms.)

Each node has fine-grained, real-time clock hardware, and all node clocks are synchronized to a common, global time. Each node uses the same radio frequency: one frequency is shared spatially by all nodes in the network. Media access is managed by CSMA/CA: if node p has a message ready to transmit, but is receiving some signal, then p does not begin transmission until it detects the absence of signal; and before p transmits a message, it waits for some random period (as implemented, for instance, in [27]) with a fixed upper bound. A queued transmit that has to wait for absence of signal longer than this upper bound is aborted. We assume that the implementation of CSMA/CA satisfies the following: there exists a constant $\tau > 0$ such that the probability of a frame transmission without collision is at least τ. This assumption corresponds to typical models for multiaccess channels [1]; the independence of τ for different frame transmissions indicates our assumption of an underlying memoryless probability distribution in a Markov model. We also assume that some constants $d > 0$, $\alpha > 0$ exist such that any p having a queued message to send at time t will observe the medium to be available for transmission before time $t + d$ with probability at least α. Jointly, τ and α imply a nonzero probability of being able to send a message without collision with at most delay d before transmission.

Nodes do not have the ability to detect collision of concurrently transmitted messages. For instance if node p begins transmitting and a message from node q concurrently arrives to p, then p will not detect any type of collision. Similarly if two messages, say from q and r concurrently arrive at node p, then p does not detect collision. In such a model, collisions are treated as noise, where noise is defined as any reception of a bits, in the form of a frame, whose error detection coding does not corroborate the frame's payload data. In some sensor network architectures, software is only able to observe the arrival of (correct) frames and unable to observe even the presence of noise, so techniques which rely on collision detection, such as those used in [3], are inapplicable here.

Notation. We describe algorithms using the notation of guarded statements: $G \rightarrow S$ represents a guarded statement, where G is a predicate of the local variables of a node, and S is an assignment to local variables of the node or a command, such as *transmit*. Predicate G is called the *guard*. Execution of a $G \rightarrow S$ consists of evaluating G, and if G is *true*, then S is executed, otherwise S is skipped. Some guards can be event predicates that hold upon the event of receiving a message: we assume that all such guarded assignments execute atomically when a message is received. At any system state where a given guard G holds, we say that G is *enabled* at that state. The [] operator is the nondeterministic composition of guarded assignments; $([]q : \ q \in M_p : \ G_q \rightarrow S_q)$ is a closed-form expression of $G_{q_1} \rightarrow S_{q_1} \ [] \ G_{q_2} \rightarrow S_{q_2} \ [] \ \cdots \ [] \ G_{q_k} \rightarrow S_{q_k}$, where $M_p = \{q_1, q_2, \ldots, q_k\}$.

Execution Semantics. The life of computing at every node consists of the infinite repetition of finding a guard and executing its corresponding assignment or skipping the assignment if the guard is *false*. Generally, we suppose that when a node executes its program, all statements with guards evaluating to *true* are executed within some constant time bound (done, for example, in round-robin order).

2.1 Shared Variable Propagation

A certain subset of the variables at any node are designated as *shared* variables. Nodes periodically transmit the values of their shared variables, based on a timed discipline. A simple protocol in our notation for periodic retransmission would be *true* $\rightarrow$ *transmit*(var_p) for each shared variable of p. One local node "variable" we do not explicitly use is the clock, which advances continuously in real time; guards and assignments could refer to the clock, but we prefer to discipline the use of time by the following procedure.

In addition to periodic retransmission, let each assignment to a shared variable schedule immediate transmission of the value of that variable. We impose the following time constraint: if $G \rightarrow S$ assigns to a shared variable, then we suppose execution of the statement is slow enough, in real time, so that its execution speed does not exceed some desired rate; we also suppose that the transmission protocol provides randomized delay to avoid collision in messages that carry shared variable values. Execution of $G \rightarrow S$ is not complete until the transmission of the message carrying the shared variable value is complete (whether collision occurs or not). However, to enforce the atomicity of $G \rightarrow S$ we require that evaluation of all local and shared variables in G and expressions used with S be effectively instantaneous: all of the time duration of $G \rightarrow S$ is due to random delay, queueing, and transmission. Call this type of time-constrained transmission with random delay a *time-constrained propagation*. Our intent of giving some upper bound on execution speed is to ensure that assignments to shared variables in nodes do not occur too fast: we use this bound to prove properties of a naming protocol presented in Section 3. We assume that periodic retransmissions are interleaved with assignment-triggered transmissions, if the node has any such triggered transmissions; no statement executes during a periodic retransmission. It follows that statements are executed serially within each node, there is at most one scheduled transmission at any instant.

Time-constrained propagation could be implemented using a timer associated with $G \rightarrow S$. One technique for implementing $G \rightarrow S$ is the following procedure:

> Suppose the previous invocation of the procedure implementing $G \rightarrow S$ finished at time t; the next evaluation of $G \rightarrow S$ occurs at time $t + \beta$, where β is a random delay inserted by the CSMA/CA implementation. After executing S, or skipping it if G is *false*, the node attempts a message transmission containing all shared variable values. Note that for CSMA/CA, message transmission may be postponed if the node is currently receiving a message, in which case the pending message is sent at the first available moment when no message is being received. Finally, after transmitting the message, the node waits for an additional κ time units, where κ is a given constant. Thus, in brief, $G \rightarrow S$ is forever evaluated by waiting for a random period, atomically evaluating $G \rightarrow S$, transmitting shared variable(s), and waiting for a constant time period κ. Figure 3 illustrates the cycle of shared variable propagation for one node.

Given the discipline of repeated transmission of shared variables, each node can have a cached copy of the value of a shared variable for any neighbor. This cached copy is updated atomically upon receipt of a message carrying a new value for the shared variable. To reconcile our earlier assumption of atomic processing of messages with the discipline of shared variable propagation, *no guarded assignment execution should change a shared variable in*

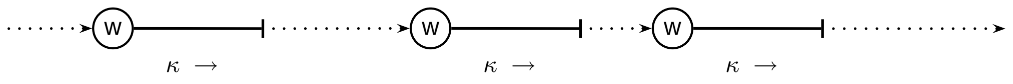


Figure 3: shared variable propagation

the atomic processing of receiving a message. This prescription simplifies our reasoning about concurrency and enables us to use traditional proof techniques for self-stabilizing protocols composed of guarded statements [12].

Lemma 1 *The model of atomic guarded statement execution is observed by our protocol of sending messages of cached variables.*

Proof: Our program constraints are as follows: for any guarded command that is triggered upon message receipt, no shared variable is modified. Also, we assumed the following axiom. For any guarded command that is triggered by a predicate on the local state of a node p, the actual execution of the guard can be seen as two different parts: an atomic part that uses the local state of the node, and a non-atomic part that transmits triggered or periodic messages.

First, we show that for any node p, guarded commands that are triggered upon message receipt either do not interfere with one another, because they relate to different private variables, or are received in sequence, because they were received by the same node. (Indeed, all of our algorithms use a cache mechanism to hold variables that pertain to particular neighbors, so the cache variable relative to one particular neighbor is to be updated only by this neighbor.)

Second, we show that a guarded command that is triggered upon message receipt does not interfere with a guarded command that is triggered by a predicate on a local state of a node p. The first part of the predicate-triggered command is atomic, so by hypothesis it does not interfere with the message-triggered command. The second part of the predicate-triggered command only transmits messages. Since none of the message-triggered commands are allowed to write shared variables or message buffers, those may not interfere with the non-atomic part of the predicate-triggered command. As a result, the two kinds of actions (message-triggered commands, and non atomic part of predicate-triggered commands) do not interfere with one another and may be executed in any order. In particular, they can be executed with the non-atomic part of the predicate-trigerred command occuring first.

Finally, predicate-triggered commands executed at the same node p are executed in sequence.

Since no actions are interfering with one another with the assumed hypothesis, our protocols exhibit the model of atomic guarded statement protocols. ❑

The goal of this paper is to provide an implementation of a general purpose, collision-free communication service. This service can be regarded as a transformation of the given wireless model into an abstract model without collisions. The method for shared variable propagation specifies that any execution of some $G \rightarrow S$ that assigns to a shared variable schedule immediate transmission of that variable's value. We also desire to exploit the property that shared variable propagation can be expected to succeed within constant time. A more efficient policy for evaluating guarded statements could be the following: atomically execute all guarded statements that assign to shared variables, in some arbitrary sequence, and then immediately schedule the combined shared variable propagation. All the protocols given in this paper have a constant number of guarded assignment statements, so this atomic evaluation of numerous statements is feasible. The resulting behavior can be shown to obey the semantics of guarded statement execution by arguments similar to those in the proof of Lemma 1.

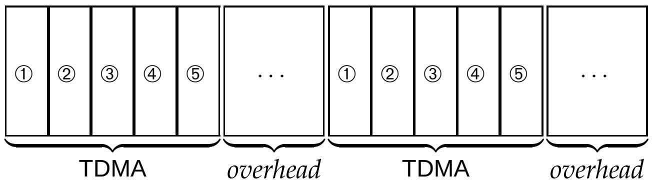
2.2 Problem Definition

Let $\mathcal{T}$ denote the task of assigning TDMA slots so that each node has some assigned slot(s) for transmission, and this transmission is guaranteed to be collision-free. We seek a solution to $\mathcal{T}$ that is distributed and self-stabilizing in the sense that, after some transient failure or reconfiguration, node states may not be consistent with the requirements of collision-free communication and collisions can occur; however eventually the algorithm corrects node states to result in collision-free communication.

2.3 Model Construction

Our first design decision is to suppose that the implementation we seek is not itself free of collisions. That is, even though our goal is to provide applications a collision-free service, our implementation is allowed to introduce "overhead messages" that are susceptible to collisions. Initially, in the development of algorithms, we accept collisions and resends of these overhead messages, which are internal to $\mathcal{T}$ and not visible to the user application which is using only TDMA slots for its own communication.

To solve $\mathcal{T}$ it suffices to assign each node a color and use node colors as the schedule for a TDMA approach to collision-free communication [21]. Even before colors are assigned, we use a schedule that partitions radio time into two parts: one part is for TDMA scheduling of application messages and the other part is reserved for the overhead messages of the algorithm, which assigns colors and time slots to nodes. The following diagram illustrates such a schedule, in which each TDMA part has five slots. Each overhead part occupies a fixed duration within in the TDMA schedule.



The programming model, including the technique for sharing variables described in Section 2, refers to message and computation activity in the overhead parts. It should be understood that the timing of shared variable propagation illustrated in Figure 3 may span overhead slots: the computation by the solution to $\mathcal{T}$ operates in the concatenation of all the overhead slots. Whereas CSMA/CA is used to manage collisions in the overhead slots, the remaining TDMA slots do not use random delay. During initialization or after a dynamic topology change, frames may collide in the TDMA slots, but after the slot assignment algorithm self-stabilizes, collisions do not occur in the TDMA slots.

With respect to any given node v, a solution $\mathcal{T}$ is *locally stabilizing* for v with convergence time t if, for any initial system state, after at most t time units, every subsequent system state satisfies the property that any transmission by v during its assigned slot(s) is free from collision. Solution $\mathcal{T}$ is *globally stabilizing* with convergence time t if, for every initial state, after at most t time units, every subsequent system state has the property that all transmissions during assigned slots are free from collision. For randomized algorithms, these definitions are modified to specify expected convergence times (all stabilizing randomized algorithms we consider are probabilistically convergent in the Las Vegas sense). When the qualification (local or global) is omitted, convergence times for local stabilization are intended for the presented algorithms.

Several primitive services that are not part of the initial model can simplify the design and expression of $\mathcal{T}$'s implementation. All of these services need to be self-stabilizing. Briefly put, our plan is to develop a sequence of algorithms that enable TDMA implementation. These algorithms are: neighborhood-unique naming, maximal independent set, minimal coloring, and the assignment of time slots from colors. In addition, we rely on neighborhood services that update cached copies of shared variables.

2.4 Neighborhood Identification

We do not assume that a node p has built-in knowledge of its neighborhood N_p or its distance-three neighborhood N_p^3. This is because the type of network under considering is *ad hoc*, and the topology dynamic. Therefore some protocol is needed so that a node can refer to its neighbors. We describe first how a node p can learn of N_p^2, since the technique can be extended to learn N_p^3 in a straightforward way.

Each node p can represent N_p^i for $i \in 1..3$ by a list of identifiers learned from messages received at p. However, because we do not make assumptions about the initial state of any node, such list representations can initially have arbitrary data. Let L be a data type for a list of up to δ items of the form $a : A$, where a is an identifier and A is a set of up to δ identifiers. Let sL_p be a shared variable of type L. Let message type mN with field of type L refer to a shared variable propogation message for sL_p. Let L_p be a private variable of a type that is an augmentation of L. Variable L_p contains $(a : A)$-type items and also associates a real number with each item: let *age*$(a : A)$ denote a positive real value attached to the item $(a : A)$. Our intent is that *age*$(a : A)$ should, at any instant, be the difference between the current clock and the time at which item $(a : A)$ was last updated. In the neighborhood identification protocol, L_p informs p of N_q^1 for each neighbor q, thereby enabling p to learn N_p^2.

Function update$(L_p, a : A)$ changes L_p to have new item information: if L_p already has some item whose first component is a, it is removed and replaced with $a : A$ (which then has age zero); if L_p has fewer than δ items and no item with a as first component, then $a : A$ is added to L_p; if L_p has already δ items and no item with a as first component, then $a : A$ replaces some item with maximal age.

Let maxAge be some constant designed to be an upper limit on the possible age of items in L_p. Function neighbors(L_p) returns the set

$$\{ q \mid q \neq p \ \wedge \ (\exists (a : A) : \ (a : A) \in L_p : \ a = q) \}$$

Given these variable definitions and functions, we present the algorithm for neighborhood identification.

```
N0: $receive$ mN$(a : A) \ \rightarrow \ $update$(L_p, a : A \setminus \{p\})$
N1: $([] \, (a : A) \in L_p : \ age(a : A) > $ maxAge $\ \rightarrow \ L_p := L_p \setminus (a : A) \ )$
N2: $true \ \rightarrow \ sL_p := (p :$ neighbors$(L_p))$
```

We cannot directly prove that this algorithm stabilizes because the CSMA/CA model admits the possibility that a frame, even if repeatedly sent, can suffer arbitrarily many collisions. Therefore the *age* associated with any element of L_p can exceed maxAge, and the element will be removed from L_p. The constant maxAge should be tuned to be small enough to remove old or invalid neighbor data, yet it must be large enough to retain current neighbor information until other mN messages can arrive before age expiration. This is an implementation issue beyond of the scope of this paper: our abstraction of the behavior of the communication layer is the assumption that, eventually for any node, the guard of N1 remains *false* for any $(a : A) \in L_p$ for which $a \in N_p$.

Proposition 1 *Eventually, for every node p, $sL_p = N_p$ holds continuously.*

Proof: Note: as is usual in the literature of self-stabilization, we prove the claim only for periods when there are no topology changes. Eventually any element $(a : A) \in L_p$ such that $a \notin N_p$ is removed. Therefore, eventually every node p can have only its neighbors listed in sL_p. Similarly, with probability 1, each node p eventually receives an mN message from each neighbor, so sL_p contains exactly the neighbors of p. ❏

By a similar argument, eventually each node p correctly has knowledge of N_p^2 as well as N_p. The same technique can enable each node to eventually have knowledge of N_p^3. In all subsequent sections, we use N_p^i for $i \in 1..3$ as constants in programs with the understanding that such neighborhood identification is actually obtained by the stabilizing protocol described above. In Section 5, we make use of slightly more information than N_p^3: we consider that we are aware of the local topology of p at distance three Γ_p^3; this information is to be acquired through the same means.

Building upon L_p, cached values of the shared variables of nodes in N_p^i, for $i \in 1..3$, can be maintained at p; erroneous cache values not associated with any node can be discarded by the aging technique. We use the following notation in the rest of the paper: for node p and some shared variable var_q of node $q \in N_p^3$, let ✉ var_q refer to the cached copy of var_q at p. The method of propagating cached copies of shared variables is generally self-stabilizing only for shared variables that eventually do not change value. With the exception of one algorithm presented in Section 3, all of our algorithms use cached shared variables in this way: eventually, the shared variables become constant, implying that eventually all cached copies of them will be coherent.

For algorithms developed in subsequent sections, we require a stronger property than eventual propagation of shared variable values to their caches. We require that with some constant probability, any shared variable will be propagated to its cached locations within constant time. This is tantamount to requiring that with constant probability, a node will transmit within constant time and the transmission will not collide with any other frame. Section 2 states our assumption on wireless transmission, based on the constants τ and α for collision-free, queued transmission. The discipline of shared variable propagation illustrated in Figure 3 spaces shared-variable updates by $\kappa + \beta$, where β is a random variable with an upper bound. It follows that there is a positive probability that any particular shared variable propagation attempt will succeed without collisions within a fixed time interval, and this positive probability is bounded below by some constant. Therefore the expected number of attempts to propagate a shared variable value before successfully writing to all its caches is $O(1)$. We henceforth assume that the expected time for shared variable propagation is constant.

3 Neighborhood Unique Naming

To this point, the architecture of the protocol being described has layered the neighbor identification protocol upon a protocol for shared variable propagation. In this section we propose to add another layer, which creates new names for nodes using a smaller space of names. This renaming layer relies on the output of the neighborhood identification protocol: it needs knowledge of N^3. Note that following a topology change or a transient failure, knowledge of N^3 could be faulty. Our renaming protocol can mistakenly perform many operations starting from such a state. Eventually, the neighbor identification protocol stabilizes and thereafter the renaming steps are meaningful. In subsequent sections also, we continue this layered approach to the full protocol construction. Section 7 shows how all parts are combined.

An algorithm providing neighborhood-unique naming gives each node a name distinct from any of its N^3-neighbors. This may seem odd considering that we already assume that nodes have unique identifiers, but when we try to use the identifiers for certain applications such as coloring, the potentially large namespace of identifiers can cause scalability problems. Therefore it can be useful to give nodes smaller names, from a constant space of names, in a way that ensures names are locally unique.

Our neighborhood unique naming algorithm is roughly based on the randomized technique described in [9], and introduces some new features. Define $\Delta = \lceil \delta^t \rceil$ for some $t > 3$; the choice of t to fix constant Δ has two competing motivations discussed at the end of this section. We call Δ the *namespace*. Let shared variable Id_p have domain $0..\Delta$; variable Id_p is the *name* of node p. A local function is used to collect the names of neighboring nodes: $Cids_p = \{ \text{✉} Id_q \mid q \in N_p^3 \setminus \{p\} \}$. Let $\mathsf{random}(S)$ choose with uniform probability some element of set S. Node p uses the following function to compute Id_p:

$$\mathsf{newId}(Id_p) = \begin{cases} Id_p & \text{if } Id_p \notin Cids_p \\ \mathsf{random}(\Delta \setminus Cids_p) & \text{otherwise} \end{cases}$$

The algorithm for unique naming is the following.

```
N3: $true \;\rightarrow\; Id_p := \mathsf{newId}(Id_p)$
```

Define $\mathsf{Uniq}(p)$ to be the predicate that holds iff (*i*) no name mentioned in $Cids_p$ is equal to Id_p, (*ii*) for each $q \in N_p^3$, $q \neq p$, $Id_q \neq Id_p$, (*iii*) for each $q \in N_p^3$, one name in $Cids_q$ equals Id_p, (*iv*) for each $q \in N_p^3$, $q \neq p$, the equality $\text{✉} Id_p = Id_p$ holds at node q, and (*v*) no cache update message *en route* to p conveys a name that would update $Cids_p$ to have a name equal to Id_p. Predicate $\mathsf{Uniq}(p)$ states that p's name is known to all nodes in N_p^3 and does not conflict with any name of a node q within N_q^3, nor is there a cached name liable to update $Cids_p$ that conflicts with p's name. A key property of the algorithm is the following: $\mathsf{Uniq}(p)$ is a stable property of the execution. This is because after $\mathsf{Uniq}(p)$ holds, any node q in N_p^3 will not assign Id_q to equal p's name, because N3 avoids names listed in the cache of distance-three neighborhood names – this stability property is not present in the randomized algorithm [9]. The property $(\forall r :\; r \in R :\; \mathsf{Uniq}(r))$ is similarly stable for any subset R of nodes. In words, once a name becomes established as unique for all the neighborhoods it belongs to, it is stable. Therefore we can reason about a Markov model of executions by showing that the probability of a sequence of steps moving, from one stable set of ids to a larger stable set, is positive.

Lemma 2 *Starting from any state, for any p, there is a constant, positive probability that* $\mathsf{Uniq}(p)$ *holds within constant time.*

Proof: The proof has three cases for p: (*a*) $\mathsf{Uniq}(p)$ holds initially, (*b*) $\neg\mathsf{Uniq}(p)$ holds, but p cannot detect this locally (this means that there exists some neighbor q of p such that $\text{✉} Id_p \neq Id_p$ at q); or (*c*) p detects $\neg\mathsf{Uniq}(p)$ and chooses a new name. Case (*a*) trivially verifies the lemma. For case (*b*), it could happen that $\mathsf{Uniq}(p)$ is established only by actions of nodes other than p within constant time, and the lemma holds; otherwise we rely on the periodic mechanism of cache propagation and the lower bound τ on the probability of collision-free transmission to reduce (*b*) to (*c*) with some constant probability within constant time. For case (*c*) we require a joint event, which is the following sequence: p chooses a name different from any name in N_p^3 and in their caches (or in messages *en route*), then p transmits the new name without collision to N_p, each node $q \in N_p$ transmits the cache of p's name without collision, and then each node in $N_p^2 \setminus N_p$ transmits the cache of p's name without collision. Fix some constant time Φ for this sequence of events; time Φ could be $(\delta + 1) \cdot \mu$, where μ is the average time for a cached value to be transmitted without collision. The joint probability x for this scenario is the product of probabilities for each event, with the constraint that the event is transmission without collision within the desired time constraint μ. This sequence is not enough, however to fully estimate the probability for case (*c*), because it could be that nodes of N_p^3 concurrently assign new identifiers, perhaps equal to p's name. Therefore we multiply by x the product of probabilities that each invocation of newId by $q \in N_p^3$ during the time period Φ does not return a name equal to p's name. Notice that the number of times that any $q \in N_p^3$ can invoke N3 is bounded by Φ/κ, because assignment to shared variables follows the discipline of at least κ delay. Thus the entire number of invocations of newId in the Φ-length time period is bounded by a constant. Therefore the overall joint probability is estimated by the product of x and a fixed number of constant probabilities; the joint probability for this scenario is thus bounded by a product of constant probabilities (dependent on Δ, δ, τ, $alpha$, and κ). Because this joint probability is bounded below by a nonzero constant, the expected number of trials to reach a successful result is constant. ❑

Corollary 1 *The algorithm N3 self-stabilizes with probability 1 and has constant expected local convergence time.*

Proof: The Markov chain for the algorithm has a trapping state for any p such that Uniq(p) holds. The stability of Uniq(p) for each p separately means that we can reason about self-stabilization for each node independently. The previous lemma implies that each node converges to Uniq(p) with probability 1, and also implies the constant overall time bound. ❑

Using the names assigned by N3 is a solution to $L(1,1)$ coloring, however using Δ colors is not the basis for an efficient TDMA schedule. The naming obtained by the algorithm does have a useful property. Let P be a path of t distinct nodes, that is, $P = p_1, p_2, \ldots, p_t$. Define predicate *Up*$(P)$ to hold if $id_{p_i} < id_{p_j}$ for each $i < j$. In words, *Up*(P) holds if the names along the path P increase.

Lemma 3 *Every path P satisfying Up(P) has fewer than $\Delta + 1$ nodes.*

Proof: If a path P satisfying *Up*(P) has $\Delta + 1$ nodes, then some name appears at least twice in the path. The ordering on names is transitive, which implies that some name a of a node in P satisfies $a < a$, and this contradicts the total order on names. ❑

This lemma shows that the simple coloring algorithm gives us a property that node identifiers do not have: the path length of any increasing sequence of names is bounded by a constant. Henceforth, we suppose that node identifiers have this property, that is, we treat N_p^i as if the node identifiers are drawn from the namespace of size Δ.

There are two competing motivations for tuning the parameter t in $\Delta = \delta^t$. First, t should be large enough to ensure that the choice made by newId is unique with high probability. In the worst case, $|N_p^3| = \delta^3 - 2\delta^2 + \delta + 1$, and each node's cache can contain about δ^3 names, so choosing $t \approx 6$ could be satisfactory. Generally, larger values for t decrease the expected convergence time of the neighborhood unique naming algorithm. On the other hand, smaller values of t will reduce the constant Δ, which will reduce the convergence time for algorithms described in subsequent sections.

4 Leaders via Maximal Independent Set

Simple distance two coloring algorithms may use a number of colors that is wastefully large. Our objective is to find an algorithm that uses a reasonable number of colors and completes, with high probability, in constant time. We observe in this section that an assignment to satisfy distance two coloring can be done in constant time given a set of neighborhood leader nodes distributed in the network. In Section 5, it is shown how the leaders dictate coloring for nearby nodes. The coloring enabled by this method is minimal (not minimum, which is an NP-hard problem).

An algorithm selecting a maximal independent set is our basis for selecting the leader nodes. Selecting which nodes are leaders can be thought of as an iteration of stages. In the first stage, nodes with locally minimum names, which have been assigned from the Δ-size namespace by the randomized protocol of Section 3, become leaders. The second stage makes leaders of those nodes with the smallest names that are neither leaders in the first phase nor neighbor to a first-phase leader. Remaining stages continue in this fashion. The self-stabilizing algorithm presented below does not use explicit stages, though the notion of stages helps structure the proof of correctness.

Let each node p have a boolean shared variable ℓ_p. In an initial state, the value of ℓ_p is arbitrary. A legitimate state for the algorithm satisfies $(\forall p :\ p \in V :\ \mathcal{L}_p)$, where

$$\begin{array}{rll}\mathcal{L}_p & \equiv & (\ell_p \Rightarrow (\forall q :\ q \in N_p :\ \neg \ell_q)) \\ & \wedge & (\neg \ell_p \Rightarrow (\exists q :\ q \in N_p :\ \ell_q))\end{array}$$

Thus the algorithm should elect one leader (identified by the ℓ-variable) for each neighborhood. As in previous sections, $\boxtimes\,\ell_p$ denotes the cached copy of the shared variable ℓ_p. Note that N_p, treated here as a constant, is in fact an output of the neighborhood identification protocol given in Section 2.4, upon which the naming and leader selection protocols are based.

```
R1: $(\forall q :\ q \in N_p :\ q > p) \ \rightarrow\ \ell_p := \mathit{true}$
R2: $([]\, q :\ q \in N_p :\ \boxtimes\,\ell_q \ \wedge\ q < p \ \rightarrow\ \ell_p := \mathit{false})$
R3: $(\exists q :\ q \in N_p :\ q < p) \ \wedge\ (\forall q :\ q \in N_p \wedge\ (q > p \ \vee\ \neg\boxtimes\,\ell_q)) \ \rightarrow\ \ell_p := \mathit{true}$
```

Although the algorithm does not use randomization, its convergence technically remains probabilistic because our underlying model of communication uses CSMA/CA based on random delay. The algorithm's progress is therefore guaranteed with probability 1 rather than by deterministic means.

Lemma 4 *With probability 1 the algorithm R1-R3 converges to a solution of maximal independent set; the convergence time is $O(1)$ if each timed variable propagation completes in $O(1)$ time.*

Proof: We prove by induction on the namespace that each node p stabilizes its value of ℓ_p within $O(\Delta)$ time. For the base case, consider the set S of nodes with locally minimum names, that is, $(\forall p, q :\ p \in S\ \wedge\ q \in N_p :\ p < q)$. Any node $p \in S$ stabilizes in $O(1)$ time to $\ell_p = \mathit{true}$. The claim follows from the fact that guards of R2 and R3 are *false*, whereas the guard of R1 is permanently *true*. Therefore for the induction step, we can ignore R1, as it is dealt with in the base case.

To complete the induction, suppose that each node r has stabilized the value of ℓ_r, where $r \leq k$. Now consider the situation of a node p with name $k+1$ (if there is no such node, the induction is trivially satisfied). As far as the guards of R2 and R3 are concerned, the value of ℓ_q is only relevant for a neighbor q with $q < p$, and for any such neighbor, ℓ_q is stable by hypothesis. Since guards of R2 and R3 are exclusive, it follows that p stabilizes ℓ_p and $\text{✉}\,\ell_p$ is propagated within $O(1)$ time.

Finally, we observe that in any fixed point of the algorithm R1–R3, no two neighbors are leaders (else R2 would be enabled for one of them), nor does any nonleader find no leader in its neighborhood (else R1 or R3 would be enabled). This establishes that $\mathcal{L}_p$ holds at a fixed point for every $p \in V$. The induction terminates with at most $|\Delta|$ steps, the size of the namespace, and because Δ is a constant, the convergence time is $O(1)$ for this algorithm. ❑

5 Leader Assigned Coloring

Our method of distance-two coloring is simple: colors are assigned by the leader nodes given by maximal independent set output. The following variables are introduced for each node p:

color_p is a number representing the color for node p.

$\mathit{min}\ell_p$ is meaningful only for p such that $\neg\ell_p$ holds: it is intended to satisfy

$$\mathit{min}\ell_p\ =\ \min\ \{\ q \mid q \in N_p\ \wedge\ \text{✉}\,\ell_q\ \}$$

In words, *min*ℓ_p is the smallest id of any neighbor that is a leader. Due to the uniqueness of names in N_p^3, the value *min*ℓ_p stabilizes to a unique node.

$\mathit{spectrum}_p$ is a set of pairs (c, r) where c is a color and r is an id. Pertaining only to nonleader nodes, $\mathit{spectrum}_p$ should contain $(\mathit{color}_p, \mathit{min}\ell_p)$ and $(\text{✉}\,\mathit{color}_q,\ \text{✉}\,\mathit{min}\ell_q)$ for each $q \in N_p$.

$setcol_p$ is meaningful only for p such that ℓ_p holds. It is an array of colors indexed by identifier: $setcol_p[q]$ is p's preferred color for $q \in N_p$. We consider $color_p$ and $setcol_p[p]$ to be synonyms for the same variable. In the algorithm we use the notation $setcol_p[A] := B$ to denote the parallel assignment of a set of colors B based on a set of indices A. To make this assignment deterministic, we suppose that A can be represented by a sorted list for purposes of the assignment; B is similarly structured as a list.

dom_p for leader p is computed to be the nodes to which p can give a preferred color; these include any $q \in N_p$ such that $min\ell_q = p$. We say for $q \in dom_p$ that *p dominates q*.

f is a function used by each leader p to compute a set of unused colors to assign to the nodes in dom_p. The set of *used* colors (along with their associated node) for p is

$$\{\, (c,s) \mid (\exists\, q,r :\ q \in N_p^2 \ \wedge\ s \in N_p^3 \ \wedge\ \boxtimes color_s \ =\ c \ \wedge\ (c,r) \in \boxtimes spectrum_q \ \wedge\ r < p) \,\}$$

That is, used colors with respect to p are those colors in N_p^3 that are already assigned by leaders with smaller identifiers than p. The complement of the *used* set is the range of possible colors that p may prefer for nodes it dominates. Let f be the function to minimize the number of colors preferred for the nodes of dom_p, ensuring that the colors for dom_p are distinct, and assigning smaller color indices (as close to 0 as possible) preferentially, using the topology at distance 3. Function f returns a list of colors to match the deterministic list of dom_p in the assignment of R5.

```
R4: $\ell_p \ \rightarrow\ dom_p := \{\, p \,\} \cup \{\, q \mid q \in N_p \wedge \boxtimes min\ell_q = p \,\}$
R5: $\ell_p \ \rightarrow\ setcol_p[dom_p] := f(\{\, (c,s) \mid$
      $(\exists\, q,r :\ q \in N_p^2 \ \wedge\ s \in N_p^3 \ \wedge\ \boxtimes color_s \ =\ c \ \wedge\ (c,r) \in \boxtimes spectrum_q \ \wedge\ r < p) \,\})$
R6: $true \ \rightarrow\ min\ell_p := \min\{\ q \mid q \in N_p \cup \{p\} \ \wedge\ \boxtimes \ell_q \,\}$
R7: $\neg\ell_p \ \rightarrow\ color_p := \boxtimes setcol_r[p]$, where $r = min\ell_p$
R8: $\neg\ell_p \ \rightarrow\ spectrum_p := (color_p, min\ell_p) \ \cup\ \bigcup\{\ (c,r) \mid$
      $(\exists\, q,c,r :\ q \in N_p :\ c = \boxtimes color_q \ \wedge\ r = \boxtimes min\ell_q) \,\}$
```

Lemma 5 *The algorithm R4-R8 converges to a distance-two coloring, with probability 1; the convergence time is $O(1)$ if each timed variable propagation completes in $O(1)$ time.*

Proof: The proof is a sequence of observations to reflect the essentially sequential character of color assignment. We consider an execution where the set of leaders has been established by R1–R3 initially. Observe that in $O(1)$ time the assignments of R6 reach a fixed point, based on the local reference to $\boxtimes \ell_q$ for neighbors. Therefore, in $O(1)$ time, the shared variables $min\ell_p$ are propagated to N_p and caches $\boxtimes min\ell_p$ are stable. Similarly, in $O(1)$ additional time, the assignments of R4 reach a fixed point, so that leaders have stable *dom* variables.

The remainder of the proof is an induction to show that color assignments stabilize in $O(\Delta)$ phases (recall that Δ is the constant of Lemma 3). For the base case of the induction, consider the set S of leader nodes such that for every $p \in S$, within N_p^3 no leader of smaller name than p occurs. We use distance three rather than distance two so that such a leader node's choice of colors is stable, independent of the choices made by other leaders. Set S is non-empty because, of the set of leaders in the network, at least one has minimal name, which is unique up to distance three. Call S the set of *root* leaders. Given such a leader node p, each neighbor $q \in N_p$ executes R8 within $O(1)$ time and assigns to $\mathit{spectrum}_q$ a set of tuples with the property that for any $(c, r) \in \mathit{spectrum}_q$, $r \geq p$. Notice that although $\mathit{spectrum}_q$ could subsequently change in the course of the execution, this property is stable. Therefore, in $O(1)$ additional time, no tuple of ✉$\mathit{spectrum}_q$ has a smaller value than p in its second component. It follows that any subsequent evaluation of R5 by leader p has a fixed point: p assigns colors to all nodes of N_p. After $O(1)$ delay, for $q \in N_p$, ✉setcol_p stabilizes. Then in $O(1)$ time, all nodes of dom_p assign their *color* variables using R7. This completes the base case, assignment of colors by root leaders.

We complete the induction by examining nodes with minimum distance $k > 0$ from any root leader along a path of increasing leader names (referring to the *Up* predicate used in Lemma 3). The hypothesis for the induction is that nodes up to distance $k - 1$ along an increase path of leader names have stabilized to a permanent assignment of colors to the nodes they dominate. Arguments similar to the base case show that such nodes at distance k eliminate colors already claimed by leaders of the hypothesis set in their evaluations of R5. The entire inductive step — extending by one all paths of increasing names from the root leaders — consumes $O(1)$ additional time. The induction terminates at Δ, thanks to Lemma 3, hence the overall bound of $O(\Delta)$ holds for convergence. ❑

Only at one point in the proof do we mention distance-three information, which is to establish the base case for root leaders (implicitly it is also used in the inductive step as well). Had we only used neighborhood naming unique up to distance two, it would not be ensured that a clear ordering of colors exists between leaders that compete for dominated nodes, *eg*, a leader p could find that some node $r \in N_p^2$ has been assigned a color by another leader q, but the names of p and q are equal; this conflict would permit q to assign the same color to r that p assigns to some neighbor of r. We use distance-three unique naming to simplify the presentation, rather than presenting a more complicated technique to break ties. Another useful intuition for an improved algorithm is that Lemma 3's result is possibly stronger than necessary: if paths of increasing names have at most some constant length d with high probability, and the algorithms for leader selection and color assignment tolerate rare cases of naming conflicts, the expected convergence time would remain $O(1)$ in the construction.

A vertex coloring at distance two is locally minimal if any change of a color at one node p in the network (that preserves a proper distance two vertex coloring) does not reduce the number of colors in N_p^2.

Lemma 6 *The distance two coloring achieved by Algorithm R4-R8 is minimal.*

Proof: Assume for contradiction that there exists a node p such that changing p's color reduces the number of colors used in N_p^2. There are two cases depending whether p is a leader or not. If p is a leader, then it chose its color in such a way that it was minimal considering the input of its two-neighborhood, so p's color cannot be changed in a way that reduces the number of colors in N_p^2. If p is not a leader, p's color was given by a neighbor q of p that is a leader (if p is not a leader, it has a leader in N_p). Now, q is aware of the topology and color information at distance three from itself, and thus is aware of the topology and color information at distance two from p. So, q chose p's color so that it is minimal considering N_p^2. As a result, p's color cannot be changed in a way that reduces the number of colors in N_p^2 in this case either. Thus a contradiction. ❑

6 Assigning Time Slots from Colors

Given a distance-two coloring of the network nodes, the next task is to derive time slot assignments for each node for TDMA scheduling. Our starting assumption is that each node has equal priority for assigning time slots, *i.e.*, we are using an unweighted model in allocating bandwidth. Before presenting an algorithm, we have two motivating observations.

First, the algorithms that provide coloring are local in the sense that the actual number of colors assigned is not available in any global variable. Therefore to assign time slots consistently to all nodes apparently requires some additional computation. In the first solution of Figure 1, both leftmost and rightmost nodes have color 1, however only at the leftmost node is it clear that color 1 should be allocated one ninth of the time slots. Local information available at the rightmost node might imply that color 1 should have one third of the allocated slots.

The second observation is that each node p should have about as much bandwidth as any other node in N_p^2. This follows from our assumption that all nodes have equal priority. Consider the number of colors in the distance two neighborhood shown in Figure 2 that correspond to the colorings of Figure 1. The rightmost node p in the second coloring has three colors in its two-neighborhood, but has a neighbor q with four colors in its two-neighborhood. It follows that q shares bandwidth with four nodes: q's share of the bandwidth is at most $1/4$, whereas p's share is at most $1/3$. It does not violate fairness to allow p to use $1/3$ of the slot allocation if these slots would otherwise be wasted. Our algorithm therefore allocates slots in order from most constrained (least bandwidth share) to least constrained, so that extra slots can be used where available.

To describe the algorithm that allocates media access time for node p, we introduce these shared variables and local functions.

$base_p$ stabilizes to the number of colors in N_p^2. The value $base_p^{-1} = 1/base_p$ is used as a constraint on the share of bandwidth required by p in the TDMA slot assignment.

$itvl_p$ is a set of intervals of the form $[x, y)$ where $0 \leq x < y \leq 1$. For allocation, each unit of time is divided into intervals and $itvl_p$ is the set of intervals that node p can use to transmit messages. The expression $|[x, y)|$ denotes the time-length of an interval.

$g(b, S)$ is a function to assign intervals, where S is a set of subintervals of $[0, 1)$. Function $g(b, S)$ returns a maximal set T of subintervals of $[0, 1)$ that are disjoint and also disjoint from any element of S such that $(\sum_{a \in T} |a|) \leq b$.

To simplify the presentation, we introduce S_p as a private (nonshared) variable.

```
R9:  $\mathit{true} \rightarrow \mathit{base}_p := |\{ \boxtimes\, \mathit{color}_q \mid q \in N_p^2 \}|$
R10: $\mathit{true} \rightarrow S_p := \bigcup \{ \boxtimes\, \mathit{itvl}_q \mid q \in N_p^2 \ \wedge$
     $(\boxtimes\, \mathit{base}_q > \mathit{base}_p \ \vee$
     $(\boxtimes\, \mathit{base}_q = \mathit{base}_p \ \wedge \ \boxtimes\, \mathit{color}_q < \mathit{color}_p)) \}$
R11: $\mathit{true} \rightarrow \mathit{itvl}_p := g(\mathit{base}_p^{-1}, S_p)$
```

Lemma 7 *With probability 1 the algorithm* **R9–R11** *converges to an allocation of time intervals such that no two nodes within distance two have conflicting time intervals, and the interval lengths for each node p sum to $|\{ \mathit{color}_q \mid q \in N_p^2 \}|^{-1}$; the expected convergence time of* **R9-R11** *is $O(1)$ starting from any state with stable and valid coloring.*

Proof: Similar to that for Lemma 5; we omit details. ❑

Definition 1 *The TDMA bandwith B_p of node p refers to the sum of the sizes of each element of itvl_p.*

Lemma 8 *Consider a graph of degree with minimal distance two coloring, then the TDMA bandwith B_p allocated by Algorithm* **R9-R11** *to each node p is at least $1/(|C_p^2|)$, where C_p^2 denotes the set of colors in the distance two neighborood of p.*

Proof: Assume the contrary. We consider a stable configuration where for every node p, $\mathit{base}_p = |C_p^2|$, and where every shared variable is correctly cached up to distance 3. By hypothesis, there is one node p that is allocated less than $1/|C_p^2|$ TDMA bandwidth. Then, the set S_p that is calculated by p would represent more than a $1 - 1/|C_p^2|$ portion of the $[0, 1)$ interval (By Lemma 7). In turn, this implies that there would exist a node q in N_p^2 such that it has higher priority than p to allocate TDMA bandwith and holds more than a $1/|C_p^2|$ portion of the TDMA bandwidth. This node q satisfies one of the following predicates: *(i)* $\mathit{base}_q > \mathit{base}_p$ (with $\mathit{base}_q = |C_q^2|$), or *(ii)* $\mathit{base}_q = \mathit{base}_p \wedge \mathit{color}_q < \mathit{color}_p$. Case *(i)* is impossible because then q would have allocated *less* bandwith than p (since it is more constrained than p, the upper bound $|C_q^2|$ given to the g function is stricly lower than $|C_p^2|$). Case *(ii)* is also impossible because then q would have allocated *as much* bandwith as p (since it is as constrained as p, the upper bound $|C_q^2|$ given to the g function is equal to $|C_p^2|$). So, in any case, there remains at node p at least $1/|C_p^2|$ of the bandwith to consume. ❑

Lemma 9 *The TDMA bandwith B_p that is allocated to p satisfies:*

$$\frac{1}{|C_p^2|} \leq B_p \leq 1 - \sum_{j \in N_p} B_j$$

Proof: The first part of the inequality is given by Lemma 8. The second part of the inequality comes from Lemma 7. ❑

It can be verified of R9-R11 that, at a fixed point, no node $q \in N_p^2$ is assigned a time that overlaps with interval(s) assigned to p; also, all available time is assigned (there are no left-over intervals). A remaining practical issue is the conversion from intervals to a time slot schedule: a discrete TDMA slot schedule will approximate the intervals calculated by g. We do not address this issue of conversion in the paper.

7 Assembly

Given the component algorithms of Sections 2.4–6, the concluding statement of our result follows.

Theorem 1 *The composition of* N0–N3 *and* R1–R11 *is a probabilistically self-stabilizing solution to $\mathcal{T}$ with $O(1)$ expected local convergence time.*

Proof: The infrastructure for neighborhood identification and shared variable propagation N0–N2 contributes $O(1)$ delay (partly by assumption on the CSMA/CA behavior), and N3 establishes neighborhood unique naming in expected $O(1)$ time. For any node p, the leader protocol layered above N3 assumes that N_p is constant, meaning that each neighbor of p has a stable name. The convergence bound must account for the expected time that all names in p's neighborhood are stable. For each of $q \in N_p$, the expected time to obtain a stable name is $O(1)$, and since $|N_p| = O(1)$, the expected time for the event of a stably named neighborhood has an $O(1)$ time bound. More generally, the size of extended neighborhood of p, say N_p^{Δ} also stabilizes within $O(1)$ expected time.

The subsequent layers R1–R3, R4–R8, and R9–R11, each have $O(1)$ convergence time, and each layer is only dependent on the output of the previous layer. None of these layers is randomized, depending only on the assumption of constant time shared variable propagation for its expected convergence time. The hierarchical composition theorem (see [26]) implies overall stabilization, and the expected convergence time is the sum of the expected convergence times of the components. ❑

8 Conclusion

Sensor networks differ in characteristics and in typical applications from other large scale networks such as the Internet. Sensor networks of extreme scale (hundreds of thousands to millions of nodes) have been imagined [14], motivating scalability concerns for such networks. The current generation of sensor networks emphasizes the *sensing* aspect of the nodes, so services that aggregate data and report data have been emphasized. Future generations of sensor networks will have significant actuation capabilities. In the context of large scale sensor/actuator networks, end-to-end services can be less important than regional and local services. Therefore we emphasize local stabilization time rather than global stabilization time in this paper, as the local stabilization time is likely to be more important for scalability of TDMA than global stabilization time. Nonetheless, the question of global stabilization time is neglected in previous sections. We speculate that global stabilization time will be sublinear in the diameter of the network (which could be a different type of argument for scalability of our constructions, considering that end-to-end latency would be linear in the network diameter even after stabilization). Some justification for our speculation is the following: if the expected local time for convergence is $O(1)$ and underlying probability assumptions are derived from Bernoulli (random name selection) and Poisson (wireless CSMA/CA) distributions, then these distributions can be approximately bounded by exponential distributions with constant means. Exponential distributions define half-lives for populations of convergent processes (given asymptotically large populations), which is to say that within some constant time γ, the expected population of processes that have not converged is halved; it would follow that global convergence is $O(\lg n)$.

We close by mentioning two important open problems. Because sensor networks can be deployed in an *ad hoc* manner, new sensor nodes can be dynamically thrown into a network, and mobility is also possible, the TDMA algorithm we propose could have a serious disadvantage: introduction of just one new node could disrupt the TDMA schedules of a sizable part of a network before the system stabilizes. Even if the stabilization time is expected to be $O(1)$, it may be that better algorithms could contain the effects of small topology changes with less impact than our proposed construction. One can exploit normal notifications of topology change as suggested in [5], for example.

In the protocols we considered, only the last protocol (that allocates time slots according to the minimal distance two coloring) explicitly refers to a global time mechanism. So, another interesting question is whether the assumption of globally synchronized clocks (often casually defended by citing GPS availability in literature of wireless networks) is really needed for self-stabilizing TDMA construction; we have no proof at present that global synchronization is necessary.

References

[1] D. Bertsekas and R. Gallager. *Data Networks*, Prentice-Hall, 1987.

[2] H. L. Bodlaender, T. Kloks, R. B. Tan, and J. van Leeuwen. Approximations for λ-coloring of graphs. University of Utrecht, Department of Computer Science, Technical Report 2000-25, 2000 (25 pages).

[3] C. Busch, M. Magdon-Ismail, F. Sivrikaya, and B. Yener. Contention-free MAC protocols for wireless sensor networks. In *Proceedings of 18th International Conference of Dis-*

tributed Computing (DISC), Amsterdam, The Netherlands, October 4-7, 2004. Springer Lecture Notes in Computer Science 3274, pp. 245-259.

[4] D. E. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo. A network-centric approach to embedded software for tiny devices. In *Proceedings of Embedded Software, First International Workshop EMSOFT 2001*, Springer LNCS 2211, pp. 114-130, 2001.

[5] S. Dolev and T. Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago Journal of Theoretical Computer Science*, 3(4), 1997.

[6] M. Gairing, W. Goddard, S. T. Hedetniemi, P. Kristiansen, and A. A. McRae. Distance-Two Information in Self-Stabilizing Algorithms. *Parallel Processing Letters*, 14(3), 387-398, 2004

[7] S. Gandham, M. Dawande, and R. Prakash. Link scheduling in sensor networks: distributed edge coloring revisited. INFOCOM 2005 (to appear).

[8] S. Ghosh and M. H. Karaata. A self-stabilizing algorithm for coloring planar graphs. Distributed Computing 7:55-59, 1993.

[9] M. Gradinariu and C. Johnen. Self-stabilizing neighborhood unique naming under unfair scheduler. In *Euro-Par'01 Parallel Processing, Proceedings,* Springer LNCS 2150, 2001, pp. 458-465.

[10] M. Gradinariu and S. Tixeuil. Self-stabilizing vertex coloration of arbitrary graphs. In *4th International Conference On Principles Of DIstributed Systems, OPODIS'2000*, 2000, pp. 55-70.

[11] M. Haenggi and X. Liu. Fundamental throughput limits in Rayleigh fading sensor networks. *In submission*, 2003.

[12] T. Herman. Models of self-stabilization and sensor networks. *Fifth International Workshop on Distributed Computing (IWDC 2003)*, Springer Lecture Notes in Computer Science LNCS 2918, December 2003, pp. 205-214.

[13] T. Herman and S. Tixeuil. A distributed tdma slot assignment algorithm for wireless sensor networks. In *Proceedings of the First Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors'2004)*, number 3121 in Lecture Notes in Computer Science, pages 45-58, Turku, Finland, July 2004. Springer-Verlag.

[14] J. Kahn, R. Katz, and K. Pister. Next century challenges: mobile networking for "smart dust". In *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking (MOBICOM '99)*, 1999.

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

[16] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. In *Proceedings of the Twenty-Second ACM Symposium on Principles of Distributed Computing*, (PODC 2003), pp. 25-32, 2003.

[17] S. S. Kulkarni and U. Arumugam. Collision-free communication in sensor networks. In *Proceedings of Self-Stabilizing Systems, 6th International Symposium*, Springer LNCS 2704, 2003, pp. 17-31.

[18] S. S. Kulkarni and U. Arumugam. Transformations for Write-All-With-Collision Model. In *Proceedings of the 7th International Conference on Principles of Distributed Systems (OPODIS)*, Springer LNCS, 12/03. (Martinique, French West Indies, France).

[19] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing* 15:1036-1053, 1986.

[20] M. Mizuno and M. Nesterenko. A transformation of self-stabilizing serial model programs for asynchronous parallel computing environments. *Information Processing Letters* 66 (6 1998):285-290.

[21] S. Ramanathan. A unified framework and algorithm for channel assignment in wireless networks. *Wireless Networks* 5(2 1999):81-94.

[22] S. Ramanathan and E. L. Lloyd. Scheduling algorithms for multi-hop radio networks. *IEEE/ACM Transactions on Networking*, 1(2 1993):166-177.

[23] S. Shukla, D. Rosenkrantz, and S. Ravi. Developing self-stabilizing coloring algorithms via systematic randomization. In *Proceedings of the International Workshop on Parallel Processing*, pages 668–673, Bangalore, India, 1994. Tata-McGrawhill, New Delhi.

[24] S. Shukla, D. Rosenkrantz, and S. Ravi. Observations on self-stabilizing graph algorithms for anonymous networks. In *Proceedings of the Second Workshop on Self-stabilizing Systems (WSS'95)*, pages 7.1–7.15, 1995.

[25] S. Sur and P. K. Srimani. A self-stabilizing algorithm for coloring bipartite graphs. *Information Sciences*, 69:219–227, 1993.

[26] G. Tel. *Introduction to Distributed Algorithms*, Cambridge University Press, 1994.

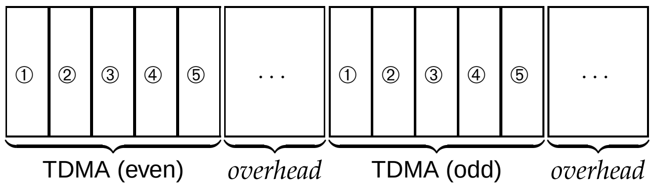
[27] A. Woo and D. Culler. A transmission control scheme for media access in sensor networks. In *Proceedings of the Seventh International Conference on Mobile Computing and Networking (Mobicom 2001)*, pp. 221-235, 2001.

[28] F. Zhao and L. Guibas (Editors). *Proceedings of Information Processing in Sensor Networks, Second International Workshop, IPSN 2003*, Springer LNCS 2634. April, 2003.

A Model Emulation

The paper [18] is the first to consider various theoretical models commonly used in the literature of self-stabilization in the context of sensor network models. In fact, TDMA is one technique proposed by [18], and one application of our mechanism is to provide a transformer that takes as input a traditional self-stabilizing algorithm (written for the shared memory model with reliable atomic neighborhood communications) and gives as output a self-stabilizing algorithm written for the wireless sensor network model with probabilistically unreliable communications for the same problem. The stabilization time would be the same (up to a constant factor). This is to be compared with the alternative transformer by Herman [12], that, although simpler, does not provide any bound of the cost of the induced transformation.

Two possible implementations of this transformer with our algorithm are as follows. For the distributed daemon (at each step, any subset of the activatable nodes is scheduled for execution), we divide the TDMA schedule in two parts, the even parts and the odd ones.



The even parts of the TDMA are used to propagate shared variables between nodes. Since our algorithms provide a collision-free mechanism, it is guaranteed that shared variables are transmitted among neighboring nodes in one TDMA part. The odd parts of the TDMA schedule are used to execute rules of the upper algorithm as if in the shared memory model. Since shared variables are accurate at this point, the semantics of the original algorithm are preserved, and the stabilization time is expanded to a constant factor of 2. Interestingly enough, if nodes are able to execute the rules of the upper layer algorithm during the overhead phase (as they require no communication), the stabilization time remains the same as the original one.

For the locally central demon (at each step, a subset of non-neighboring nodes is scheduled for execution), we divide the TDMA schedule in $2 \times \delta^2$ parts, and number TDMA parts from $0 \bmod 2 \times \delta^2$. The parts of the TDMA that are even mod $2 \times \delta^2$ are used to propagate shared variables between nodes. The other parts of the TDMA schedule are used to execute rules of the upper layer algorithm as if in the shared memory model. A node having color c only executes at TDMA part $2 \times c + 1 \bmod 2 \times \delta^2$. Since shared variables are accurate each time a node executes its upper algorithm, the semantics of the original algorithm are preserved. Since δ^2 is a constant, the stabilization time is expanded up to a constant factor. Note that algorithms written in the shared memory model assuming the central demon (at each step, a single activatable node is scheduled for execution) also perform correctly using the locally central demon. Both transformers preserve the self-stabilizing properties of the original algorithm, and its asymptotic stabilization time.

This scheme can be further refined if the algorithm to be transformed expects to be able to read the shared variables of its neighborood up to distance k (whre k is some constant greater than 1). For example, [6] shows that problems such as 2-packing can be solved in a straighforward maneer if distance two information is available. We simply divide the TDMA schedule in $k+1$ parts. The first k parts are used to collect information k hops away, and the $k+1^{\text{th}}$ part is used to actually execute rules of the upper layer algorithm as if in a distance k shared memory model.