RAPPORT DE RECHERCHE

L R I

A DISTRIBUTED TDMA SLOT ASSIGNMENT ALGORITHM FOR WIRELESS SENSOR NETWORKS

HERMAN T / TIXEUIL S

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud-LRI

09/2003

Rapport de Recherche N° 1370

CNRS – Université de Paris Sud
Centre d'Orsay
LABORATOIRE DE RECHERCHE EN INFORMATIQUE
Bâtiment 650
91405 ORSAY Cedex (France)

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 Équipe Grand Large,
Université Paris-Sud XI
F-91405 Orsay cedex, France
Sebastien.Tixeuil@lri.fr

September 1, 2003

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 convergence time is $O(1)$ for any size network satisfying a constant bound on the size of a node neighborhood.

Résumé

Les réseaux sans fils de senseurs bénéficient de protocoles de communication qui évitent les collisions de trames utilisées pour la communication. Les méthodes utilisant le multiplexage temporel pour l'accès au media (TDMA) planifient les transmissions pour éviter les collisions, cependant ces méthodes souffrent d'un problème de dimensionnement quand elles sont utilisées dans des réseaux *ad hoc* à topologie dynamique et sujets à des défaillances de nœuds. Cet article propose un algorithme réparti pour la planification TDMA qui est auto-stabilisant aux défaillances transitoires et aux changements dynamiques de topologie. Le temps de convergence attendu est $O(1)$ indépendement de la taille du réseau si celui-ci admet une borne constante sur la taille du voisinage de chaque nœud.

Chapter 1

Introduction

Collision management and avoidance are fundamental issues in wireless network protocols. Networks now being imagined for sensors [9] and small devices [8] require energy conservation, scalability, tolerance to transient faults, and adaptivity to topology change. Communication protocols that avoid collisions may indirectly conserve energy, because the need for message retransmission is reduced. Time Division Media Access (TDMA) is a reasonable technique for avoiding collisions, however the priorities of scalability and fault tolerance are not emphasized by most previous research.

The algorithmic problem of allocating time slots for TDMA is related to the well-studied problem of allocating frequencies in FDMA. These problems can be formulated as constrained vertex coloring problems in a graph [5]. For FDMA, each color represents a frequency and the basic constraint for collision avoidance is to ensure that no pair of vertices at distance two or less have the same color. A further engineering constraint is to allocate colors so that the frequencies of neighboring vertices are separated far enough apart to avoid interference. Let the set of vertex colors be the integers from the range $[0, \lambda]$; then colors (f_v, f_w) of neighboring vertices (v, w) should satisfy $|f_v - f_w| > 1$ to avoid interference. The standard 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 for FDMA thus should satisfy the $L(2, 1)$ constraint. Furthermore, a solution using the fewest number of colors is desirable, since this reduces the number of frequencies needed.

The graph coloring problem for TDMA is slightly different from that of FDMA. 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 $\ell_i \bmod (\lambda + 1)$. This constraint represents the fact that time slots wrap around, unlike frequencies. (a deeper explanation for the wrap around of time slots is given in Chapter 3). The usual coloring constraint for TDMA is $L'(1, 1)$. If time slots are imprecise (perhaps due to imperfect time synchronization), one could ask for a stricter separation of colors, for instance $L'(2, 2)$ could be a desired constraint. In this paper, we confine attention to $L'(1, 1)$, which is equivalent to $L(1, 1)$. (Our algorithms can be extended to satisfy $L'(2, 2)$.) Minimizing the number of colors for TDMA is desirable because, if a time period corresponding to the sequence of colors $0..\lambda$ is normalized to the unit interval $[0, 1]$, then each color represents

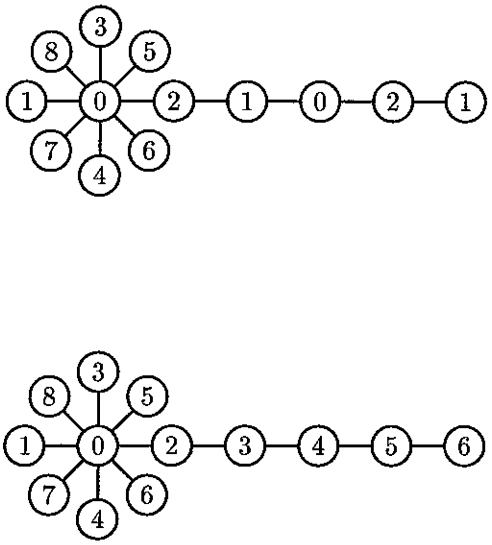


Figure 1.1: two solutions to distance-two coloring

a $1/(\lambda + 1)$ fraction of bandwidth; a smaller value of λ gives each node a larger share of bandwidth.

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, 3, 4]; many such problems are NP-complete and although approximation algorithms have been proposed, such algorithms are typically not distributed.

Even a solution to minimum coloring does not necessarily give the best result for TDMA slot assignment. Consider the two colorings shown in Figure 1.1, which are minimum $L(1,1)$ 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 1.2 for the respective colorings of Figure 1.1. We see that some of the nodes find more colors in their distance-two neighborhoods in the second coloring of Figure 1.1. We will see in Chapter 8 that the solution with fewer nodes in distance-two neighborhoods is preferable. 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. Of course, the problem of getting such an interesting coloring for TDMA is also NP-complete.

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 the system state converges to a valid TDMA assignment after any transient fault or topology change event. Our approach to scalability is to propose a

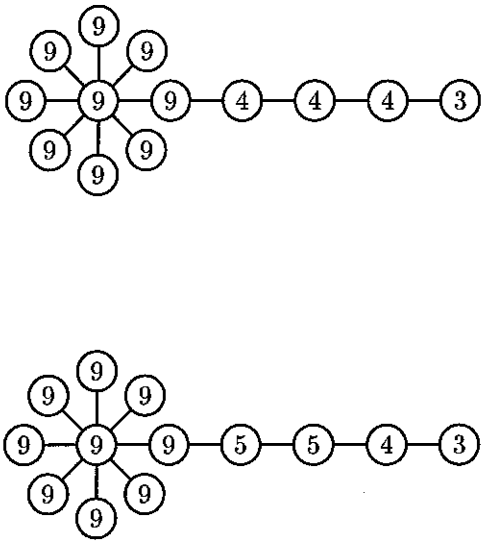


Figure 1.2: number of colors used within distance two

randomized slot assignment algorithm with $O(1)$ expected convergence time. The basis for our algorithm is, in some way, a probabilistically fast clustering technique which could be exploited for other problems of sensor networks.

Related Work. The work of Kulkarni and Arumugam [1] is the first to develop self-stabilizing TDMA. That paper starts from the view of a grid topology (and is applicable to general networks by mapping them to grids) and supposes each node is given knowledge of its location in the grid; knowing the location in grid can be used for generating a TDMA schedule. Also, using the approach of [1] in general networks requires the grid mapping be the same for all nodes and known before the TDMA algorithm is started. Thus their apporach does not suit dynamic networks. By contrast, the algorithms of this paper are for general networks, and do not require a node to have location information.

In the self-stabilizing setting, the most studied vertex coloring problem is $L(1,0)$. Gosh and Karaata [13] provide an elegant solution for coloring planar graphs, Sur and Srimani's [16] solution to the same problem is for bipartite graphs, Shukla *et al.* [14, 15] provide algorithms for complete odd-degree bipartite graphs and tree graphs, while a solution for general graphs is given by Gradinariu and Tixeuil [7]. Recently, Gradinariu and Johnen [6] presented a solution to the $L(1,1)$ coloring, using a number of colors proportional to n^2, where n denotes the number of nodes in the network. A common drawback to all previous approaches is that a reliable and powerful communication system is assumed: in one atomic step, a node is able to read the current state of every of its neighbors, and perform actions accordingly.

Our algorithms borrow from techniques of self-stabilizing coloring and renaming [6, 7]. 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.

Paper Organization. Chapter 2 explains our system model and program notation. Chapter 3 gives the overall plan for TDMA slot assignment built on a CSMA model. As a building block, Chapter 4 describes how nodes learn of their neighborhoods and how this technique can be used to support the programming model. In Chapter 5 we present a randomized neighborhood naming algorithm that is the basis for fast distributed coloring and slot assignment. Then in Chapter 6 we use the unique naming to construct a maximal independent set of nodes, which are responsible for the coloring given in Chapter 7, and the slot computation presented in Chapter 8. Chapter 9 assembles all of the components into the final result and adds some concluding remarks.

Chapter 2

Wireless Network, Program Notation

The system is comprised of a set V of nodes in an *ad hoc* wireless network. 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 $i > 1$, let

$$N_p^i \;=\; N_p^{i-1} \;\cup\; \{\, r \mid (\exists q :\; q \in N_p^{i-1} :\; r \in N_q) \,\}$$

We call N_p^i the distance-i neighborhood of p. Distribution of these nodes is such that the network is connected, meaning there exists at least one path of intersecting neighborhoods between any two nodes; the distribution is also sparse enough with respect to radio range to bound the neighborhood size of any node: there is some known constant δ such that for any node p, $|N_p| \leq \delta$. We use δ in the design and analysis of algorithms for this network.

Each node has fine-grained, real-time clock hardware. We assume that 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). Communication is half-duplex: node p cannot send one message and receive another message concurrently. In fact, while p is transmitting, p is unable to detect whether or not another node is also transmitting. Therefore, collisions are possible in this model. Nodes do not have collision detection hardware. If $q \in N_p$ and $r \in N_p$ concurrently transmit, then p receives the superposition of their transmissions, but p cannot detect that the superposition is a result of collision, because noise can corrupt messages. We assume that each message contains sufficient error detection codes so that the event of corruption or collision can be deduced – a node cannot distinguish between corruption and collision.

These facts preclude the use of CSMA/CD access control to the radio medium, however some basic techniques of CSMA are applicable: 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. A statistical technique dealing with collisions in this model is CSMA/CA: before p transmits

a message, it waits for some random period. We assume that nodes have such CSMA/CA capability (as implemented, for instance, in [10]). 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 corresponds to typical assumptions for multiaccess channels [11]; the independence of τ for different frame transmissions indicates our assumption of an underlying memoryless probability distribution in a Markov model).

To simplify the presentation, we suppose that nodes have unique identifiers. In Chapter 5, we present a randomized algorithm to give each node p an identifier that is unique within N_p^2; we conjecture that our assumption of globally unique identifiers is not necessary.

Notation. We describe algorithms using the notation of guarded assignment statements: $G \to S$ represents a guarded assignment, where G is a predicate of the local variables of a node, and S is an assignment to local variables of the node. If predicate G (called the *guard*) holds, then assignment S is executed, otherwise S is skipped. At any system state where a given guard G holds, we say that G is *enabled* at that state.

The [] operator is the commutative and associative nondeterministic composition of guarded assignments:

$$G_0 \to S_0 \; [] \; G_1 \to S_1 \; [] \; \cdots \; [] \; G_k \to S_k$$

is evaluated to be $G_i \to S_i$ for an arbitrary choice of $i \in [0, k]$ such that G_i is *true*, and if no such choice of i is possible, then the result is equivalent to a "skip" statement. The notation $([]q : \; q \in L_p : \; G_q \to S_q)$ is a closed-form expression of

$$G_{q1} \to S_{q1} \; [] \; G_{q2} \to S_{q2} \; [] \; \cdots \; [] \; G_{qk} \to S_{qk}$$

where $L_p = \{q1, q2, \ldots, qk\}$. Following the convention of many authors in describing self-stabilizing algorithms, the operator "[]" is implicit for any list of labeled guarded assignments. For example, a program of the form

$$\mathsf{R0}: \quad G_0 \to S_0 \qquad \cdots \qquad \mathsf{R}k: \quad G_k \to S_k$$

represents $([] \, i : \; 0 \leq i \leq k : \; G_i \to S_i)$.

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*. 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. Generally, we suppose that when a node executes its program, all statements with *true* guards are executed in some constant time (done, for example, in round-robin order).

One 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 as follows. A certain subset of the variables at any node are designated as *shared* variables. If a statement $G \to S$ assigns to a shared variable, then we present the statement without any reference to the clock and we suppose that there is a transformation of the statement into a computation that slows execution so that it does not exceed some desired rate, and also

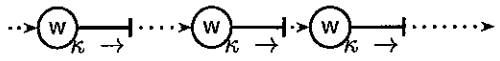


Figure 2.1: shared variable propagation

provides randomization to avoid collision in messages that contain shared variable values. This could be implemented using a timer associated with $G \rightarrow S$. One technique for implementing $G \rightarrow S$ could be by the following procedure:

Suppose the previous invocation of the procedure for $G \rightarrow S$ 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 the assignment if G is *false*), the node transmits a message containing all shared variable values. This message transmission may be postponed if the node is currently receiving a message. 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 2.1 illustrates the cycle of shared variable propagation for one node.

To reconcile our earlier assumption of immediate, atomic processing of messages with the discipline of shared variable assignment, no guarded assignment execution should change a shared variable in the atomic processing of receiving a message. All the programs in this paper have this property, that receipt of a message atomically changes only nonshared variables.

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. Further details about the propagation of shared variables and caches are given in Chapter 4.

Bibliographic Note. A typical model for self-stabilizing constructions uses shared variables (but *not* shared memory). That is, each process in the system can atomically read variables that other processes write (but no variable can be written by more than one process). Some papers assume high-atomicity programming units, where a process step can read all variables of neighboring processes atomically; other papers have weaker models, where a process can only read or write one shared variable atomically per step. Many papers investigate the possibility of implementing one model in terms of another under various topology, synchrony, and symmetry assumptions (such as the availability of unique identifiers, a distinguished "root" process, and so on). A few papers present self-stabilization for message-passing protocols. The paper [12] presents a transaction-based technique for implementing atomic evaluation of guarded assignments based on a message model. To the best of our knowledge, previous studies use assumptions of asynchronous message-passing or guaranteed time-bounds on message delivery over communication channels: previous research on self-stabilization does not investigate local area network assumptions, where message collision is a consideration.

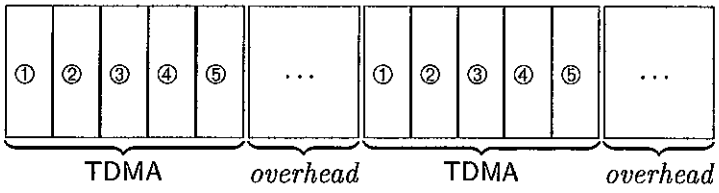
Chapter 3

Model Construction

Our goal is to provide an implementation of a general purpose, collision-free communication service. This service can be regarded as a transformation of the given model of Chapter 2 into a model without collisions. This service simplifies application programming and can reduce energy requirements for communication (messages do not need to be resent due to collisions). Let $\mathcal{T}$ denote the task of transforming the model of Chapter 2 into a collision-free model.

We seek a solution to $\mathcal{T}$ that is 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; eventually the transformer corrects node states to result in collision-free communication. 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 may introduce overhead messages susceptible to collisions. Initially, in the development of algorithms, we accept collisions and resends of these messages, which are internal to $\mathcal{T}$ and not visible to the application.

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 [5]. 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 messages of the algorithm that 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 is, in fact, a fixed-length slot in the TDMA schedule.



The programming model, including the technique for sharing variables described in Chapter 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 2.1 may span overhead slots: the computation by the solution to $\mathcal{T}$ operates in the concatenation of all the overhead slots. Whereas CSMA 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.

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 and N^3-neighborhood services that update cached copies of shared variables.

Chapter 4

Neighborhood Identification

We do not assume, in Chapter 3, 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 algorithm 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 and N_p^2 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 be the form of messages transmitted for sL_p. Let L_p be a private variable of a type that is an augmentation of L – it associates a real number with each item: $age(a : A)$ is a positive real value attached to the item.

Function $\mathsf{update}(L_p, a : A)$ 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 $\mathsf{neighbors}(L_p)$ 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* $mN(a : A)$ $\rightarrow$ $\mathsf{update}(L_p, a : A \setminus \{p\})$

N1: $([]\,(a : A) \in L_p :\ age(a : A) > \mathsf{maxAge} \rightarrow$
$L_p := L_p \setminus (a : A)\ )$

N2: $true \rightarrow\ sL_p := (p : \mathsf{neighbors}(L_p))$

We cannot directly prove that this algorithm stabilizes because the CSMA 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 safely remove old or invalid neighbor data, yet to retain current neighbor information by receiving new mN messages 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: 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 (it is likely that N_p^3 is not necessary; we discuss this issue in Chapters 5 and 7). In all subsequent chapters, we use N_p, N_p^2, and N_p^3 as constants in programs with the understanding that such neighborhood identification is actually obtained by the stabilizing protocol described above.

Building upon L_p, cached values of the shared variables of nodes in N_p, N_p^2, and N_p^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 $\boxtimes var_q$ 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 do not change value. With the exception of one algorithm presented in Chapter 5, 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 chapters, 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. Chapter 2 states our assumption on wireless transmission, based on the constant τ for collision-free transmission. The discipline of shared variable propagation illustrated in Figure 2.1 spaces

shared-variable updates (or skipping updates when there is nothing to change) by $\kappa+\beta$, where β is a random variable. Our requirement on the behavior of transmitting shared variable values thus also implies that for any time t between transmission events, there is a constant probability that the next transmission event will occur by $t + \alpha$ for some constant α. Notice that the joint probability of waiting at most time α, and then sending without collision, is bounded below by a constant. It follows that the expected number of attempts to propagate a shared variable value before successfully writing to all its caches is $O(1)$. (In fact, it would not change our analysis if random variable β is truncated by aborting attempted transmissions that exceed some constant timeout threshold.) We henceforth assume that the expected time for shared variable propagation is constant.

Chapter 5

Neighborhood Unique Naming

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.

The problem of neighborhood unique naming can be considered as an N^3-coloring algorithm and quickly suggests a solution to $\mathcal{T}$. Since neighborhood unique naming provides a solution to the problem of $L(1,1)$ coloring, it provides a schedule for TDMA. This solution would be especially wasteful if the space of unique identifiers is larger than $|V|$. It turns out that having unique identifiers within a neighborhood can be exploited by other algorithms to obtain a minimal N^2-coloring, so we present a simple randomized algorithm for N^3-naming.

Our neighborhood unique naming algorithm is roughly based on the randomized technique described in [6], 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 chapter. We call Δ the *namespace*. Let shared variable Id_p have domain $0..\Delta$; variable Id_p is the *name* of node p. Another variable is used to collect the names of neighboring nodes: $Cids_p = \{ \boxtimes Id_q \mid q \in N_p^3 \setminus \{p\} \}$. 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 $q \in N_p^2$, no name in $Cids_q$ equals Id_p,

(iv) for each $q \in N_p^3$, $q \neq p$, the equality $\bowtie Id_p = Id_p$ 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 [6]. 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 1 *Starting from any state, 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 $\bowtie Id_p \neq Id_p$ 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 in N_p^3 and their caches (or 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 $(\delta^2 + 1) \cdot \mu$, 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, 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 self-stabilizes with probability 1 and has constant expected convergence time.*

Proof: The Markov chain for the algorithm has a trapping state for any p such that $\mathsf{Uniq}(p)$ holds. The stability of $\mathsf{Uniq}(p)$ 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 $\mathsf{Uniq}(p)$ 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 2 *There is a constant d such that every path P satisfying $Up(P)$ has fewer than d nodes.*

Proof: Let $d = |\Delta| + 1$. If a path P satisfying $Up(P)$ has d 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 $a < a$, which 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, $|N_p^3| = \delta^3 + 1$, and each node's cache can contain δ^3 names, so a choosing $t \approx 6$ could be satisfactory. Generally, larger values for t decrease the expected convergence time of the neighborood unique naming algorithm. On the other hand, smaller values of t will reduce the constant d, which will reduce the convergence time for algorithms described in subsequent chapters. The need for uniqueness up to distance three is an artifact of our presentation: uniqueness within distance two will likely suffice and reduce the constants considerably. We only use N_p^3 at one point in a proof in Chapter 7, and further remarks there indicate an alternative method could be used.

Chapter 6

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 that an assignment to satisfy distance two coloring can be done in constant time given a subset of leader nodes distributed in the network. 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.

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

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

Thus the algorithm should elect one leader (identified by the ℓ-variable) for each neighborhood. As in previous chapters, $\boxtimes \ell_p$ denotes the cached copy of the shared variable ℓ_p.

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 based on random delay. The algorithm's progress is therefore guaranteed with probability 1 rather than by deterministic means.

Lemma 3 *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 each node p stabilizes its value of ℓ_p within $O(\Delta)$ time. For the base case, a node p of name zero 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*. Furthermore, after $O(1)$ additional time, $\boxtimes \ell_p = \mathit{true}$ is a stable property at every node in N_p. A generalization of the base case is the set S of nodes with locally minimum names, that is, $(\forall p, q : \ p \in S \ \wedge \ q \in N_p : \ p < q)$. Each node $s \in S$ stabilizes to $\ell_s = \mathit{true}$ in $O(1)$ time. 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 $\boxtimes \ell_p$ 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. ❑

Chapter 7

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.

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

$$min\ell_p \;=\; \min\{\, q \mid q \in N_p \;\wedge\; \boxtimes \ell_q \,\}$$

In words, $min\ell_p$ is the smallest id of any neighbor that is a leader.

$spectrum_p$ is a set of pairs (c, r) where c is a color and r is an id. Pertaining only to nonleader nodes, $spectrum_p$ should contain $(color_p, min\ell_p)$ and $(\boxtimes\, color_q, \boxtimes\, min\ell_q)$ 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 for p is

$$\{\, c \mid (\exists\, q, r :\; q \in N_p \;\wedge\; (c, r) \in spectrum_q \;\wedge\; r < p) \,\}$$

That is, used colors with respect to p are those colors in N_p^2 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. 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: $\ell_p \rightarrow setcol_p[dom_p] := f(\{ c \mid \exists q : q \in N_p \wedge r < p \wedge (c,r) \in \boxtimes spectrum_q \})$

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 4 *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 $O(d)$ phases, where d is the constant of Lemma 2. 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 $spectrum_q$ a set of tuples with the property that for any $(c,r) \in spectrum_q$, $r \geq p$. Notice that although $spectrum_q$ could subsequently change in the course of the execution, this property is stable. Therefore, in $O(1)$ additional time, no tuple of $\boxtimes spectrum_q$ 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 $q \in N_p$, $\boxtimes setcol_p$ 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 2). 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 d, thanks to Lemma 2, hence the overall bound of $O(d)$ 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 2'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.

Due to space restrictions, we omit the proof that the resulting coloring is minimal (which follows from the construction of f to be locally minimum, and the essentially sequential assignment of colors along paths of increasing names).

Chapter 8

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, *ie*, 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 1.1, 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 at least as much bandwidth as any other node in N_p^2. This follows from our assumption that all nodes have equal priority. Consider the N_p^2 sizes shown in Figure 1.2 that correspond to the colorings of 1.1. The rightmost node p in the first 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 time slots 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 $0 \leq x < y \leq 1$. To compute slots, 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 slots, 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: $true \rightarrow base_p := |\{ \bowtie color_q \mid q \in N_p^2 \}|$

R10: $true \rightarrow S_p := \bigcup \{ \bowtie itvl_q \mid q \in N_p^2 \ \wedge$
$(\bowtie base_q > base_p \vee$
$(\bowtie base_q = base_p \wedge \bowtie color_q < color_p)) \}$

R11: $true \rightarrow itvl_p := g(base_p^{-1}, S_p)$

Lemma 5 *With probability 1 the algorithm R9–R11 converges to an allocation of time slots such that no two nodes within distance two have conflicting time slots, and the interval lengths for each node p sum to* $|\{ color_q \mid q \in N_p^2 \}|^{-1}$*; the expected convergence time of R9-R11 is* $O(1)$ *starting from any state with stable and valid coloring.*

Proof: The proof follows the same structure as that given for Lemma 4 and we omit details. ❑

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 leftover intervals). A remaining practical issue is the conversion from intervals to a time slot schedule: a TDMA slot schedule will approximate the intervals calculated by g.

Chapter 9

Conclusion

Given the component algorithms of Chapters 4–8, 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 convergence time.*

Proof: The infrastructure for neighborhood discovery and shared variable propagation N0–N2 contributes $O(1)$ delay (partly by assumption on the CSMA behavior), and N3 establishes neighborhood unique naming in expected $O(1)$ time. The subsequent layers R1–R3, R4–R8, and R9–R11, each have $O(1)$ expected convergence time, and each layer is only dependent on the output of the previous layer. The hierarchical composition theorem (see [17]) implies overall stabilization, and the expected convergence time is the sum of the expected convergence times of the components. ❑

Discussion.

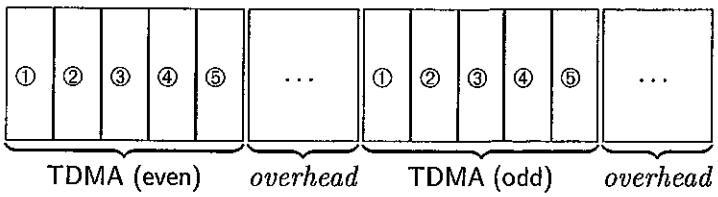
Our scheme provides a collision-free communication medium between node in a wireless sensor network. Our solution is self-stabilizing so that it can recover from any transient fault, but it is also scalable since its stabilization time is $O(1)$. Nodes dynamically leaving bring usually no problems since no new collision can occur, but nodes dynamically coming up in any initial state can be harmful – containing the effects of such events is an issue for further research.

Some component algorithms could find application to other problems in *ad hoc* networks. The neighborhood unique naming combined with leader selection by maximal independent set can be viewed as a clustering built in expected $O(1)$ time. The clusterheads are nodes with locally minimum names and the borders between clusters are nodes with locally maximum names.

Another 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 neighborood 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).

Two possible implementations of this transformer with our algorithm are as follows. For the distributed deamon (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.

For the locally central demon (at each step, a subset of non-neighboring nodes is scheduled for execution), we divide the TDMA schedule in $2 \times \Delta$ parts, and number TDMA parts from 0 mod $2 \times \Delta$. The parts of the TDMA that are even mod $2 \times \Delta$ are used to propagate shared variables between nodes. The other parts of the TDMA schedule are used to execute rules of the upper algorithm as if in the shared memory model. A node having color c only executes at TDMA part $2 \times c + 1$ mod $2 \times \Delta$. Since shared variables are accurate each time a node executes its upper algorithm, the semantics of the original algorithm are preserved. Since Δ 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.

Bibliography

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

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

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

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

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

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

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

[8] D. E. Culler, J. Hill, P. Buonadonna, R. Szewczyk, 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.

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

[10] A. Woo, 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.

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

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

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

[14] 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.

[15] 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.

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

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

RAPPORTS INTERNES AU LRI - ANNEE 2003

N°	Nom	Titre	Nbre de pages	Date parution
1345	FLANDRIN E LI H WEI B	A SUFFICIENT CONDITION FOR PANCYCLABILITY OF GRAPHS	16 PAGES	01/2003
1346	BARTH D BERTHOME P LAFOREST C VIAL S	SOME EULERIAN PARAMETERS ABOUT PERFORMANCES OF A CONVERGENCE ROUTING IN A 2D-MESH NETWORK	30 PAGES	01/2003
1347	FLANDRIN E LI H MARCZYK A WOZNIAK M	A CHVATAL-ERDOS TYPE CONDITION FOR PANCYCLABILITY	12 PAGES	01/2003
1348	AMAR D FLANDRIN E GANCARZEWICZ G WOJDA A P	BIPARTITE GRAPHS WITH EVERY MATCHING IN A CYCLE	26 PAGES	01/2003
1349	FRAIGNIAUD P GAURON P	THE CONTENT-ADDRESSABLE NETWORK D2B	26 PAGES	01/2003
1350	FAIK T SACLE J F	SOME b-CONTINUOUS CLASSES OF GRAPH	14 PAGES	01/2003
1351	FAVARON O HENNING M A	TOTAL DOMINATION IN CLAW-FREE GRAPHS WITH MINIMUM DEGREE TWO	14 PAGES	01/2003
1352	HU Z LI H	WEAK CYCLE PARTITION INVOLVING DEGREE SUM CONDITIONS	14 PAGES	02/2003
1353	JOHNEN C TIXEUIL S	ROUTE PRESERVING STABILIZATION	28 PAGES	03/2003
1354	PETITJEAN E	DESIGNING TIMED TEST CASES FROM REGION GRAPHS	14 PAGES	03/2003
1355	BERTHOME P DIALLO M FERREIRA A	GENERALIZED PARAMETRIC MULTI-TERMINAL FLOW PROBLEM	18 PAGES	03/2003
1356	FAVARON O HENNING M A	PAIRED DOMINATION IN CLAW-FREE CUBIC GRAPHS	16 PAGES	03/2003
1357	JOHNEN C PETIT F TIXEUIL S	AUTO-STABILISATION ET PROTOCOLES RESEAU	26 PAGES	03/2003
1358	FRANOVA M	LA "FOLIE" DE BRUNELLESCHI ET LA CONCEPTION DES SYSTEMES COMPLEXES	26 PAGES	04/2003
1359	HERAULT T LASSAIGNE R MAGNIETTE F PEYRONNET S	APPROXIMATE PROBABILISTIC MODEL CHECKING	18 PAGES	01/2003
1360	HU Z LI H	A NOTE ON ORE CONDITION AND CYCLE STRUCTURE	10 PAGES	04/2003
1361	DELAET S DUCOURTHIAL B TIXEUIL S	SELF-STABILIZATION WITH r-OPERATORS IN UNRELIABLE DIRECTED NETWORKS	24 PAGES	04/2003
1362	YAO J Y	RAPPORT SCIENTIFIQUE PRESENTE POUR L'OBTENTION D'UNE HABILITATION A DIRIGER DES RECHERCHES	72 PAGES	07/2003
1363	ROUSSEL N EVANS H HANSEN H	MIRRORSPACE : USING PROXIMITY AS AN INTERFACE TO VIDEO-MEDIATED COMMUNICATION	10 PAGES	07/2003

RAPPORTS INTERNES AU LRI - ANNEE 2003

N°	Nom	Titre	Nbre de pages	Date parution
1364	GOURAUD S D	GENERATION DE TESTS A L'AIDE D'OUTILS COMBINATOIRES : PREMIERS RESULTATS EXPERIMENTAUX	24 PAGES	07/2003
1365	BADIS H AL AGHA K	DISTRIBUTED ALGORITHMS FOR SINGLE AND MULTIPLE-METRIC LINK STATE QoS ROUTING	22 PAGES	07/2003