A Distributed and Self-Organizing Scheduling Algorithm for Energy-Efficient Data Aggregation in Wireless Sensor Networks

SUPRIYO CHATTERJEA, TIM NIEBERG, NIRVANA MERATNIA, and PAUL HAVINGA
University of Twente

20

Wireless sensor networks (WSNs) are increasingly being used to monitor various parameters in a wide range of environmental monitoring applications. In many instances, environmental scientists are interested in collecting raw data using long-running queries injected into a WSN for analyzing at a later stage, rather than injecting snap-shot queries containing data-reducing operators (e.g., MIN, MAX, AVG) that aggregate data. Collection of raw data poses a challenge to WSNs as very large amounts of data need to be transported through the network. This not only leads to high levels of energy consumption and thus diminished network lifetime but also results in poor data quality as much of the data may be lost due to the limited bandwidth of present-day sensor nodes. We alleviate this problem by allowing certain nodes in the network to aggregate data by taking advantage of spatial and temporal correlations of various physical parameters and thus eliminating the transmission of redundant data. In this article we present a distributed scheduling algorithm that decides when a particular node should perform this novel type of aggregation. The scheduling algorithm autonomously reassigns schedules when changes in network topology, due to failing or newly added nodes, are detected. Such changes in topology are detected using cross-layer information from the underlying MAC layer. We first present the theoretical performance bounds of our algorithm. We then present simulation results, which indicate a reduction in message transmissions of up to 85% and an increase in network lifetime of up to 92% when compared to collecting raw data. Our algorithm is also capable of completely eliminating dropped messages caused by buffer overflow.

Categories and Subject Descriptors: D.4.7 [**Operating Systems**]: Organization and Design—*Distributed systems*; *real-time systems and embedded systems*; D.4.1 [**Operating Systems**]: Process Management—*Scheduling*; D.4.4 [**Operating Systems**]: Communications Management—*Network communication*

General Terms: Algorithms, Performance, Measurement

Additional Key Words and Phrases: Wireless sensor network; scheduling; in-network data aggregation; self-organizing; cross-layer optimization; spatio-temporal correlation

This work was performed as part of the Dutch NWO funded CONSENSUS project and the EU funded e-Sense project.

Author's addresses: Pervasive Systems Group, Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, PO Box 217, 7500AE Enschede, The Netherlands.

ACM Reference Format:
Chatterjea, S., Nieberg, T., Meratnia, N., and Havinga, P. 2008. A distributed and self-organizing scheduling algorithm for energy-efficient data aggregation in wireless sensor networks. ACM Trans. Sens. Netw., 4, 4, Article 20 (August 2008), 41 pages. DOI = 10.1145/1387663.1387666 http://doi.acm.org/10.1145/1387663.1387666

1. INTRODUCTION

Wireless sensor networks (WSNs) are increasingly being used to carry out various forms of environmental monitoring. Monitoring vineyards [Burrell et al. 2004], wildlife habitats [Mainwaring et al. 2002], office buildings [Wen 2006], suspension bridges [Smyth et al. 2003], forests [Tolle et al. 2005], and even marine environments [Chatterjea et al. 2006] are just a few of the diverse range of sensor network applications that can be found in current literature. One of the primary motivations for using WSNs is that they allow environments to be monitored at extremely high spatial and temporal resolutions—something that is not possible using existing monitoring technologies. This is mainly due to the fact that sensor nodes are usually deployed in very high densities [Intanagonwiwat et al. 2002].

However, extracting the vast amounts of data generated by large-scale, high-density sensor network deployments can cause a wide range of problems. The fact that sensor nodes are typically battery powered devices makes energy resources a precious commodity. Transmitting every single acquired sensor reading would cause nodes to drain their batteries in a matter of days. WSN deployments however, will only be practically viable if they are able to run unattended for long durations. Furthermore, the limited bandwidth of present-day sensor nodes prevents all the acquired readings from being propagated successfully toward the sink. This results in dropped packets, which in turn has a negative impact on the quality of data collected.

As sensor readings of adjacent nodes in a high-density network may display a high degree of correlation, one way to reduce the amount of data that needs to be transmitted would be to exploit the spatial correlation between adjacent nodes. Thus, instead of having every node transmit its readings, we suggest a method that requires only a small subset of nodes in the network to transmit messages that represent all the remaining nodes at any point in time. We refer to nodes belonging to this subset as *correlating nodes*. Every correlating node initially transmits a message containing correlation information that indicates how the particular node's readings are correlated with its adjacent neighbors. Subsequently, the correlating node continues to transmit its own readings until a change in correlation is detected, in which case the updated correlation information is transmitted to the sink node. The sink node uses the correlation information and combines it with the subsequent reading received from a correlating node to deduce the readings of the adjacent neighbors of the correlating node. As it would be pointless to have two adjacent nodes act as correlating nodes simultaneously, in this article we present a completely distributed and self-organizing scheduling algorithm that decides when a particular node should act as a correlating node. Our contributions are stated as

follows:

(1) We present a completely distributed scheduling algorithm that enables every node to autonomously choose schedules based only on locally available information.
(2) We prove our algorithm possesses self-stabilizing properties that allow it to recover within a finite amount of time regardless of any disturbances in the network, such as topology changes or communication errors. We present theoretical upper bounds for message transmissions and network stabilization times when topology changes occur.
(3) We illustrate how our algorithm is able to adapt quickly to topology changes due to its close interaction with the underlying MAC layer. The algorithm also improves energy-efficiency by taking advantage of cross-layer information provided by the MAC.
(4) We present performance estimates and theoretical upper bounds for the performance of our algorithm. We evaluate the algorithm by presenting simulation results, which indicate a reduction in message transmissions of up to 85% and an increase in network lifetime of up to 92% when compared to collecting raw data. Our algorithm is also capable of completely eliminating dropped messages caused by buffer overflow.

An example application scenario and a list of assumptions we make are described in the following two sections. Section 4 provides the motivation and focus of this article. An overview of our approach is presented in Section 5. Sections 6 and 7 provide background information about the underlying MAC protocol and self-stabilization. The main scheduling algorithm is described in Section 8. We evaluate the performance of our approach in Section 9. Section 10 mentions the related work and finally the article is concluded in Section 11.

2. APPLICATION SCENARIO

We are currently working together with the Australian Institute of Marine Science (AIMS) [AIMS 2006] to set up a large-scale wireless sensor network to monitor various environmental parameters on the Great Barrier Reef (GBR) in Australia. Scientists at AIMS intend to use the collected data to study coral bleaching, reef-wide temperature fluctuations, and the impact of temperature on aquatic life and pollution.

One of the reefs under study is the Davies Reef, which is approximately 80km northeast of the city of Townsville in North Queensland, Australia. Currently, AIMS has a couple of data loggers situated on the reef that records temperature at two separate depths once every thirty minutes. Scientists from AIMS need to visit the reef periodically to download the data from the loggers.

The drawback of the current system is that it only allows single-point measurements. Thus it is impossible to get a true representation of the temperature gradients spanning the entire reef, which is approximately 7km in length. Also, the practice of collecting the data once every few weeks makes it impossible to study the trends of various parameters in real-time. Deploying a sensor network would not only allow high resolution monitoring in both the spatial and

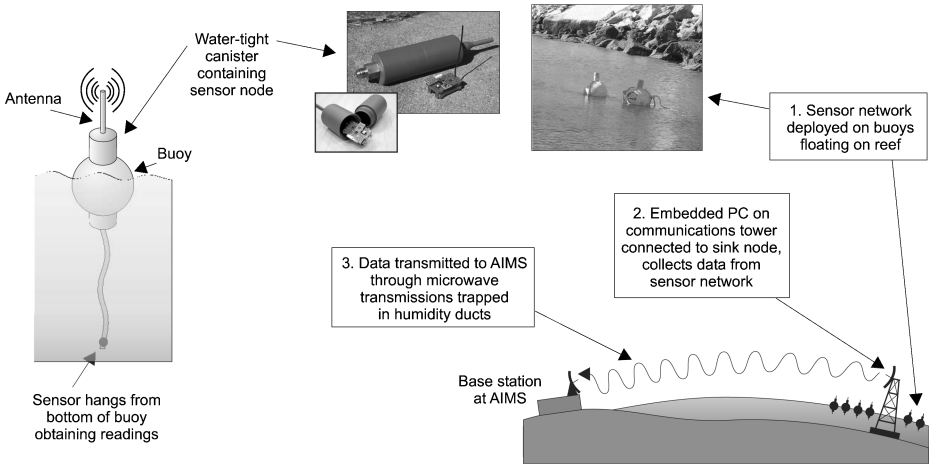

Fig. 1. Overview of data collection system at Davies Reef.

temporal dimensions, but would also enable scientists to improve their understanding of the complex environmental processes by studying data streaming in from the reef in real-time.

The new data collection system that we are deploying at Davies reef can be broken down into three main components as shown in Figure 1:

Ambient μNodes. These are the sensor nodes from Ambient Systems [Ambient 2006a] that will be placed in water and shock-proof canisters and then placed in buoys around the reef.

Embedded PC. An embedded PC will be placed on a communication tower and will act as the sink node, collecting data from all the sensors in the reef.

Microwave link. This will allow data to be transmitted from the Embedded PC to the AIMS base station 80 km away, using microwave transmissions trapped inside humidity ducts that form directly above the surface of the sea [Palazzi et al. 2005].

The work presented in this article focuses on the first component. We describe a distributed and self-organizing scheduling algorithm that runs on the Ambient μNodes and subsequently allows energy-efficient data gathering to be performed. We present a more in-depth explanation of the focus and motivation of this article in Section 4.

It is important to highlight however, that our work is not strictly tailored for the GBR. As mentioned later in Section 4, it can be used in a wide range of environmental monitoring scenarios where fine-grained spatio-temporal resolutions are required. We have simply chosen to use the GBR as a test bed to illustrate the feasibility of our solution.

3. ASSUMPTIONS

Based on our application scenario described above, we have made a few assumptions about the data that will be collected, and about the network itself. Firstly, as there will be a very large number of sensor nodes (∼100) and since they

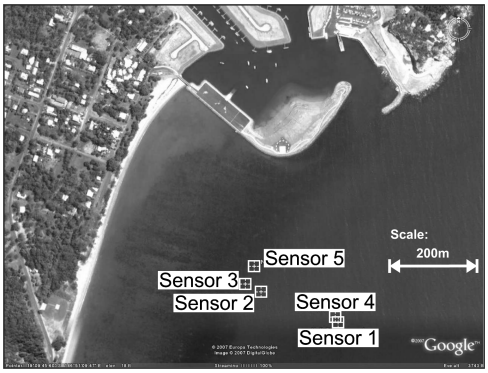

Fig. 2. Sensors deployed in Nelly Bay, Great Barrier Reef, Australia.

may be required to obtain readings at a high frequency, a large amount of data can be expected to flow through the network. Given the limited bandwidth and memory capacity of individual sensor nodes, assuming that nodes are transmitting data via a communication tree towards the sink node, nodes that are closer to the sink node will be prone to buffer overflows [Dulman et al. 2006]. This will result in loss of messages, which will greatly reduce the quality of data collected. Secondly, as there will be a very high density of sensor nodes, that is, they will be placed very close to each other, we can expect readings between neighboring nodes to be correlated during most parts of the day. This assumption can be verified by looking at data that has been collected from Nelly Bay in the GBR as shown in Figure 2 [Bondarenko et al. 2007]. Figure 3(a) presents a matrix that shows three characteristics of the five deployed sensors: temperature readings (d), correlation between the readings of any two sensors (c), and how correlation varies over time (b). It can be clearly seen that the correlation remains relatively constant over a 12 day duration. Note that temperature readings were obtained every 10 minutes.

As the sensor nodes will be placed on the reef for possibly a number of years, we assume that the topology of the network is relatively static. We do however take into consideration the fact that the network topology may change occasionally since the nodes are prone to failure (e.g., due to the harsh environment or dead batteries), and new nodes may be added to expand the network.

4. MOTIVATION AND FOCUS

Taking advantage of spatial correlations between neighboring nodes would enable nodes to filter out redundant data. This in turn would help reduce problems such as excessive energy usage, buffer overflows, and reduced data quality. Instead of transmitting every acquired sensor reading to the sink node, a node that discovers a correlation with its neighboring nodes, only transmits the correlation information, followed by its own readings. Thus, the sink node can then predict the readings of the neighboring nodes using the correlation information and the transmitted readings from the node performing the correlation. This is illustrated in Figure 4(b).

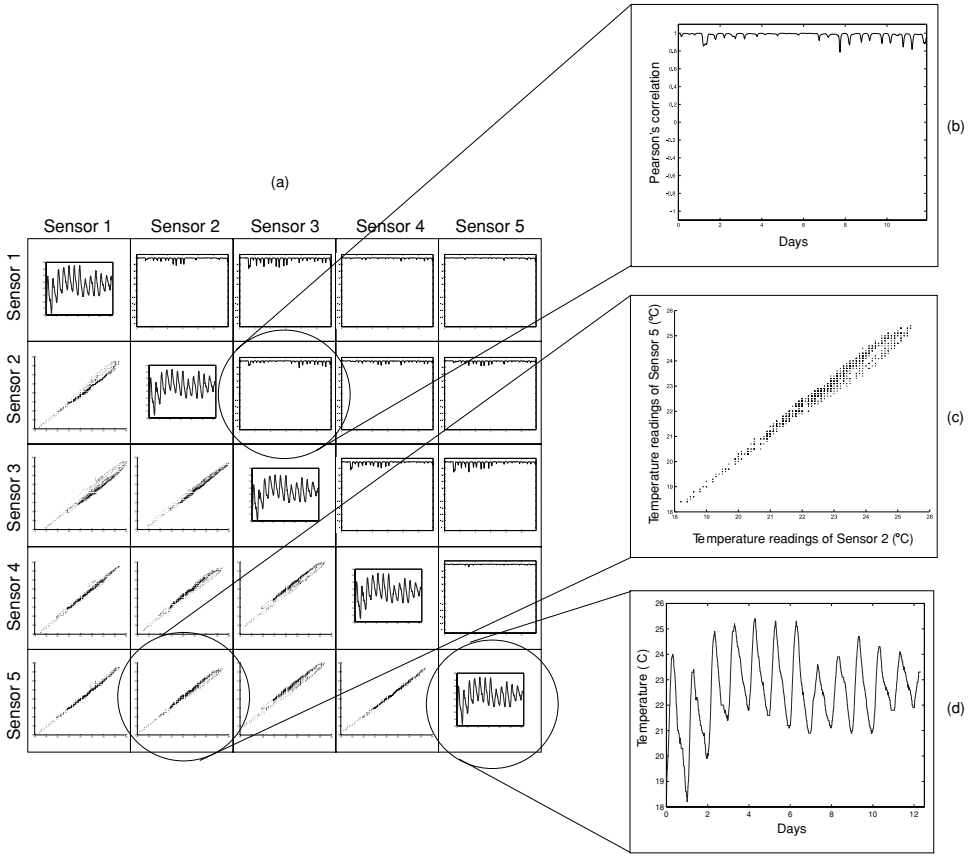

Fig. 3. (a) Correlation matrix, (b) Variation of correlation over time, (c) Correlation between two sensor readings, (d) Temperature readings.

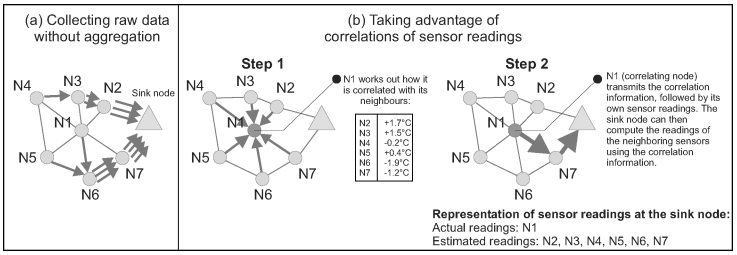

Fig. 4. Advantage of using correlation information (b) instead of transmitting raw data (a).

The approach of taking advantage of spatial and temporal correlations of sensor readings involves two issues that need to be addressed:

Identifying correlations and keeping correlation information updated. It is important to note that correlation is not a static attribute. Correlation between two neighboring sensors may exist at only certain times of the day. Thus a node

needs to be able to identify when a correlation may arise, and it also needs to ensure that the correlation information it has is up-to-date. Naturally, if trends of sensor readings change extremely rapidly, such a scheme would incur a very high overhead that would exceed the cost of collecting raw data from the network, due to frequent updates of the correlation information. However, preliminary readings obtained from our four different sensor network test beds situated in diverse environments ranging from the coral reef, to microclimates in trees, and even a typical office environment, have shown that sudden changes in trends of sensor readings are not particularly common. This characteristic is also clearly shown in Figure 3(b). In fact, during most parts of the day, sensors placed geographically close to one another, tend to display similar behavior. Our work is not designed for applications where correlations fluctuate rapidly.

Deciding when a node should act as a correlating node. It would not make sense for all nodes to send correlation information to the sink node simultaneously as this would involve sending more information than even transmitting raw sensor readings. Thus when one node is transmitting correlation data, the neighboring nodes should refrain from doing so. This implies that while nodes transmitting the correlation information (i.e., correlating nodes) are represented at the root node by their actual (own) readings, their neighbors are represented by estimated readings that are based on the correlation information transmitted by the correlating nodes (Figure 4(b)). Note that a correlating node initially transmits the correlation information followed by its own sensor readings. Thus, two neighboring nodes should not act as correlating nodes simultaneously at any instant of time. Furthermore, it is important to ensure that at all times, every node in the network is represented at the sink node either by an actual reading or by an estimated reading. This in turn means that if a node is not a correlating node at a given time, it must be connected to at least one neighboring correlating node.

Having a static scheduling scheme that fixes the correlating nodes for the entire lifetime of the network, is not desirable. This is because, though there would be a number of correlating nodes sending their own sensor readings in addition to the correlation information, a significant proportion of the nodes would always be represented at the root node by only estimated readings. Thus such a scheme would be prone to errors in the event that the correlating node fails for some reason and starts sending erroneous correlation information to the sink.

Thus in order to have a more robust scheme, every node in the network should be given an opportunity to be a correlating node. This would allow the sink to raise an alarm in case it notices that the actual readings of a node indicate a distinctly different characteristic compared to the estimated readings of the same node.

This clearly implies that there needs to be a scheduling scheme that decides when a certain node should be in charge of sending correlation information in the event that a correlation exists.

The work in this article focuses on the latter issue and presents a Distributed and self-Organizing Scheduling Algorithm ($\mathcal{DOSA}$), which that allows nodes to autonomously reassign the schedules if a change in topology

is detected, whether it is due to the failure or to the addition of nodes. We make the assumption in this article that correlations between neighboring sensor nodes do exist. The exact mechanisms for identifying correlations and keeping correlation models updated, does not fall within the scope of this article.

5. A MACRO PERSPECTIVE OF THE $\mathcal{DOSA}$ APPROACH

As we mentioned in Section 4, the primary objective of $\mathcal{DOSA}$ is to help decide when a particular node should act as a correlating node and thus be put in charge of representing the sensor readings of all the nodes in its one-hop neighborhood. Note that during the correlating node's schedule, the node initially transmits correlation information to the sink node followed by its own sensor readings. None of the nodes in the correlating node's one-hop neighborhood transmit their sensor readings to the sink during this period.

Since $\mathcal{DOSA}$ is intended to solve a scheduling problem, we make use of a distributed graph-coloring algorithm to assign schedules to individual nodes [Lynch 1996]. Thus, from a graph-theoretic point of view, since no two adjacent nodes can act as a correlating node simultaneously, all the nodes chosen by $\mathcal{DOSA}$ to be correlating nodes need to form an *independent set*. Additionally, the correlating nodes for a particular instant of time need to form a *dominating set* since every noncorrelating node must be joined to at least one correlating node by some edge. Also note that the subset of nodes that are both independent and dominating is known as a *maximal independent set*. A maximal independent set cannot be extended further by the addition of any other nodes from the graph.

It is these requirements that help us define the constraints, outlined later in Section 8, that $\mathcal{DOSA}$ follows in order to perform its intended task.

In order to hasten the speed at which the nodes are assigned schedules, $\mathcal{DOSA}$ makes use of the information provided by the underlying MAC protocol, LMAC [van Hoesel and Havinga 2004]. In other words, instead of $\mathcal{DOSA}$ having to color all the nodes from scratch, it takes advantage of the schedules (or colors) already assigned by LMAC and subsequently builds upon that to ensure that the requirements of $\mathcal{DOSA}$ are met. An added advantage of this form of cross-layer optimization is that fewer messages need to be transmitted for all the schedules to be assigned properly, since we make use of information that already exists. Furthermore, $\mathcal{DOSA}$'s dependence on LMAC makes it more reactive to changes in topology since any changes in neighborhood detected by LMAC are immediately filtered to $\mathcal{DOSA}$.

Because the operation of $\mathcal{DOSA}$ is completely dependent on LMAC, we first give a brief overview of LMAC and then proceed to present the operation of $\mathcal{DOSA}$.

6. LMAC: A LIGHTWEIGHT MEDIUM ACCESS CONTROL PROTOCOL

LMAC is a TDMA-based lightweight medium access control protocol designed specifically for wireless sensor networks. Instead of contending for the medium, like carrier-sensing based MAC protocols [Ye et al. 2002; Dam and Langendoen

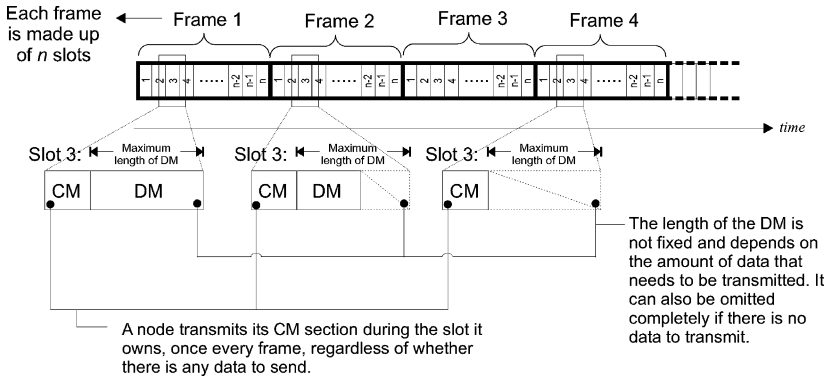

Fig. 5. Illustration of frames and slots in LMAC.

2003], time in LMAC is divided into frames, each of which is further divided into a fixed number of time slots (Figure 5). Every node chooses its own slot using a distributed algorithm that uses only locally available information. A node is allowed to pick any slot as long as it is not owned by any other node within its two-hop neighborhood. This mechanism effectively helps avoid the hidden-terminal problem because it makes it impossible for two nodes that are two hops away from each other to transmit at the same time. It also prevents all slots from being used up, since LMAC ensures that two nodes that are at least three hops away from each other can reuse the same time slot.

A time slot consists of two sections, the Control Message (CM), and the Data Message (DM). The CM, which contains control information and has a fixed length, is broadcast by a node to its neighbors during its own time slot, once every frame, irrespective of whether the node has any data to send. The CM contains a table that identifies the slots that are occupied by itself and by its one-hop neighbors as well as other control information. Every node maintains a *Neighbor Table* that stores the information about its one-hop neighbors, such as, ID, occupied slot, number of hops to sink node, and so forth. Occupied slots are marked with a 1, whereas unoccupied ones are marked with a 0. A node joining the network, first listens out for the CMs of all its neighbors and then picks one of the slots that is marked as unoccupied, by performing an OR-operation. This mechanism is illustrated in Figure 6.

The DM contains higher layer protocol messages. The length of the DM can vary depending on the amount of data that a node needs to send. It does however, have a maximum length as shown in Figure 5.

7. PRELIMINARIES FOR SELF-STABILIZATION

Since we later illustrate how $\mathcal{DOSA}$ initializes during start-up and how it is capable of recovering from topology changes caused by the addition or removal of nodes, we follow the self-stabilization [Dijkstra 1974; Dolev 2000] approach to formalizing the self-organizing properties of the algorithm. Self-stabilization allows a system that enters an illegitimate state (e.g., due to the occurrence of transient faults) to converge back to a legitimate state within a finite

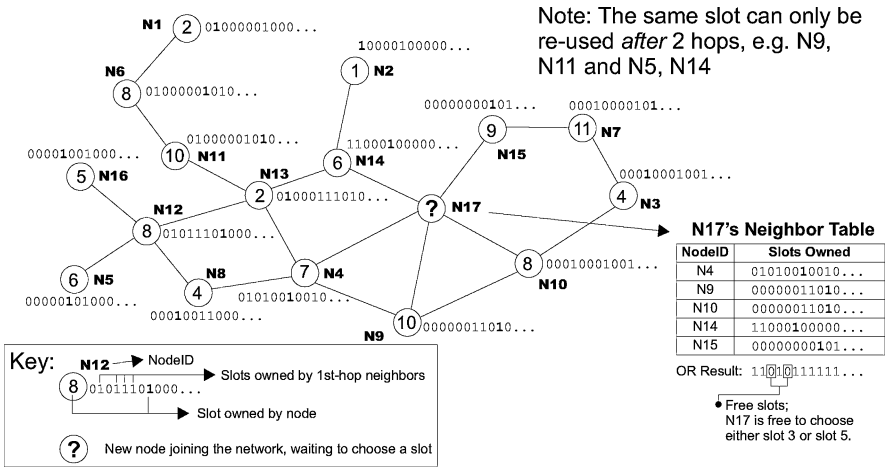

Fig. 6. Distributed slot allocation in LMAC.

time without any external intervention. We now present some preliminaries of self-stabilization.

All nodes in the network are assumed to have unique IDs and to have knowledge of their adjacent neighbors. Each node has a state that is specified by its local variables. The state of the entire system is called the *global state* or *configuration* and is the union of the local states of all the nodes. The objective of the system is to reach a desirable global final-state called a *legitimate state*. The state of a system can either be *legitimate* or *illegitimate*. We use $\mathcal{S}$ to denote the set of all possible states. In order for the system to recover after a transient fault, all the affected nodes repeatedly execute a piece of code consisting of a finite set of rules having the form $(label)[guard] : <statement>;$. The statement part of the rule is the description of the algorithm used to compute the new values for local variables. A rule is *enabled* when its guard is true. The *execution* of an enabled rule determines the new state value of a node using the algorithm described by the statement part of the rule.

We denote the set of all legitimate states by $\mathcal{L}$ such that $\mathcal{L} \subseteq \mathcal{S}$. We denote the set of rules using $\mathcal{R}$ where $\mathcal{R} \in \mathcal{S} \times \mathcal{S}$ such that $(s_i, s_j) \in \mathcal{R}$. An execution of e is a maximal sequence of states, $e = s_i, s_{i+1}, ...s_j$ such that $\forall i \geq 1, s_i \in \mathcal{S}$, and s_i is reached from s_{i-1} by executing a particular rule.

A system can be considered to be self-stabilizing if the following two conditions hold:

—Closure: If $s \in \mathcal{L}$ and $s \rightarrow s'$ then $s' \in \mathcal{L}$. Therefore the closure property means that when a system is in a legitimate state, the following state is always a legitimate state as well, regardless of the rule executed.

—Convergence: Starting from any configuration $s \in \mathcal{S}$, every execution reaches $\mathcal{L}$ within a finite number of transitions.

The preliminaries presented thus for are used in the following sections to illustrate how $\mathcal{DOSA}$ is able to start-up properly and also how it is capable of recovering when the system experiences certain transient faults.

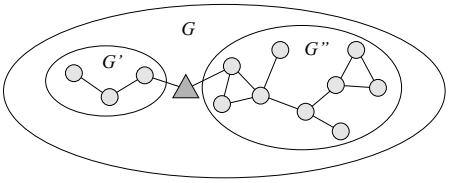

Fig. 7. Two independent components in G.

8. $\mathcal{DOSA}$: A DISTRIBUTED AND SELF-ORGANIZING SCHEDULING ALGORITHM

$\mathcal{DOSA}$ uses a distributed graph coloring approach to decide when a particular node should be a correlating node. Every color owned by a node represents a particular frame of time during which a node is required to act as a correlating node. In conventional graph coloring approaches, colors are assigned to vertices such that adjacent vertices are assigned different colors and the number of colors used is minimized. While $\mathcal{DOSA}$'s graph coloring approach also ensures that adjacent nodes in the network do not own the same colors, it differs in the sense that each node is allowed to own *multiple* colors, that is, a node can have multiple schedules. Moreover, the number of colors used in $\mathcal{DOSA}$ is fixed and is equal to the number of slots that are assigned to an LMAC frame.

Before we proceed, we first state certain definitions that are used throughout the rest of this article.

We model the network topology as an undirected graph G, where $G = (V, E)$. V represents the vertices or nodes in the network while two nodes are connected by an edge in E if they are within radio transmission range of each other. $\mathbf{K}$ represents the set of colors used to color all the nodes. So $|\mathbf{K}|$ is equal to the number of slots per frame in LMAC. Also, we denote the *closed* neighborhood of a node $v \in V$ by $\Gamma(v)$:

$$\Gamma(v) := \{u \in V | (u, v) \in E\} \cup \{v\}.$$

In other words, the closed neighborhood of v includes not only its adjacent neighbors but also the node v itself. Using the graph-theoretic distance $d_G(u, v)$, that denotes the number of edges on a shortest path in G between vertices u and v, we can define the r^{th} neighborhood of v as $\Gamma_r(v) := \{u \in V | d_G(u, v) \leq r\}$. Similarly, we define the *open* neighborhood of a node v by $\Gamma'(v) := \{u \in V | (u, v) \in E\}$.

We refer to C_v as the set of colors owned by node v. For C_v it can easily be seen that

$$0 < |C_v| < (|\mathbf{K}| - |\Gamma'(v)|)$$

has to hold.

Given that a *node-induced subgraph* is a subset of the nodes of a graph G together with edges whose endpoints are both in this subset, we define a *component* as a node-induced subgraph of a subset of nodes. Furthermore, we call two components independent if they are not connected by an edge. As an example, in Figure 7, G' and G'' are two independent components in G.

Before describing the details of the operation of $\mathcal{DOSA}$, we first state the constraints derived from the requirements stated in Section 5, which define its behavior. The following two constraints must be met when two nodes u and v are adjacent to each other:

Constraint 1: $C_v \cap C_u = \emptyset$

In other words, two adjacent nodes cannot own the same colors. This is because two adjacent nodes should not be assigned as correlating nodes in the same time instant.

Constraint 2: $C_{\Gamma(v)} = \mathrm{K}$

All colors should be present within the one-hop neighborhood of node v, that is, if node v does not own a particular color itself, the color must be present in one of its neighboring nodes that is one hop away. This ensures that every node's readings will be represented at the sink node for every time instant either, directly or through a correlated reading.

LEMMA 8.1. *The combination of constraints 1 and 2 ensures that at any time slot, c_i, all nodes owning the color c_i, which correspond to that time slot, form a maximal independent set on G.*

PROOF. At any time instant according to Constraint 1, two adjacent nodes will never own the color c_i, thus resulting in an independent set I. Constraint 2 ensures that in the closed neighborhood of every node $v \in V$, every color is present. This clearly results in a maximal independent set. □

8.1 Details of Simulation Setup

For the sake of easier comparison, we present the simulation results immediately after the description of the theoretical performance bounds of $\mathcal{DOSA}$ in every subsection that follows. Thus we first state the salient details of our simulation setup and then proceed with the rest of the sections.

All simulations are implemented in Matlab [2006]. Simulation results (unless otherwise specified) are averaged out over 100 randomly generated network topologies for a particular average node connectivity. Each topology consists of 100 nodes randomly distributed in a 100 x 100 unit area. The average connectivity (or neighbor density) has been varied from 5 to 11 by setting different transmission ranges for the nodes. Nodes are static and homogeneous in the sense that all the nodes have the same transmission radii. The number of slots per frame in the LMAC implementation is 32.

8.2 Dependency of $\mathcal{DOSA}$ on LMAC

As mentioned in Section 6, LMAC assigns a slot to every node in the network. $\mathcal{DOSA}$ begins its distributed coloring scheme by considering the initial slot assignment phase in LMAC as an input. Slot assignments in LMAC correspond to partial color assignments in $\mathcal{DOSA}$. Thus, while LMAC assigns every to node with a single color, $\mathcal{DOSA}$ assigns the remaining colors that ensure the adherence to the constraints 1 and 2, given in the previous section. We can then state that $C_v = C_{vLMAC} \cup C_{vDOSA}$, where C_{vLMAC} refers to the color corresponding

to the LMAC slot owned by node v, and $C_{v_{DOSA}}$ refers to the colors assigned to node v, by $\mathcal{DOSA}$.

Similarly, the colors owned by the nodes adjacent to node v, $C_{\Gamma'(v)}$, are also made up of LMAC and $\mathcal{DOSA}$ colors. Thus we can state, $C_{\Gamma'(v)} = C_{\Gamma'(v)_{LMAC}} \cup C_{\Gamma'(v)_{\mathcal{DOSA}}}$.

The dependency of $\mathcal{DOSA}$ on LMAC allows nodes to adapt autonomously and immediately to changes in network topology. For example, the addition or removal of a node results in the change being reflected in the LMAC neighbor tables of all other neighboring nodes within range. $\mathcal{DOSA}$ detects changes in LMAC's neighbor table and performs a reassignment of schedules if any of the neighboring nodes do not meet the constraints mentioned above. Utilizing such cross-layer information from LMAC ensures that $\mathcal{DOSA}$ does not spend additional resources trying to detect topology changes itself.

We also make the assumption that the maximum degree of a single node in the network is always known prior to deployment. This information is used to choose the appropriate number of slots in a particular frame in LMAC. In case the maximal degree of the nodes cannot be bounded accurately enough, LMAC also offers functionality to operate nodes passively, that is, without owning a time-slot, when the network gets (locally) dense (see cf. Nieberg [2006]). However, for ease of notation and argumentation, we only consider active nodes that are assumed to acquire a free slot when carrying out slot assignment. The proper operation of LMAC also guarantees the proper operation of $\mathcal{DOSA}$.

8.3 General Operation of $\mathcal{DOSA}$

$\mathcal{DOSA}$ uses a *greedy* approach to assign colors to nodes. Coloring is performed using two types of colors: *LMAC colors* and $\mathcal{DOSA}$ *colors*. LMAC colors refer to the colors that have been assigned by LMAC, due to the slot assignment. $\mathcal{DOSA}$ colors refer to the additional colors that are assigned by $\mathcal{DOSA}$ to ensure that constraints 1 and 2 are met. This occurs after the LMAC colors have been assigned. $\mathcal{DOSA}$ does not have any control over the LMAC color of a node since that depends purely on the slot assignment performed by LMAC. In fact, such control is not required. Therefore, in the following, we refer to $\mathcal{DOSA}$ colors simply as colors, unless otherwise indicated.

Colors are acquired based on a calculated priority. A node computes its priority within its one-hop neighborhood, based on its degree and on its node ID. The higher the degree of a node, the higher its priority. If two neighboring nodes have the same degree, priority is calculated based on the unique node ID; the node with the larger node ID will have the higher priority. This priority computation is performed in Line 4 of Algorithm 1.

Once all nodes have acquired their LMAC slots, a *BeginSecondPhase* message is injected into the network through the sink node, requesting the nodes to begin the $\mathcal{DOSA}$ coloring phase. At this stage, any node receiving the *BeginSecondPhase* message only has an LMAC color and so does not satisfy the constraints mentioned earlier. Thus, these nodes mark themselves as `Unsatisfied`. A node only attains the `Satisfied` status when it satisfies the

two constraints mentioned in Section 8. Upon receiving the *BeginSecondPhase* message, a node broadcasts its *NodeStatus* message. This message contains information about the node's degree, its status (i.e., Satisfied/Unsatisfied), and the list of colors owned. The *ColorsOwned* field is a string of $|\mathbf{K}|$ bits where every color owned by a node is marked with a 1. The rest of the bits are marked with a 0. Initially, a node only marks its own LMAC color as 1 due to the initial LMAC slot assignment. A neighboring node that receives the *NodeStatus* message then performs coloring using $\mathcal{DOSA}$ as outlined in Algorithm 1 . Note that the *NodeStatus* message is the only message that is used for the operation of $\mathcal{DOSA}$.

Algorithm 1. $\mathcal{DOSA}$—Normal Initialization

Input: NodeStatusMSG(Degree, SatisfiedStatus(TRUE/FALSE), ColoursOwned)
Output: NodeStatusMSG(Degree, SatisfiedStatus(TRUE), ColoursOwned)/NIL

```
1:  UPDATE(LocalInfoTable, v)
2:  if LocalInfoTable contains entries from ALL adjacent nodes then
3:    if SatisfiedStatus(v) = FALSE then
4:      Compute PRIORITY(v)
5:      if PRIORITY(v) = Highest then
6:        C_v ← K\C_Γ'(v)
7:        ColorsOwned ← C_v
8:        SatisfiedStatus ← TRUE
9:        UPDATE(LocalInfoTable, v)
10:       BROADCAST NodeStatusMSG(Degree, SatisfiedStatus, ColoursOwned)
11:     end if
12:   end if
13: end if
```

We now briefly describe the operation of $\mathcal{DOSA}$ as outlined in Algorithm 1. Upon receiving a *NodeStatus* message, a node first updates its *LocalInfoTable* (Line 1). This table stores all the information contained in the *NodeStatus* messages that are received from all the adjacent nodes. Once a node receives *NodeStatus* messages from all of its immediate neighbors (Line 2), if its status is `Unsatisfied` (Line 3), the node proceeds to compute its priority. PRIORITY computes the priority of a node only among its unsatisfied neighbors (Line 4), that is, as time progresses and more nodes attain the `Satisfied` status, PRIORITY needs to consider a smaller number of neighboring nodes. The highest priority is given to the node with the largest degree among its adjacent `Unsatisfied` neighbors. If more than one node has the same degree, then the highest priority is given to the `Unsatisfied` node with the largest NodeID.

The node that has the highest priority among all its immediate unsatisfied neighbors, acquires all the colors that are not owned by any of its adjacent neighbors (Line 7). Since as the node has then satisfied both constraints of $\mathcal{DOSA}$, it switches to the `Satisfied` state, updates its own *LocalInfoTable*, and informs all its neighbors through a broadcast operation (Lines 8–10). Note that this technique corresponds to a highest-degree greedy approach.

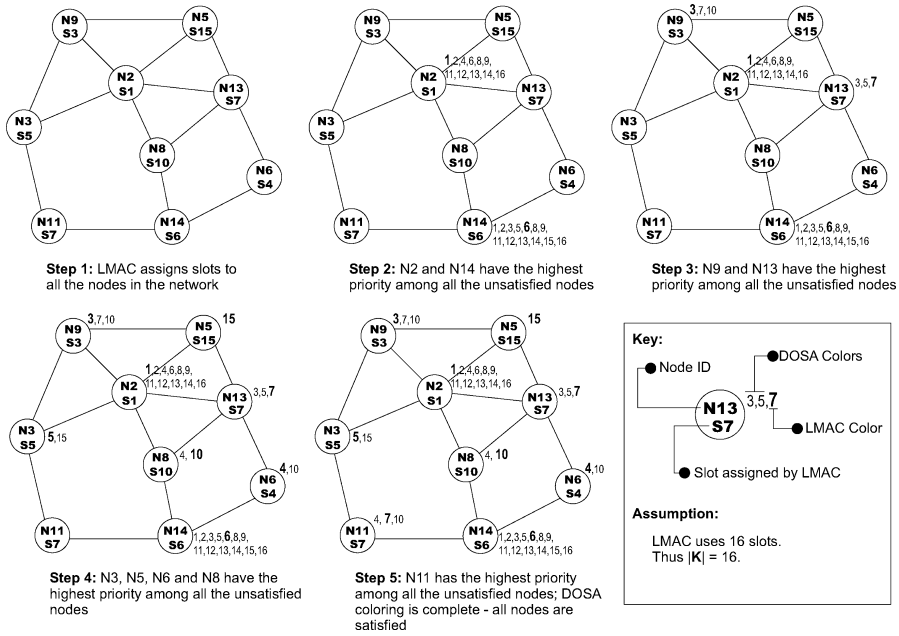

Fig. 8. A step-by-step example of how $\mathcal{DOSA}$ colors are assigned.

Figure 8 provides a step-by-step example of how the $\mathcal{DOSA}$ algorithm assigns colors to the nodes in a network. We make the assumption in the example that LMAC uses 16 slots.

8.3.1 *Correctness of $\mathcal{DOSA}$.* In this section we illustrate how $\mathcal{DOSA}$ is able to successfully carry out initialization within a finite time given any arbitrarily chosen network. We initially assume that no transmission errors occur throughout the initialization phase, but we subsequently describe how such issues are handled in Section 8.3.2.

In order for $\mathcal{DOSA}$ to operate properly, it is absolutely imperative that every node always has up-to-date state information about its immediate neighbors. If a node n experiences a certain change in state (e.g., a change from `Satisfied` to `Unsatisfied`) and fails to inform an adjacent neighbor of the change, this neighbor node might execute certain inappropriate steps based on its outdated state information for n. This error may prevent $\mathcal{DOSA}$ from stabilizing within a finite time. Thus it is essential for $\mathcal{DOSA}$ to possess the *cache coherence* property [Herman 2003].

Let each node $v \in V$ in the sensor network have a variable, C_v indicating the colors owned by node v. For each $(u, v) \in E$, let u have a variable $\diamond_u C_v$, which denotes a cached version of C_v. We can call a system *cache coherent* if $\forall u, v : (u, v) \in E : \diamond_u C_v = C_v$ [Herman 2003]. This means that whenever v assigns a value to C_v, node v also broadcasts the new value to all its neighbors. The moment a node u receives an updated value of C_v, it instantaneously (and atomically) updates $\diamond_u C_v$.

If we consider the operation of LMAC alone, the cache coherency property does not hold. Let us consider the case where two adjacent nodes v and u own the slots i and j respectively, where $j > i$. Suppose v first broadcasts its updated state information to u during its own slot i. Now consider the case where the state of v changes in slot l where $i < l < j$. In this case, v will be unable to broadcast its newly updated status to u, since the earliest time when it can transmit will be in slot $i + n$ where n is the number of slots in a single frame, that is, v would have to wait one entire frame. This delay in transmission prevents the cache coherence property from existing. Nevertheless, for $\mathcal{DOSA}$ we have the following lemma:

LEMMA 8.2. *Assuming no errors occur, nodes executing the $\mathcal{DOSA}$ algorithm on top of the LMAC protocol are all cache coherent.*

PROOF. In order to ensure cache coherence, $\mathcal{DOSA}$ carries out *pre-transmission state information processing* or PSIP. PSIP ensures that while a node updates its cache information the moment it receives updated state information from any adjacent neighbor, the node blocks any processing of the information in its cache until the point just before it transmits during its own slot. In other words, when a node receives updated state information from a neighboring node, it simply saves it. The node delays the processing of all the received state information until the point at which the node is just about to transmit during its own slot. This effectively means that a node broadcasts any updated state change the moment it is detected, and a node cannot experience a change in state at any time other than during its own slot. Thus, while LMAC alone does not support cache coherence, PSIP guarantees that the state information used by $\mathcal{DOSA}$ is always cache coherent. □

There are a few properties that $\mathcal{DOSA}$ possesses that ensure that it stabilizes within a finite time: (1)cache coherence (Shown in Lemma 8.2), (2)closure property, (3)convergence property. We describe the convergence and closure properties in greater detail below.

LEMMA 8.3. *$\mathcal{DOSA}$ demonstrates both the convergence and closure properties.*

PROOF. Recall from Section 7 that $\mathcal{S}$ denotes the set of all possible states. Let $\mathcal{M} \in \mathcal{S}$ (i.e., $\mathcal{S} \backslash \mathcal{M} = \mathcal{L}$) denote the set of all illegitimate states. In $\mathcal{DOSA}$, we consider all the nodes in the network that are not in the `Satisfied` state to belong to the set $\mathcal{M}$. Similarly, $\mathcal{L}$ represents all the nodes that have acquired the `Satisfied` state. $\mathcal{DOSA}$'s prioritization scheme, which is based on the combination of degree and ID of a node, implies that a node can always compute a unique priority. This ensures that as long as $|\mathcal{M}| > 0$, in every atomic step, at least one node is enabled and thus attains the `Satisfied` state, that is, if $n \in \mathcal{M}$, $|\mathcal{M}| = i$ and $|\mathcal{L}| = j$ in step r, then at step $r + 1$, $n \in \mathcal{L}$, $|\mathcal{M}| = i - k$ and $|\mathcal{L}| = j + k$ where $k > 0$. Thus over a finite number of steps, all nodes in $\mathcal{M}$ eventually converge towards $\mathcal{L}$.

Furthermore, since we assume that no communication errors or topology changes occur during the initialization process, a node that acquires the

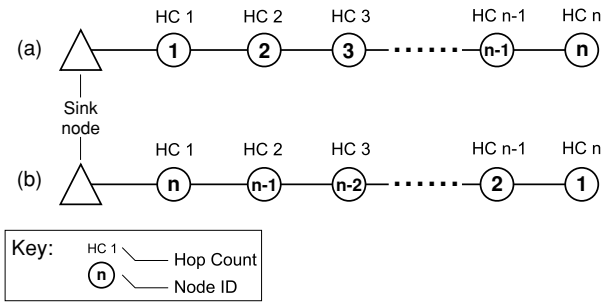

Fig. 9. (a)Worst and (b)best case scenarios for $\mathcal{DOSA}$ initialization.

`Satisfied` state remains in that state forever, regardless of the messages received. This is synonymous with the closure property. □

LEMMA 8.4. *Assuming no transmission errors or topology changes occur, given that d is the number of nodes in G'_{max}, which is the largest independent component in G, the time taken for all nodes in G to attain the* `Satisfied` *state, t_s (in frames) in $\mathcal{DOSA}$ during the initialization, is such that $d+1 \leq t_s \leq 2d-1$.*

PROOF. As the $\mathcal{DOSA}$ initialization phase can run in parallel in separate independent components within a single graph G, and since the time taken for initialization to complete is dependent on the number of nodes, we can conclude that, given a graph G, the initialization time is dependent on the cardinality of the largest independent component in G, that is, G'_{max}.

From Figure 9(a) it can be seen that initialization takes the longest time when nodes in G'_{max} are arranged such that the smaller the hop count from the sink node, the smaller the node ID. In this example, node $n-1$ will have the highest priority and so all the nodes will reach the legitimate state only when node 1 receives the *NodeStatus* message from node 2. Given that there are d nodes in all, this occurs in frame $2d-1$ assuming that the sink node transmits the *BeginSecondPhase* message to node 1 in frame 1.

From Figure 9(b) it can be seen that initialization takes the shortest time when nodes in G'_{max} are arranged such that the larger the hop count from the sink node, the smaller the node ID. Thus a node at hop count d only acquires the `Satisfied` state when it receives the *NodeStatus* message from its adjacent neighbor at hop count $d+1$. This occurs in frame $d+1$. □

LEMMA 8.5. *During the initialization of DOSA, every node in the network transmits a total of 3 messages.*

PROOF. For the $\mathcal{DOSA}$ initialization to complete, every node in the network needs to broadcast a *BeginSecondPhase* message, a NodeStatus message with the SatisfiedStatus field set to *FALSE* (broadcast when a *BeginSecondPhase* message is received), and finally a NodeStatus message with the SatisfiedStatus field set to *TRUE* when a node attains the `Satisfied` state. Note that the number of messages transmitted by a single node is independent of the size of the network. □

8.3.2 *Handling Message Corruption.* Up to now, we have assumed that all communication is error free. However, to make our analysis realistic, we now describe the steps taken by $\mathcal{DOSA}$ to ensure that it continues to operate normally even when transmission errors, due to poor link quality or topology changes, do occur.

A node uses the acknowledgement field in the CM section of a slot in LMAC to indicate whether it has successfully received an incoming message. Recall that since this field is in the CM section, every node transmits it once every frame. The number of bits in the acknowledgement field corresponds to the total number of slots used in a frame. Thus if a node n receives a message successfully from a particular neighbor m in slot i, a 1 is placed in the ith bit of the acknowledgement field in the CM section. Similarly, a '0' is placed in the ith bit if the incoming message received in slot i becomes corrupt. Node m can resend the message if it notices a 0 in the ith bit of the acknowledgement field of the CM received from node n.

Formally, we state that every node n uses a Boolean $b_n(m)$ for each neighbor m. For moving from statement G to A in $\mathcal{DOSA}$, we can then state $(\forall m : (n, m) \in E : b_n(m)) \wedge G \rightarrow A$. If n receives a message correctly from a neighbor m, n assigns $b_n(m) := \mathit{true}$. If the message gets corrupted, $b_n(m) := \mathit{false}$ for every m. Thus n blocks the execution of $\mathcal{DOSA}$ the moment it receives a corrupt message and only continues executing the program once it has correctly received messages from all the neighbors.

Additionally, up to this point we have assumed that no topology changes occur during the initialization process. We would like to point out that this assumption was made simply to allow the initialization mechanism to be explained in a simpler manner. If a topology change does occur, for example, a node disappears or reappears, $\mathcal{DOSA}$ makes use of the algorithms described in Sections 9.3 and 9.4 (which handle node removal and addition respectively) in order to ensure that the system continues to operate properly and eventually completes the initialization phase.

9. PERFORMANCE OF $\mathcal{DOSA}$

In this section we initially investigate the effectiveness of $\mathcal{DOSA}$ in several ways. First, we observe the reduction in the number of nodes generating readings as compared to raw data collection. We also illustrate through simulations, how this reduction in message transmissions translates into longer network lifetime and also improved data quality.

Following this, we describe the behavior of $\mathcal{DOSA}$ when a node dies or is added to the network.

9.1 Effectiveness of $\mathcal{DOSA}$ in Terms of Message Generation

The effectiveness of $\mathcal{DOSA}$ can be evaluated by observing the number of correlating nodes at any point in time, and comparing it with the raw data collection model, in which every node is involved in transmitting raw sensor readings, Figure 10(a). Let us consider the two graphs in Figures 10(b) and (c). The black nodes, representing correlating nodes in both graphs form maximal

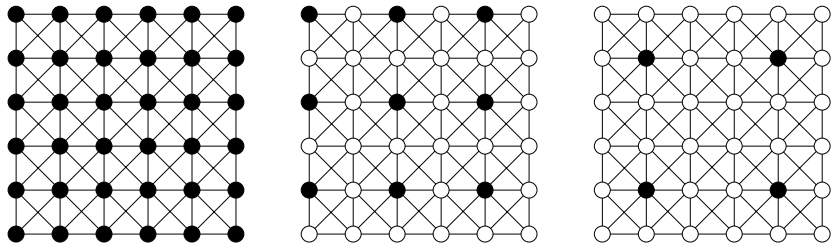

Fig. 10. Impact of cardinality of maximal independent set.

independent sets. However, it can be seen that the cardinality of the maximal independent set can vary greatly depending on the set of chosen nodes. This results in varying degrees of energy efficiency since a larger cardinality means lower efficiency as compared with raw data collection.

This then leads us to the following question: Given a particular graph, what is the maximum cardinality of the maximal independent set formed by $\mathcal{DOSA}$? This would essentially give us an estimation or bound on the worst case performance of $\mathcal{DOSA}$. Since computing the maximum maximal independent set of a given graph is NP-hard [Crescenzi and Kann 2005a], we take a covering approach to give a bound on the worst case performance of $\mathcal{DOSA}$.

LEMMA 9.1. *The worst case performance of $\mathcal{DOSA}$ can be guaranteed to result in a message reduction of at least* $(\frac{2nr^2}{xy} - 1) \times 100\%$ *compared with raw data collection in which n nodes are uniformly distributed in an area of dimensions* $x \times y$ *and every node has a circular transmission radius of* r.

PROOF. Let us divide the area $x \times y$ into m squares where,

$$m = \frac{xy}{2r^2} \tag{1}$$

Since the nodes are assumed to be randomly distributed, we may reasonably assume that nodes are present in all m squares, Figure 11. Note that this results in a worst-case estimation. Furthermore, we assume that exactly one node in every square forms part of a maximal independent set. We immediately see that it is not possible to have more than one node, that is part of the maximal independent set in a single square, because these extra nodes would be in range of the first chosen node. This consequently implies that the cardinality of the maximal independent set would be m. It would be impossible to increase the size any further by adding any more nodes. We can therefore conclude that the maximum cardinality of the maximal independent set created by $\mathcal{DOSA}$ is m. Thus, the percentage in message reduction of $\mathcal{DOSA}$ compared with the collection of raw data would be, $\frac{n-m}{m} \times 100$. This can then be simplified to $(\frac{2nr^2}{xy} - 1) \times 100\%$. □

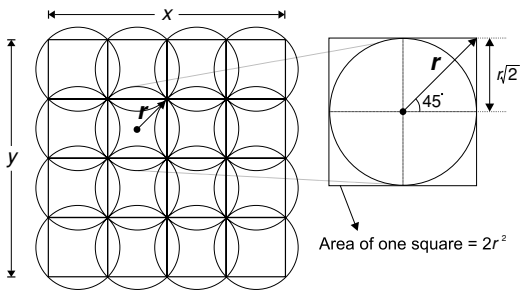

Fig. 11. Estimating the cardinality of the maximum maximal independent set generated by $\mathcal{DOSA}$.

As stated in Bulusu et al. [2001], network density, μ can be defined as follows:

$$\mu = \frac{n\pi r^2}{xy}. \tag{2}$$

Using equations 1 and 2, we can then state,

$$|I| \leq \frac{n\pi}{2\mu}, \tag{3}$$

where I is any independent set also including the one computed by $\mathcal{DOSA}$. We would like to indicate, however, that network density is approximately equal to average connectivity such that,

$$\frac{n\pi}{2\mu} \approx \frac{n\pi}{2(\rho - 1)}, \tag{4}$$

where ρ is the average connectivity. This result is used to plot the graph in Figure 12(b), which estimates the cardinality of $\mathcal{DOSA}$ as the average connectivity is varied.

The simulation results presented in Figure 12(a), show that even for a high cardinality, the number of correlating nodes is never greater than approximately 31%, thus resulting in a reduction in the number of message transmissions of approximately 69% compared with collecting raw data from every node in the network. This is true in cases where the average connectivity of the network is very low. As can be observed from Figure 12(a), the cardinality of the maximal independent set reduces further as the average connectivity of the network is increased. This is quite intuitive since node can be used to represent a larger number of adjacent neighbors as the connectivity increases. The average reduction in message transmissions due to $\mathcal{DOSA}$ compared with raw data collection, goes up to approximately 85% when the connectivity is increased to 11.

The prioritization scheme used in $\mathcal{DOSA}$ also has a large impact on the performance of the algorithm. We can observe two characteristics from the fact that $\mathcal{DOSA}$ gives the highest priority to the nodes with the largest connectivity. First, since nodes that have the highest degree in their local 1-hop neighborhood acquire the colors first, using a greedy approach, the cardinality of the

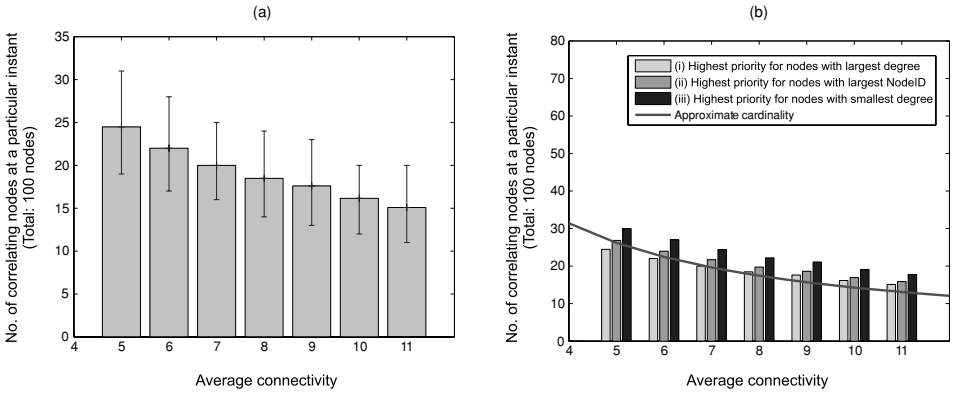

Fig. 12. (a) Impact of average connectivity on the number of correlating nodes at a particular instant (Total number of nodes in the network = 100), (b) Effect of prioritization scheme on cardinality of maximum independent set.

maximal independent set tends toward the minimum maximal independent set. In Figure 12(b) we illustrate the effects of using three different priority schemes: (1) Highest priority given to node with largest degree, (2) Highest priority given to node with largest node ID, and (3) Highest priority given to node with smallest degree. By following the same argument as in scheme (1), scheme (3) results in a maximal independent set that has a cardinality that is closer to the cardinality of the maximum maximal independent set. Scheme (2) however, due to its random nature, still results in a maximal independent set, but does not tend toward the minimum or the maximum cardinality. Note that the difference between the estimated cardinality and the actual results can be attributed to boundary effects.

It is important to note however, that while the minimum maximal independent set would result in an optimal solution (i.e., the smallest number of correlating nodes), and thus appear to be the most efficient in terms of energy efficiency, it is not something that $\mathcal{DOSA}$ strives to attain. At this point, we would like to remark that computing an optimal, that is minimum cardinality maximal independent set is NP-hard [Crescenzi and Kann 2005b]. Therefore, given the scarce resource limitations of WSNs, we resort to the presented, faster approach. However, in Nieberg [2006] it is shown that for wireless communication networks, the greedy strategy of $\mathcal{DOSA}$ results in a constant-factor approximation with respect to the cardinality of an optimal solution.

9.2 Effectiveness of $\mathcal{DOSA}$ in Terms of Network Lifetime and Data Quality

As mentioned previously, the transmission of raw sensor readings has a detrimental impact on network lifetime and also on data quality. The reduction in message generation described in the previous subsection naturally leads to improvements in both of these factors.

In this subsection, we have carried out simulations to illustrate the benefits of $\mathcal{DOSA}$ in terms of network lifetime and data quality. Note that we define

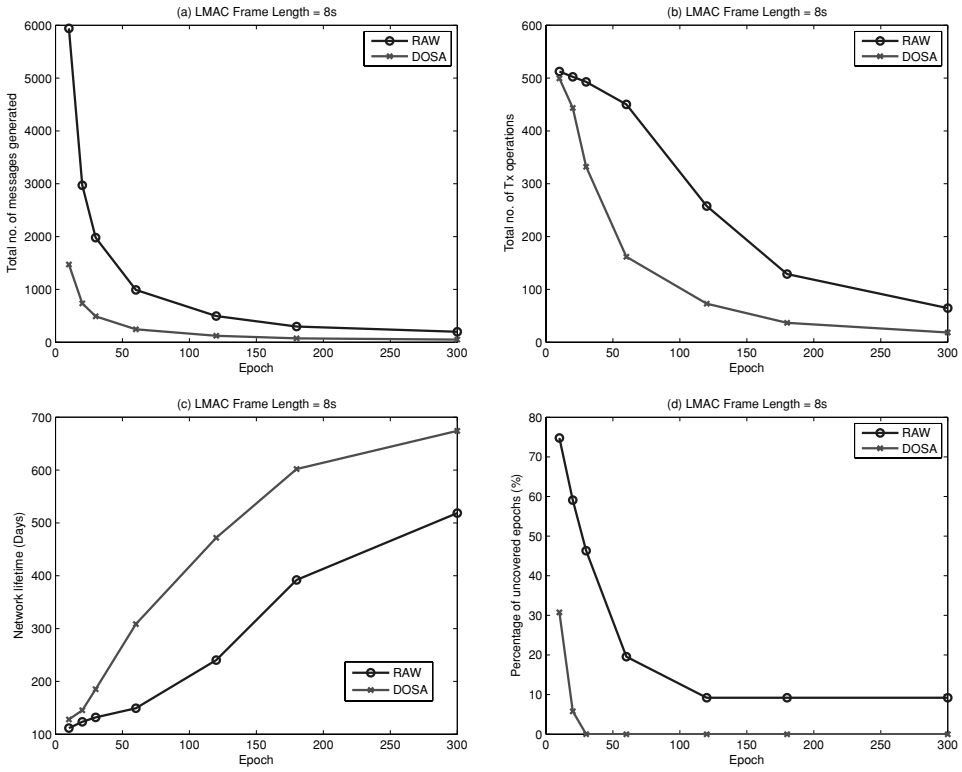

Fig. 13. (a) Total number of messages generated, (b) Total number of transmit operations, (c) Network lifetime, (d) Percentage of uncovered epochs.

network lifetime as the total time taken before the death of the first node in the network. In the simulations, LMAC uses a frame length of 8 seconds. We use the following specifications based on the RFM TR1001 [RF Monolithics 2007] transceiver: transmit -36mW, receive -11.4mW and standby -0.7μW to compute network lifetime. We also assume that correlations between sensor readings remain constant during this interval. All results have been collected over 10 minutes and have been averaged over 100 topologies where each topology consists of 100 nodes. Readings for the various graphs have been collected at the following epochs (in seconds): 10, 20, 30, 60, 120, 180, 300.

Figure 13(a) shows the total number of sensor readings that are generated during a 10 minute interval using both data collection techniques. Figure 13(b) shows the total number of transmit operations performed by all the nodes in the network for the entire duration of the simulation. One can clearly see that Figures 13(a) and 13(b) do not have similar shapes. This is primarily because both raw data collection and $\mathcal{DOSA}$ experience heavy message losses for high sampling rates. The left-hand side of the graphs in Figure 13(b) tend toward each other as the limit of the maximum throughput of LMAC is nearly reached.

It is this same characteristic that produces the shape of the network lifetime graph in Figure 13(c). Since the total number of message transmissions is nearly the same for both data collection methods at high sampling rates, the network lifetime is also quite similar. It can be seen from Figure 13(c) that $\mathcal{DOSA}$ can help network lifetime improve by up to 83.5% (Epoch = 120s) as compared with raw data collection.

Apart from helping to improve network lifetime, $\mathcal{DOSA}$ also has a significant positive impact on the quality of data collected. When analyzing dropped messages for both data collection scenarios, it is important to realize that every message generated under the $\mathcal{DOSA}$ scheme carries a lot more weight than a single message in the raw data collection process. This is because a single sensor reading transmitted by a node n under the $\mathcal{DOSA}$ scheme, represents not only the reading of n but also those of its adjacent neighbors. For this reason, we analyze data quality by observing the number of epochs that are not represented at the sink rather than simply counting the number of dropped messages. As an example, suppose a message generated by node n representing its own reading and that of its neighbors, q, r, and s for the epoch E, is lost on the way to the sink due to a buffer overflow event. This would mean that during epoch E, the sink would not have any readings for nodes n, q, r, and s. Based on this example, we present the results of data quality in Figure 13(d). At high sampling rates for example, when the Epoch is 10s, raw data collection results in approximately 75% uncovered epochs while $\mathcal{DOSA}$ results in only 30% uncovered epochs. The percentage of uncovered epochs under $\mathcal{DOSA}$ quickly reduces to 0 and remains there as the sampling frequency is reduced. For raw data collection however, the percentage of uncovered epochs levels off at about 10%. We now explain this leveling-off characteristic.

Usually, a node drops messages when its buffers get filled up. Thus the higher the sampling rate, (i.e., the smaller the value of the Epoch) the larger the proportion of nodes in the network that experience buffer overflows. This naturally also increases the number of lost messages and in turn the percentage of uncovered epochs. However, as the sampling rate is reduced, the number of nodes experiencing buffer overflows might not continue decreasing to zero. In most topologies, due to the simultaneous generation of messages by all nodes in the network, there will be a certain set of nodes that will always experience buffer overflows and will only allow a fixed number of messages to successfully traverse toward the root. Thus for low sampling rates, in every epoch, only a fixed number of messages will reach the root regardless of the chosen epoch. It is this characteristic that causes the percentage of uncovered epochs to level off for low sampling rates.

One may assume that the results from the graphs shown in Figures 13(a)–(d) clearly show that $\mathcal{DOSA}$ has a benefit only for applications that require low sampling rates. However, this is not the case. For applications that require high sampling rates and therefore high data rates, LMAC can easily be tuned such that one frame has a length of 2 seconds instead of 8 seconds. We illustrate the results of network lifetime and percentage of uncovered epochs in Figures 14(a) and (b). Note that these graphs also display the same characteristics mentioned previously.

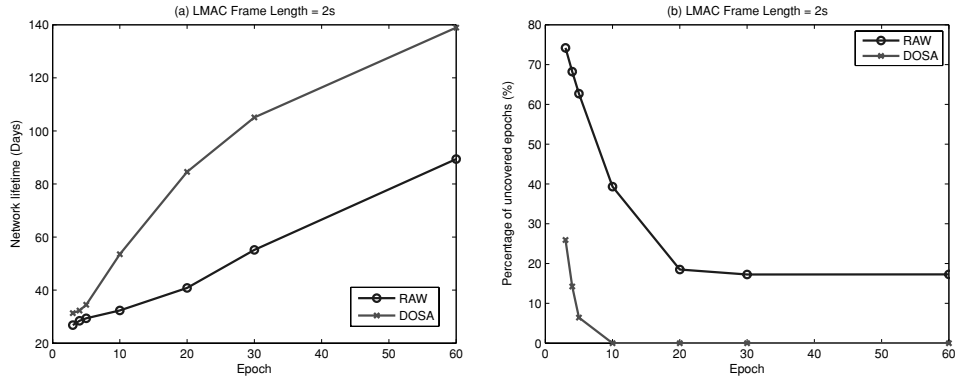

Fig. 14. (a) Network lifetime, (b) Percentage of uncovered epochs.

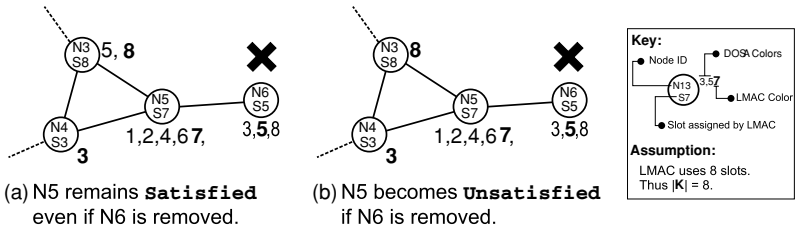

Fig. 15. Two possible scenarios when a node dies.

9.3 Coping with a Dead Node

Because the death of a node can be a common occurrence in WSNs, it is important that any algorithm designed for WSNs be able to cope with such events. $\mathcal{DOSA}$ ensures that a node is able to reorganize the scheduling algorithm within a finite amount of time autonomously, the moment a neighboring node disappears from the network. It does this by retrieving cross-layer information from the underlying LMAC protocol, that is, the death of a node triggers an update in the LMAC Neighbor Table.

The death of a node leads to the disappearance of the colors that were owned by the dead node. This can lead to two possible scenarios. First, it may be possible that one or more neighbors of the dead node still satisfy constraints 1 and 2 since the colors that have disappeared with the dead node are also present in its neighboring nodes. This is shown in Figure 15(a). In this case, the `Satisfied` neighboring nodes continue to maintain their existing schedules and do not transmit any messages. Note, however, that while their color assignments are invariant, the degree of the neighbors of the dead node does reduce by one. It is important that nodes that are one hop away from the neighbor of the dead node are informed about this change of degree because this information would be required in case any schedules need to be reassigned in the future, due to certain network perturbations. However, since our design takes advantage of cross-layer information from LMAC, explicit message transmissions are not required in order to relay information regarding a change of degree of a node. This information is instead automatically disseminated through the periodic

broadcast of the CM section of the LMAC protocol. Recall that the CM section transmitted by a node contains an occupied slot list, which lists the slots occupied by the node and its one hop neighbors. Thus, this information can also be used to deduce the degree of a node.

In the second scenario, shown in Figure 15(b), the death of a node may result in one or more neighboring nodes ending up with certain missing colors. Since these nodes no longer satisfy constraints 1 and 2, the nodes switch to the `Unsatisfied` state and broadcast this change in status to their immediate one-hop neighborhood. A node then waits one frame to see if there are any other neighboring nodes that are also in the `Unsatisfied` state. Note that waiting one frame allows the node to hear from all its neighbors in case they have any status change to report. After waiting one frame, if the node with the missing color(s) has the highest priority among all the unsatisfied nodes it will acquire all the colors it lacks. This whole process is described in Algorithm 2. If a node lacks a color but does not have the highest priority, it continues to wait until all its higher priority unsatisfied neighbors have become satisfied. In other words, the node continues to execute Algorithm 1 every time it receives a *NodeStatus* message until it finally acquires the `Satisfied` state.

Algorithm 2. $\mathcal{DOSA}$—Coping with the loss of a node

Input: LMAC Neighbor Table indicates at least one missing node
Output: NodeStatusMSG(Degree, SatisfiedStatus(FALSE & TRUE), ColoursOwned)/NIL

1: UPDATE(LocalInfoTable, v)
2: **if** MissingColours(v) = *TRUE* (i.e., SatisfiedStatus(v)=*FALSE*) **then**
3: BROADCAST NodeStatusMSG(Degree, SatisfiedStatus(FALSE), ColoursOwned)
4: WAIT one frame
5: Compute PRIORITY(v)
6: **if** Priority(v) = Highest **then**
7: $C_v \leftarrow \mathbf{K} \backslash C_{\Gamma'(v)}$
8: ColorsOwned $\leftarrow C_v$
9: SatisfiedStatus $\leftarrow$ *TRUE*
10: UPDATE(LocalInfoTable, v)
11: BROADCAST NodeStatusMSG(Degree, SatisfiedStatus(TRUE), ColoursOwned)
12: **end if**
13: **end if**

In order to explain the timing bounds of $\mathcal{DOSA}$ when a node dies, we use the same argument as in the proof of Lemma 8.4. We can extend this lemma as follows:

LEMMA 9.2. *When a node v with x neighbors dies, the maximum time taken for all nodes to converge towards the* `Satisfied` *state is* $x + 1$ *frames where* $x \leq |\mathbf{K}| - 1$.

PROOF. In the worst case, all the nodes of a dead neighbor switch to the `Unsatisfied` status and broadcast this change of state. Every `Unsatisfied`

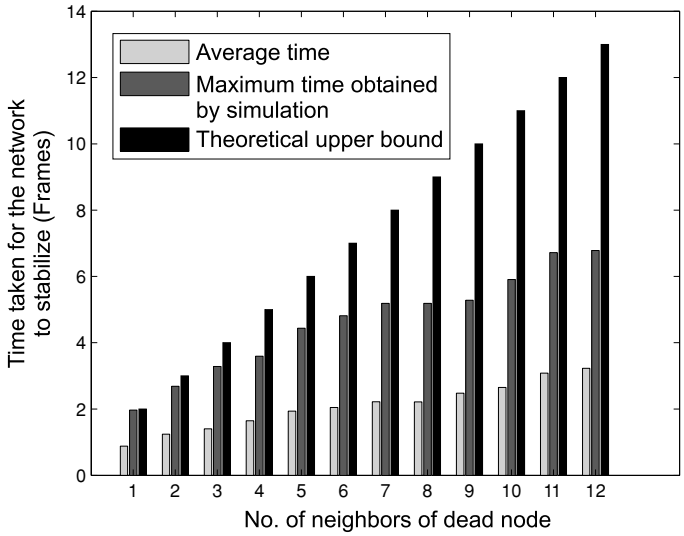

Fig. 16. Time taken for a network to stabilize once a node has been removed from the network.

neighbor then waits for its higher priority `Unsatisfied` neighbor to switch to the `Satisfied` state before acquiring the `Satisfied` state itself. This situation is then identical to situation mentioned in Lemma 8.4 and thus the same timing bounds apply. □

We have carried out simulations to compare typical network stabilization times when a node is removed, with the bounds presented above. For every topology with 100 nodes (including one sink node), we first removed one node, waited for the network to stabilize, (i.e., for all nodes to reacquire the Satisfied state), and then added it back to the network. This operation was carried out for all the 99 nodes in every topology. Thus there were 9900 node removal-and-addition cycles. The results presented in the following sections have been obtained over these 9900 cycles. Note that the average connectivity of the nodes in every topology is 8.

Figure 16 presents the time durations taken for the network to stabilize once a node was removed from the network. Generally, the average stabilization time increases with the number of neighbors of the dead node. This is also true for both the maximum stabilization times and the theoretical upper bound presented previously. However, as the number of neighbors of the dead node increases, the rate of increase of the average and maximum durations decreases. This is because the probability of having a large number of nodes arranged in an increasing manner (e.g., Figure 9(b)) reduces as the number of neighbors increases. Thus in real life settings, a higher density network does not necessarily recover more slowly when a node is removed. In fact, according to the simulation results, the worst case recorded during a simulation in which the dead node has 12 neighbors, would be approximately 50% of the theoretical upper bound.

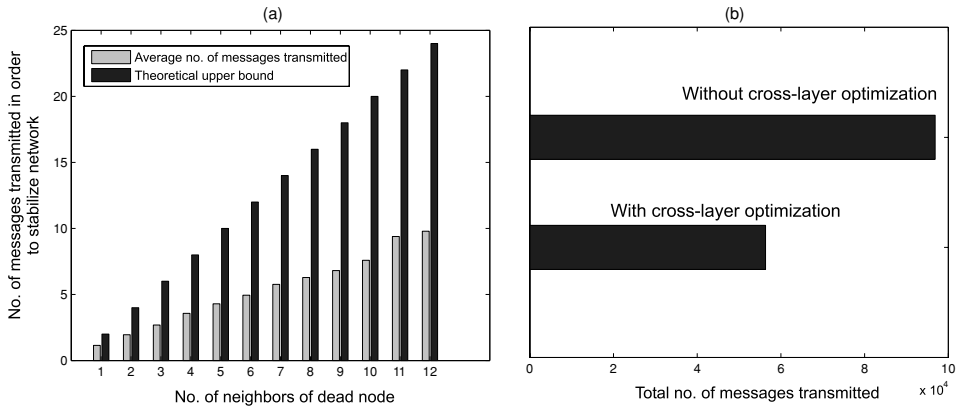

Fig. 17. (a) Number of messages transmitted in order to stabilize the network once a node dies, (b) Number of messages transmitted over 9900 runs with and without cross-layer information.

LEMMA 9.3. *When a node v with x neighbors dies, the maximum possible number of messages that may be transmitted is* $2x$*, where* $x \leq |\mathbf{K}| - 1$.

PROOF. As stated in Lemma 9.2, every in the worst case, all x neighbors may become `Unsatisfied` when node v dies. Generally, every affected node (i.e., every node with missing colors) initially transmits one *NodeStatus* message, with the status set to `Unsatisfied` the moment node v dies. Finally, when a node acquires the Satisfied state, it transmits another *NodeStatus* message that reflects this change. Note that once a particular node acquires the Satisfied state, it remains in that state indefinitely. Thus, the maximum possible number of messages that may be transmitted is $2x$. □

Figure 17(a) shows the average number of messages transmitted when a node with a particular number of neighbors is killed. Note that if all the neighbors become `Unsatisfied` due to the death of the node, every single neighbor will need to transmit two messages, as explained earlier. In random network topologies, however, the average number of messages transmitted when a node dies is less than 50% of the maximum theoretical upper bound indicated in Lemma 9.3.

The simulation results presented in Figure 17(b) show the benefit of having $\mathcal{DOSA}$ use underlying cross-layer information from LMAC. The total number of messages transmitted by all the nodes was compared over 9900 node deletions, with and without cross-layer information being used. When it is not used, every neighbor of the dead node has to transmit a *NodeStatus* message, regardless of its status. The results indicate a savings of up to 42% when cross-layer information is used.

LEMMA 9.4. *When a node v dies, only its first order neighbors may be affected, that is, may switch from the* `Satisfied` *to the* `Unsatisfied` *state.*

PROOF. The death of node v can only result in the adjacent nodes experiencing missing colors and subsequently switching to the `Unsatisfied` state.

`Unsatisfied` nodes then occupy colors they are lacking and thus ensure that their choice of colors will not cause any color collisions with their neighbors. Also, a node that is `Satisfied` and receives a *NodeStatus* message, does not switch its status, as long as Constraints 1 and 2 are met. Thus, nodes that are two or more hops away from node v cannot experience a change of state when node v dies. □

9.4 Coping with a New Node

As we illustrated in the previous subsection, when a node dies, $\mathcal{DOSA}$ can only execute one fixed set of steps to ensure that the scheduling scheme stabilizes within a finite amount of time. The node addition operation, however, is a little more involved because the set of steps taken by $\mathcal{DOSA}$ depends on the events that occur when a new node v is added to the network. For example, node v may detect an LMAC collision, or may cause colliding or missing colors in neighboring nodes, or may even cause a combination of these events. Different permutations and combinations of these events can cause the network to react in a multitude of ways. This makes it impractical to analyze the performance bounds of every particular sequence of events that causes the network to react in a certain manner. Instead, in order to simplify matters, we categorize all the permutations and combinations of events according to how far the network disturbance propagates when node v is added to the network. For example, there may be a certain combination of events that would cause nodes that are up to two hops away from v to switch to the `Unsatisfied` state. Similarly, there might be other events that would cause the network disturbance to propagate to the 3rd order neighborhood of node v.

We first begin by listing and describing the various events that could occur when node v is added. We have included an example in Figure 18 to illustrate how the various events might occur. (Note: The terms "LMAC slot" and "LMAC color" are equivalent and thus can be used interchangeably.):

(1) *Collision between LMAC slots*. This occurs when the new node v detects a collision between two or more of its adjacent neighbors. Each colliding neighbor then needs to give up the colliding slot and choose a new slot.

(2) *Collision between LMAC color (slot) and $\mathcal{DOSA}$ colors*: When a node n that is d hops away from node v chooses a new LMAC color, it causes a collision at an adjacent node m that is $d + 1$ hops away from node v, assuming that node m owns the $\mathcal{DOSA}$ color that is equal to the new LMAC color chosen by node n. Note that if $d = 0$ then $n = v$. Also $0 \leq d \leq 1$ since, using LMAC, node v can only detect slot collisions among its first order neighbors.

(3) *Missing $\mathcal{DOSA}$ colors*: A node m that is $d + 1$ hops away from the new node v experiences missing $\mathcal{DOSA}$ colors if an adjacent node, n that is d hops away from node v, gives up a $\mathcal{DOSA}$ color due to a color collision. Thus, a color collision at a node that is d hops away from v can only cause missing colors at adjacent neighbors that are $d + 1$ hops away. Since a missing

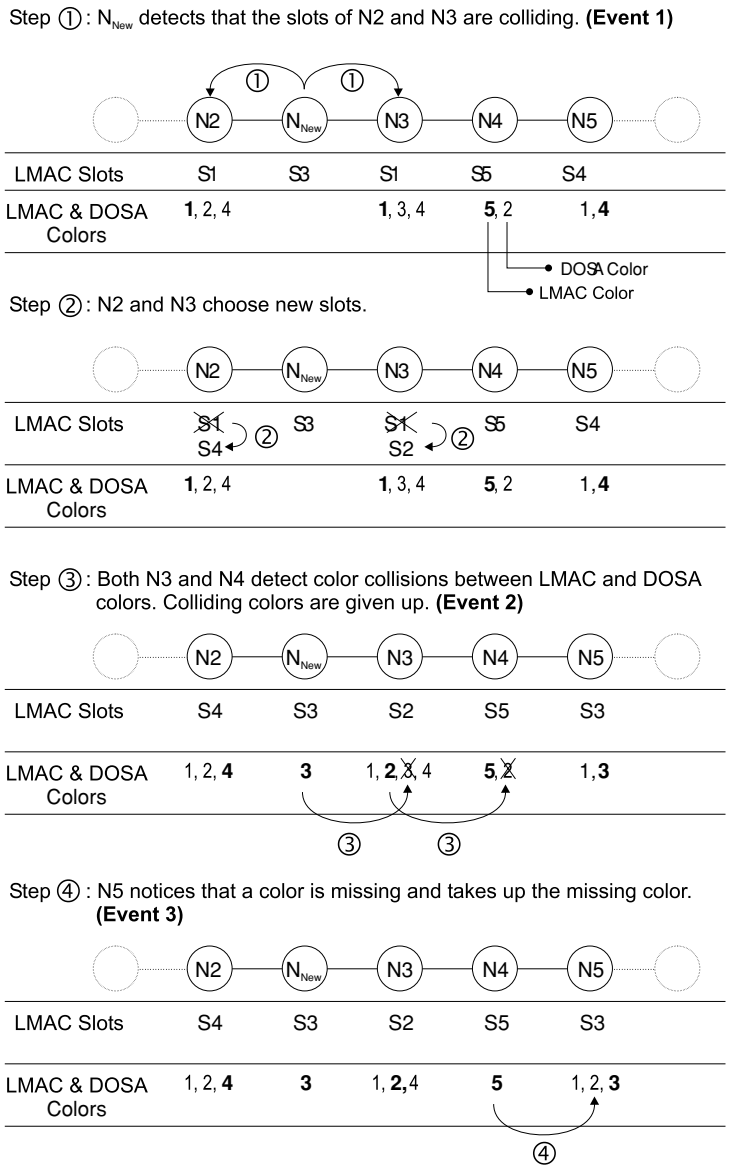

Fig. 18. An example of how certain events occur when a new node is added.

color event at a node that is $d + 1$ hops away from v can only happen in combination with a color collision event at an adjacent node that is d hops away from v, and since a color collision can only occur in the 1st order neighborhood of v, we can conclude that $d \geq 1$ if a missing color event occurs.

(4) *Node obtains Highest priority (due to largest degree in local neighborhood).* A node n that is d hops away from the new node v, realizes that it has

the highest degree in its local neighborhood after the addition of node v. This causes node n to obtain the highest priority and thus acquire all colors except the LMAC colors of its adjacent neighbors. As this event can occur either at the new node itself (i.e., $d = 0$ and $n = v$) or at a node that is adjacent to v we can conclude that $0 \leq d \leq 1$.

However, the addition of a node does not cause a domino effect in $\mathcal{DOSA}$. The reason for this is explained in the following lemma:

LEMMA 9.5. *When a node v is added, all nodes beyond the 3rd order neighborhood of v can be guaranteed to be unaffected, that is nodes that are more than three hops away will always remain in the* `Satisfied` *state, regardless of the sequence of events that occur after the addition of node v.*

PROOF. We know from the four events listed in Figure 18 that a node can switch to the `Unsatisfied` state when it experiences either a color collision or a missing color event. As explained previously, a missing color can only occur one hop away from a color collision. We also know that a color collision can occur up to a maximum of two hops away from v. This implies that a missing color event can only happen in a node that is three hops away from v. Thus, nodes more than three hops away from v cannot be affected by its addition. □

While Lemma 9.5 shows that a node addition cannot cause $\mathcal{DOSA}$ schedules to be disturbed more than three hops away from the newly added node, we also carry out simulations to analyze the actual effects of node addition.

We perform simulations over 100 topologies each consisting of 100 randomly placed nodes. For every topology, we add a node randomly to the network and collect the required statistics, for example, network stabilization time, depth of network disturbance, and so forth. This procedure is carried out for 100 nodes per topology. Thus the following results presented have been averaged out over 10,000 node additions.

Our results presented in Figure 19 indicate that in approximately 92% of the simulations, the network disturbance was restricted to within the second order neighborhood of the newly added node. In 8% of the simulations, none of the neighbors were affected. Third order neighbors were only affected in less than 1% of the simulations.

Regardless of which sequence of events occurs once a new node joins the network, initially, there are a few common steps that $\mathcal{DOSA}$ takes. Once these common steps are complete, the next set of steps taken depends on how far the network disturbance propagates. We first explain the initial common steps below.

When a new node n is added to the network, LMAC ensures that node n occupies a slot that is not used by any other node within two hops of n (Figure 20, Step 1). Node n then begins broadcasting its CM section. Neighboring nodes then detect node n and add its entry into their LMAC Neighbor Tables (Figure 20, Step 2). We explain the remaining steps taken by $\mathcal{DOSA}$ by referring to a neighboring node of node n as node v. This is also explained in Algorithm 3.

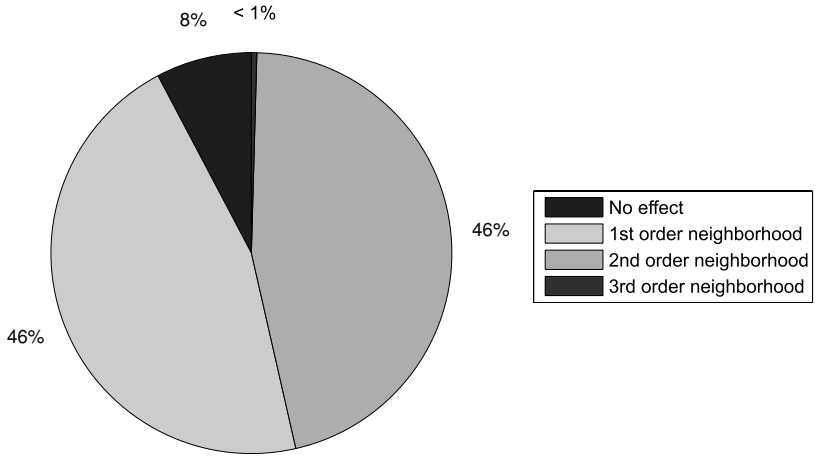

Fig. 19. Simulation results showing how often a newly added node affects neighboring nodes that are 1 to 3 hops away from the new node.

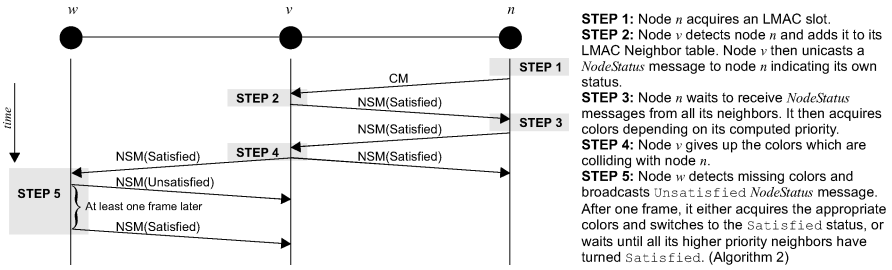

Fig. 20. Timing diagram for addition of a new node, n (Node v is adjacent to n and node w is 2 hops from n.).

Algorithm 3. $\mathcal{DOSA}$—Coping with a new node

Input: NodeStatusMSG(Degree, SatisfiedStatus(TRUE), ColoursOwned)
Output: NodeStatusMSG(Degree, SatisfiedStatus(TRUE), ColoursOwned)

```
1:  UPDATE(LocalInfoTable, n)
2:  if LocalInfoTable contains entries from ALL adjacent nodes then
3:    Compute PRIORITY(n)
4:    if PRIORITY(n) = Highest then
5:      C_n ← K\C_{Γ'_LMAC(n)}
6:    else
7:      C_n ← K\C_{Γ'(n)}
8:    end if
9:    UPDATE(LocalInfoTable, n)
10:     BROADCAST NodeStatusMSG(Degree, SatisfiedStatus(TRUE), ColoursOwned)
11:   end if
```

As soon as node v, which is already in the `Satisfied` state, detects the presence of node n, it unicasts a *NodeStatus* message to node n (Figure 20, Step 2). Node n then waits to receive node status messages from all its adjacent neighbors (Figure 20, Step 3). Note that by this stage, n would know about the existence of all its adjacent neighbors since otherwise it would not have been able to obtain an LMAC slot.

From this point onward, the actions taken by $\mathcal{DOSA}$ are dependent on the sequence of events that occur. Once n has received *NodeStatus* messages from all its adjacent neighbors, it checks to see if it has the highest priority within its immediate neighborhood. If n finds that it has the highest priority, it acquires all colors except the LMAC colors of the adjacent neighboring nodes. This helps to ensure that over time, even if the network topology changes, the cardinality of the maximal independent set continues to be low. In other words, the sink node would be able to predict the readings of a larger number of nodes when a node with a higher degree is chosen to perform the correlations, rather than a node with a very small degree. This leads to greater energy savings.

If node n realizes that it does not have the highest priority, however, it simply acquires all the colors that it is currently lacking. Since node n has now satisfied constraints 1 and 2, it broadcasts a *NodeStatus* indicating that it is `Satisfied`.

At this stage, a neighboring node v that receives the *NodeStatus* message from n, may detect that certain colors are colliding (Algorithm 4, Line 3). This would mean that Constraint 1 is not being met. Thus node v gives up the colors that are colliding with node n, attains the `Satisfied` state, updates its own *LocalInfoTable*, and informs all of its neighbors through a broadcast operation (Figure 20, Step 4).

Algorithm 4. $\mathcal{DOSA}$—Colliding colors due to a new node

```
1:  UPDATE(LocalInfoTable, n)
2:  if LocalInfoTable contains entries from ALL adjacent nodes then
3:    if C_n ∩ C_v ≠ φ then
4:      C_v ← C_v \ (C_n ∩ C_v)
5:      UPDATE(LocalInfoTable, v)
6:      BROADCAST NodeStatusMSG(Degree, SatisfiedStatus(TRUE), ColoursOwned)
7:    end if
8:  end if
```

As v has had a change in colors, it is possible that a node w, that is adjacent to v but not to n (i.e., w is two hops away from n), may become `Unsatisfied` (due to the Node Status message transmitted in Step 4 of Figure 20). Node w can then resolve the situation by executing Algorithm 2, which allows it to recover when certain colors are found to be missing (Figure 20, Step 5).

Next we present the upper bounds of $\mathcal{DOSA}$ in terms of the amount of time taken to stabilize the network and of the number of message transmissions when a new node is added. Since the addition of a node can result in the occurrence of several events, we break down the analysis into 5 possible groups, based on the depth of propagation of the network disturbance, as shown in

Table I. Upper Bounds for Time and Message Transmission When a Node is Added

	No effect (Group 1, 8%)	1st order (Group 2, 46%)
Event type	-	Color collision
Max. time (Frames)	≤ 1	≤ 1
Max Msgs Tx	$= \lvert\Gamma'_1(v)\rvert + 1$	$\leq 2\lvert\Gamma'_1(v)\rvert + 1$
	2nd order	
Event type	Color collision (Group 3, <1%)	Missing color (Group 4, 46%)
Max. time (Frames)	≤ 2	$2 + \lvert\Gamma'_2 \setminus \Gamma'_1\rvert$
Max Msgs Tx	$\leq 2\lvert\Gamma'_1(v)\rvert + 1 + \lvert\Gamma'_2 \setminus \Gamma'_1\rvert$	$\leq 2\lvert\Gamma'_1(v)\rvert + 1 + 2\lvert\Gamma'_2 \setminus \Gamma'_1\rvert$
	3rd order	
Event type	Missing color (Group 5, <1%)	
Max. time (Frames)	$3 + \lvert\Gamma'_3 \setminus \Gamma'_2\rvert$	
Max Msgs Tx	$\leq 2\lvert\Gamma'_1(v)\rvert + 1 + 2\lvert\Gamma'_2 \setminus \Gamma'_1\rvert + 2\lvert\Gamma'_3 \setminus \Gamma'_2\rvert$	

Table I. Note that these five groups encompass all the possible sequences of events that can happen due to the addition of a new node, for example, colliding LMAC slots, a new node acquiring the highest priority, and so forth. Thus for example, the *3rd order, color collision* event is not listed in Table I because such an event cannot happen for the reasons stated in the list of events presented previously in this section.

We refrain from explaining the derivations for the theoretical upper bound times for network stabilization shown in Table I, since they have been derived using the same arguments presented earlier in Lemma 8.4. However, in order to present a more concise explanation, we present the theoretical upper bounds for the number of message transmissions using the five rules listed below:

Rule 1: When a node that has already acquired $\mathcal{DOSA}$ colors detects a new neighbor node v, it unicasts one *NodeStatus* message to node v.

Rule 2: A new node v broadcasts one *NodeStatus* message once it has acquired its LMAC slot, has resolved all LMAC collisions amongst its neighbors, and has received *NodeStatus* messages from all its neighbors.

Rule 3: A node that acquires a new LMAC color that is not listed in its existing list of DOSA colors, broadcasts one *NodeStatus* message.

Rule 4: A node that experiences a color collision event transmits one *NodeStatus* message.

Rule 5: A node that experiences a missing color event transmits two *NodeStatus* messages: the first to indicate that the node is Unsatisfied due to the missing color(s), and the second to indicate the node is Satisfied after it has acquired the appropriate colors.

We now illustrate how these rules can be used to work out the upper bound for the number of message transmissions for the *3rd order, missing color* case:

Step 1: First order neighbors of new node v transmit a *NodeStatus* message after detecting it. ($\lvert\Gamma'_1(v)\rvert$ messages, *Rule* 1)

Step 2: A new node transmits a *NodeStatus* message after acquiring colors. (one message, *Rule* 2)

Step 3: Every first order neighbor acquires new colors and broadcasts one *NodeStatus* message. ($|\Gamma'_1(v)|$ messages, *Rule* 3)

Step 4: Every second order neighbor experiences a color collision event and broadcasts one *NodeStatus* message. ($|\Gamma'_2(v) \setminus \Gamma'_1(v)|$ messages, *Rule* 4)

Step 5: Every third order neighbor experiences a missing color event and broadcasts two *NodeStatus* messages. ($2(|\Gamma'_3(v) \setminus \Gamma'_2(v)|)$ messages, *Rule* 5)

Thus,

Upper bound for total number of message transmissions for the *3rd order, missing color* case = $2(|\Gamma'_1(v)|) + 1 + |\Gamma'_2(v) \setminus \Gamma'_1(v)| + 2(|\Gamma'_3(v) \setminus \Gamma'_2(v)|)$.

Our simulations indicate that Groups 2 and 4 of Table I occur in 92% of all the 10,000 simulation runs while Group 1 occurs in 8% of the cases. Groups 3 and 5 however, occur in less than 1% of the cases. Thus we present the simulation results only for Groups 2 and 4, since they represent a more significant percentage of the various events that may occur.

We first consider the first order color collision results. Figure 21(a) shows that regardless of the number of 1^{st} order neighbors a new node has, the network stabilization time remains within 1 frame. This coincides with the bounds stated in Table I. Figure 21(b) shows that only around 1% of all the 1^{st} order color collision cases resulted in scenarios in which the number of messages transmitted was approximately 90–100% of the upper bound for message transmissions when a new node is added. In nearly 50% of the cases, the number of messages transmitted was approximately 60% of the upper bound.

Next, we consider the second order missing color results. Figure 21(c) shows that approximately 92% of time, the amount of time taken for network stabilization when the second order nodes experience a missing color event, was less than 40% of the upper bound. Figure 21(d) shows that in nearly 90% of cases, the number of messages transmitted was less than 60% of the upper bound. Notice that the results in Figure 21(b) tend closer to the upper bound than those presented in Figure 21(d). This is because, while the results in Figure 21(b) only require the first order nodes to be affected, the results in Figure 21(d) involve both the first and second order nodes. Naturally, the probability of affecting nodes in both the first and second orders is lower than that of nodes in only the first order.

The overall performance of $\mathcal{DOSA}$ for node addition is presented in Figures 21(e) and 21(f). Figure 21(e) and Figure 21(f) show the distributions of the number of messages transmitted and the amount of time taken for $\mathcal{DOSA}$ to stabilize once a new node is added to the network. It can be seen that in the majority of the cases the network stabilizes within four frames.

10. RELATED WORK

Techniques used to extract data from wireless sensor networks can be classified into three separate categories: (1) *snap-shot* queries, (2) *event-based* queries, and (3) *long-running* queries. Snap-shot queries are typically used when the

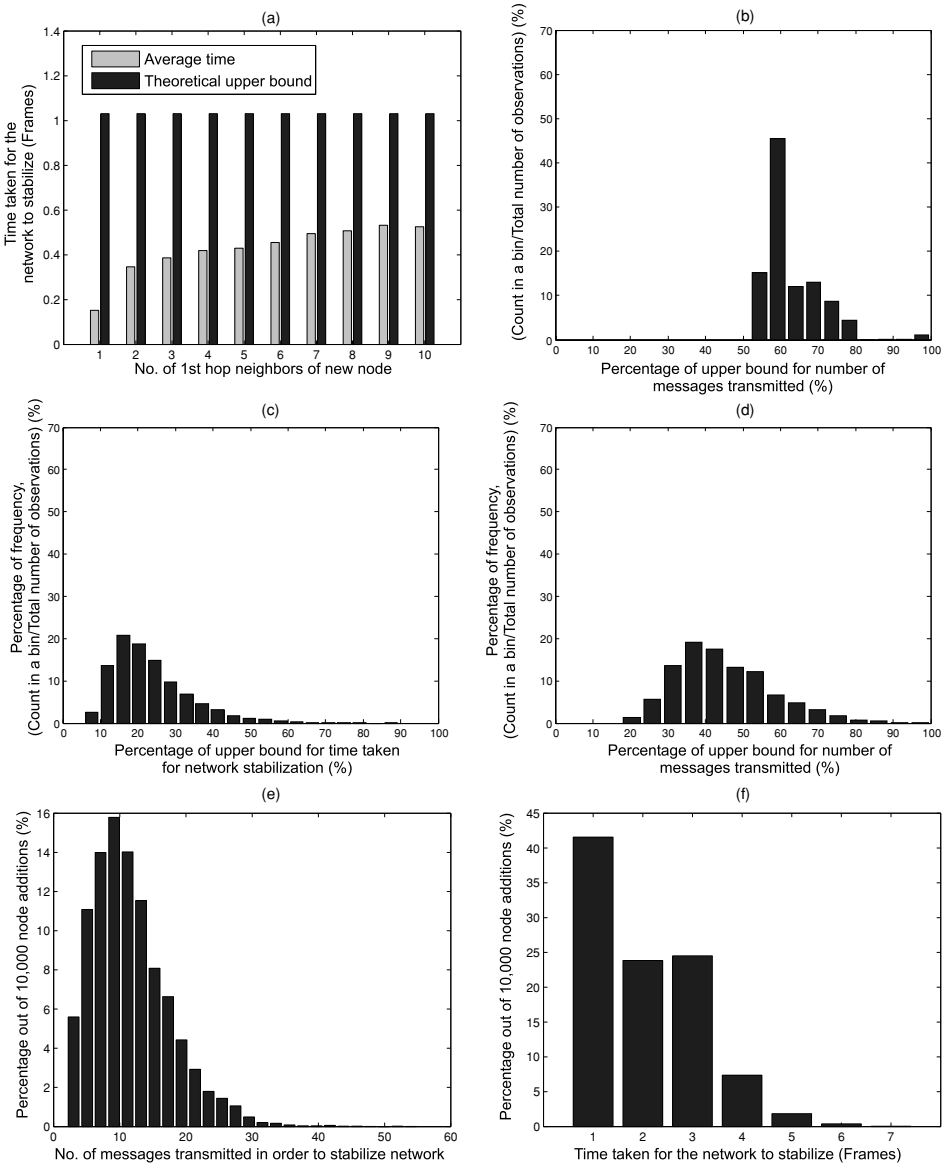

Fig. 21. (a) Time taken for a network to stabilize once a node has been added to the network, (b) How often the upper bound of the number of messages transmitted for the *1st order color collision* event is reached when a new node is added, (c) How often the upper bound of the time taken for network stabilization for the *2nd order missing color* event is reached when a new node is added, (d) How often the upper bound of the number of messages transmitted for the *2nd order missing color* event is reached when a new node is added, (e) Distribution of number of messages transmitted when a new node is added to the network, (f) Distribution of time taken to stabilize network when a new node is added to the network.

user sends in a query in order to retrieve instantaneous results that reflect the state of the sensors in the network at a certain point in time [Deshpande et al. 2005; Ratnasamy et al. 2003; Greenstein et al. 2003; Ganesan et al. 2003; Coman et al. 2005]. Event-based queries, on the other hand, are completely dependent on the environment that is being monitored, that is sensor readings are only transmitted to the sink node if an interesting event has taken place [Intanagonwiwat et al. 2003; Begum et al. 2004; Vuran et al. 2004; Vuran and Akyildiz 2006; Manjeshwar and Agrawal 2001]. Readings from long-running queries are obtained using a sampling frequency specified within a query injected into the network by the user [Madden et al. 2005; Yao and Gehrke 2002; Intanagonwiwat et al. 2003; Sharaf et al. 2004; Emekci et al. 2004; Liu et al. 2007; Chu et al. 2006].

Our application at AIMS specifically requires long-running queries. However, long-running queries can be resolved in various ways. There are long-running queries that extract every single reading acquired by all the sensors in the network. We refer to this as raw data collection. This naturally is not a feasible technique for energy-constrained WSNs due to excessive energy consumption, bottlenecks, reduction in data quality, and so forth. Since these problems were identified in the earlier days of sensor network research, a greater emphasis was placed on in-network processing, that is processing the acquired data within the network before transmitting it to the sink node. For example, in Directed Diffusion [Intanagonwiwat et al. 2003], a node may use a filter to prevent duplicate notifications of an event from being reported numerous times to the sink node. TinyDB [Madden et al. 2005] and COUGAR [Yao and Gehrke 2002] on the other hand, suggest aggregating data by executing aggregation operators (e.g., MIN, MAX, SUM, COUNT, AVERAGE) within the network. TiNA [Sharaf et al. 2004] presents improvements over TinyDB and COUGAR by taking advantage of temporal correlations of sensor readings.

However, such in-network processing techniques are not generally suitable for many environmental monitoring projects in general (e.g., our example application at the Great Barrier Reef). The main reason for this is that raw data collection allows all of the data to be captured; this data can then be analyzed in a variety of ways at a later date. As an example, scientists at AIMS are not interested in retrieving the average temperature readings at periodic intervals. Additionally, snapshot queries can always be posed on the raw data that has already been collected. Having all the data enables scientists to interpret the data in whichever way they wish at any time in the future. Other authors [Madden 2003; Chu et al. 2006] also describe similar scenarios where environmentalists prefer collecting only raw data rather than data that has been manipulated within the network using certain aggregation operators.

One of the ways to perform raw data collection is to take advantage of spatial and temporal correlations of adjacent sensors. This has been done previously in a number of research papers [Begum et al. 2004; Vuran et al. 2004; Vuran and Akyildiz 2006; Liu et al. 2007]. However, it is important to keep in mind that spatial and temporal correlations that have been identified at time t, may not necessarily hold true at time $t + x$ where $x > 0$. In fact, there could be a situation in which two neighboring nodes that usually have correlated readings

do not have correlated readings during certain hours of the day. Thus nodes should be able to adapt their operations accordingly. Though we have not discussed this issue in this article with regards to $\mathcal{DOSA}$ (since this article focuses only on the scheduling aspects), we would like to indicate that nodes that are unable to find any significant correlations between their adjacent neighbors, can autonomously opt out of the $\mathcal{DOSA}$ scheduling scheme. Thus a node that chooses to transmit correlation information, only does so if valid correlations exist. The techniques mentioned in Vuran et al. [2004], Vuran and Akyildiz [2006], and Deshpande et al. [2004] are not able to cope with sudden changes in the correlation models and fail to account for the importance of temporal fluctuations in these models. Furthermore, the approach presented in Vuran et al. [2004] and Vuran and Akyildiz [2006] is designed for event-based queries. They also assume that individual nodes are location aware. It is important to note that nodes executing $\mathcal{DOSA}$ do not need to be location aware. This definitely reduces the complexity of the software running on the nodes. Unlike $\mathcal{DOSA}$, which is designed for multihop networks, Begum et al. [2004], Liu et al. [2007], Heinzelman et al. [2002] require all nodes in the WSN to be in direct transmission range of the base station. Such a design constraint affects scalability since it prevents these solutions from being implemented in large-scale networks. While Ken [Chu et al. 2006] takes advantage of spatial and temporal correlations and works in a multihop environment, it does not mention any details regarding how to reorganize the scheduling scheme if a certain node fails or if new nodes are added to the system. $\mathcal{DOSA}$, on the other hand, is able to cope with network dynamics due to the close interaction that exists with the underlying LMAC layer. The cross-layer optimizations we perform also enable $\mathcal{DOSA}$ to operate in a more energy-efficient manner. PAQ [Tulone and Madden 2006b] takes advantage of spatial correlations between nodes to reduce transmissions. However, the cluster heads are prone to draining their energy earlier than the cluster members, since only the cluster heads are involved in periodic transmission of readings to the sink. While SAF [Tulone and Madden 2006a] improves on PAQ by ensuring that nodes send trends instead of actual sensor readings, it forms clusters off-line and thus fails to take advantage of adjacent nodes that may have correlated sensor readings. Thus all nodes that detect a change in the trend due to some sudden event, are required to transmit model updates to the sink. In $\mathcal{DOSA}$ however, only the correlating node would have to send a model update in such a scenario. Both SAF and PAQ also disregard the underlying MAC completely and are thus unable to benefit from any cross-layer optimizations. The authors of SAF and PAQ also do not provide any theoretical bounds of the energy savings that can be gained using their approach.

While there have been many MAC protocols designed for sensor networks, for example S-MAC [Ye et al. 2002], T-MAC [Dam and Langendoen 2003], and D-MAC [Lu et al. 2004], none of these protocols provide neighborhood information the way LMAC does. As shown in our results in Section 9.3, the cross-layer optimization we perform using the information presented by LMAC allows us to attain savings of up to 60%. Also, the initial assignment of LMAC slots helps in the second phase of assigning multiple $\mathcal{DOSA}$ colors to a node. The fact that LMAC is a TDMA-based MAC is an added advantage since it automatically

provides a sense of time, which is beneficial to $\mathcal{DOSA}$. While we have illustrated the operation of $\mathcal{DOSA}$ on top of LMAC, it should also be possible to run it on top of other MAC protocols. This would mean, however, that an additional as layer would have to be built that helps keep track of immediate topology information. This would naturally reduce the efficiency of the system.

11. CONCLUSION AND FUTURE WORK

The collection of raw sensor readings is of great importance to many applications in sensor networks. We have presented a distributed scheduling algorithm, $\mathcal{DOSA}$ that helps collect raw data in an energy-efficient manner by taking advantage of the spatial correlations that exist between sensor readings of adjacent nodes. Our algorithm is completely self-organizing in the sense that nodes are able to autonomously choose new schedules when there are topology changes in the network. This is possible due to the close integration of $\mathcal{DOSA}$ with the underlying MAC protocol. This cross-layer approach also results in significant energy savings. We have presented both the theoretical performance bounds and simulation results. Our simulation results indicate a reduction in message transmissions by up to 85% and an increase in network lifetime of up to 92% when compared with collecting raw data. Our algorithm is also capable of completely eliminating dropped messages due to buffer overflow, thereby improving the quality of the collected data.

We have already implemented $\mathcal{DOSA}$ on Ambient sensor nodes that use the MSP430 processor [TI 2006]. While the footprint of LMAC and the AmbientRT operating system [Ambient 2006b] comes to 2782 bytes, the footprint of $\mathcal{DOSA}$ is only 869 bytes. We have tested $\mathcal{DOSA}$ in a small network consisting of 25 nodes in an indoor environment. The convergence results gathered from the practical implementation match the theoretical results very closely. We refer the reader to Chatterjea et al. [2007] for more details regarding the practical evaluation of $\mathcal{DOSA}$. Our next step is to carry out tests in a large network consisting of about 100 nodes on the Great Barrier Reef. This environment will not only allow us to observe how the performance of $\mathcal{DOSA}$ scales, but also to test the protocol in a harsh environment where link qualities may not always be ideal.

We are currently also collecting results of the distributed data aggregation algorithm that runs on top of $\mathcal{DOSA}$. The data aggregation algorithm helps identify correlation models and keeps them sufficiently updated. It also ensures that sensors are sampled in an energy-efficient manner using a distributed protocol.

ACKNOWLEDGMENTS

We would like to thank Stuart Kininmonth from the Australian Institute of Marine Science and Ambient Systems for their help with the deployment of the sensors on the Great Barrier Reef. Olga Bondarenko helped gather the valuable data from the sensor network in Nelly Bay. The School of Marine and Tropical Biology from James Cook University supplied the data loggers for temperature collection. We would like to thank Roland Gemesi for the discussions regarding

self-stabilization. We would also like to thank the various anonymous reviewers for their useful comments which have helped improve the quality of this article.

REFERENCES

AIMS. 2006. Reef at our fingertips. http://www.aims.gov.au/pages/about/communications/waypoint/headlines-04.html.

AMBIENT. 2006a. Ambient systems. http://www.ambient-systems.net/ambient/index.htm.

AMBIENT. 2006b. AmbientRT operating system. http://www.ambient-systems.net/ambient/technology-rtos.htm.

BEGUM, S., WANG, S.-C., KRISHNAMACHARI, B., AND HELMY, A. 2004. Election: energy-efficient and low-latency scheduling technique for wireless sensor networks. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN'04)*. IEEE Computer Society, 60–67.

BONDARENKO, O., KININMONTH, S., AND KINGSFORD, M. 2007. Underwater sensor networks, oceanography and plankton assemblages. In *Proceedings of ISSNIP*. Melbourne, Australia.

BULUSU, N., ESTRIN, D., GIROD, L., AND HEIDEMANN, J. 2001. Scalable coordination for wireless sensor networks: self-conguring localization systems. In *Proceedings of the International Symposium on Communication Theory and Applications (ISCTA)*. Cumbria, UK.

BURRELL, J., BROOKE, T., AND BECKWITH, R. 2004. Vineyard computing: sensor networks in agricultural production. *IEEE Peruas. Comput. 03,* 1, 38–45.

CHATTERJEA, S., KININMONTH, S., AND HAVINGA, P. J. M. 2006. Sensor networks. *GeoConnexion 5,* 9, 20–22.

CHATTERJEA, S., NIEBERG, T., ZHANG, Y., AND HAVINGA, P. J. M. 2007. Energy-efficient data acquisition using a distributed and self-organizing scheduling algorithm for wireless sensor networks. In *Proceedings of DCOSS*. 368–385.

CHU, D., DESHPANDE, A., HELLERSTEIN, J. M., AND HONG, W. 2006. Approximate data collection in sensor networks using probabilistic models. In *Proceedings of ICDE*. 48.

COMAN, A., SANDER, J., AND NASCIMENTO, M. A. 2005. An analysis of spatio-temporal query processing in sensor networks. In *Proceedings of the 21st International Conference on Data Engineering Workshops (ICDEW'05)*. IEEE Computer Society, Washington, DC, 1190.

CRESCENZI, P. AND KANN, V. 2005a. A compendium of np optimization problems: maximum independent set. http://www.nada.kth.se/~viggo/wwwcompendium/node34.html.

CRESCENZI, P. AND KANN, V. 2005b. A compendium of np optimization problems: minimum independent dominating set. http://www.nada.kth.se/~viggo/wwwcompendium/node14.html.

DAM, T. AND LANGENDOEN, K. 2003. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*. Los Angeles, CA.

DESHPANDE, A., GUESTRIN, C., MADDEN, S., HELLERSTEIN, J. M., AND HONG, W. 2005. Model-based approximate querying in sensor networks. *VLDB J. 14,* 4, 417–443.

DESHPANDE, A., GUESTRIN, C., MADDEN, S. R., HELLERSTEIN, J. M., AND HONG, W. 2004. Model-driven data acquisition in sensor networks. In *Proceedings of the 30th VLDB Conference*. Toronto, Canada.

DIJKSTRA, E. 1974. Self-stabilizing systems in spite of distributed control. *Comm. ACM 17,* 11, 643–644.

DOLEV, S. 2000. *Self-Stabilization*. The MIT Press, Cambridge, MA.

DULMAN, S., CHATTERJEA, S., HOFFMEIJER, T., HAVINGA, P., AND HURINK., J. 2006. Architectures for wireless sensor networks. In *Embedded Systems Handbook*, R. Zurawski, Ed. CRC Press, Florida, 31–1–31–10.

EMEKCI, F., TUNA, S. E., AGRAWAL, D., AND ABBADI, A. E. 2004. Binocular: a system monitoring framework. In *Proceeedings of the 1st International Workshop on Data Management for Sensor Networks (DMSN'04)*. ACM Press, New York, NY, 5–9.

GANESAN, D., ESTRIN, D., AND HEIDEMANN, J. 2003. Dimensions: why do we need a new data handling architecture for sensor networks? *SIGCOMM Comput. Comm. Rev. 33,* 1, 143–148.

GREENSTEIN, B., RATNASAMY, S., SHENKER, S., GOVINDAN, R., AND ESTRIN, D. 2003. Difs: a distributed index for features in sensor networks. *Ad Hoc Netw. 1,* 2-3, 333–349.

HEINZELMAN, W. R., CHANDRAKASAN, A. P., AND BALAKRISHNAN, H. 2002. An application-specific protocol architecture for wireless microsensor networks. *IEEE Trans. Wirel. Comm. 1,* 4, 660–670.

HERMAN, T. 2003. Models of self-stabilization and sensor networks. In *Proceedings of IWDC*. 205–214.

INTANAGONWIWAT, C., ESTRIN, D., GOVINDAN, R., AND HEIDEMANN, J. 2002. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*. IEEE Computer Society, 457.

INTANAGONWIWAT, C., GOVINDAN, R., ESTRIN, D., HEIDEMANN, J., AND SILVA, F. 2003. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw. 11,* 1, 2–16.

LIU, C., WU, K., AND PEI, J. 2007. An energy efficient data collection framework for wireless sensor networks by exploiting spatiotemporal correlation. *IEEE Trans. Parall. Distr. Syst.*.

LU, G., KRISHNAMACHARI, B., AND RAGHAVENDRA, C. S. 2004. An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. *13*. 224a.

LYNCH, N. 1996. *Distributed Algorithms*. Morgan Kaufmann Publishers.

MADDEN, S. 2003. The design and evaluation of a query processing architecture for sensor networks. Ph.D. thesis, University of California, Berkeley.

MADDEN, S. R., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. 2005. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Datab. Syst. 30,* 1, 122–173.

MAINWARING, A., CULLER, D., POLASTRE, J., SZEWCZYK, R., AND ANDERSON, J. 2002. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*. ACM Press, New York, NY, 88–97.

MANJESHWAR, A. AND AGRAWAL, D. P. 2001. Teen: A routing protocol for enhanced efficiency in wireless sensor networks. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. *03*. 30189a.

MATLAB. 2006. MATLAB—the language of technical computing. http://www.mathworks.com/products/matlab/.

NIEBERG, T. 2006. Independent and dominating sets in wireless communication graphs. Ph.D. thesis, University of Twente, The Netherlands.

PALAZZI, C., WOODS, G., ATKINSON, I., AND KININMONTH, S. 2005. High speed over ocean radio link to great barrier reef. In *Proceedings of TENCON*. IEEE.

RATNASAMY, S., KARP, B., SHENKER, S., ESTRIN, D., GOVINDAN, R., YIN, L., AND YU, F. 2003. Data-centric storage in sensornets with ght, a geographic hash table. *Mob. Netw. Appl. 8,* 4, 427–442.

RF MONOLITHICS, I. 2007. Rfm tr1001 868.35MHZ hybrid transceiver. `http://www.rfm.com/products/data/tr1001.pdf`.

SHARAF, A., BEAVER, J., LABRINIDIS, A., AND CHRYSANTHIS, K. 2004. Balancing energy efficiency and quality of aggregate data in sensor networks. *VLDB J. 13,* 4, 384–403.

SMYTH, A. W., PEI, J.-S., AND MASRI, S. F. 2003. System identification of the Vincent Thomas suspension bridge using earthquake records. *Earthqu. Eng. Struct. Dyn. 32,* 3, 339–367.

TI. 2006. Msp430 ultra-low power microcontrollers overview from texas instruments. `http://focus.ti.com/mcu/docs/mcuprodoverview.tsp?sectionId=95&tabId=140%&familyId=342`.

TOLLE, G., POLASTRE, J., SZEWCZYK, R., CULLER, D., TURNER, N., TU, K., BURGESS, S., DAWSON, T., BUONADONNA, P., GAY, D., AND HONG, W. 2005. A macroscope in the redwoods. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys'05)*. ACM Press, New York, NY, 51–63.

TULONE, D. AND MADDEN, S. 2006a. An energy-efficient querying framework in sensor networks for detecting node similarities. In *Proceedings of the 9th ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWiM'06)*. ACM Press, New York, NY, 191–300.

TULONE, D. AND MADDEN, S. 2006b. Paq: time series forecasting for approximate query answering in sensor networks. In *Proceedings of EWSN*. 21–37.

VAN HOESEL, L. AND HAVINGA, P. 2004. A lightweight medium access protocol (IMAC) for wireless sensor networks: reducing preamble transmissions and transceiver state switches. In *Proceedings of INSS*. Tokyo, Japan.

VURAN, M. C., AKAN, B., AND AKYILDIZ, I. F. 2004. Spatio-temporal correlation: theory and applications for wireless sensor networks. *Comput. Netw. 45,* 3, 245–259.

VURAN, M. C. AND AKYILDIZ, I. F. 2006. Spatial correlation-based collaborative medium access control in wireless sensor networks. *IEEE/ACM Trans. Netw. 14,* 2, 316–329.

WEN, J. 2006. A smart indoor air quality sensor network. In *Proceedings of SPIE, Vol. 6174*. M. Tomizuka, C. Yun, and V. Giurgiutiu, Eds. 1277–1290.

YAO, Y. AND GEHRKE, J. 2002. The cougar approach to in-network query processing in sensor networks. In *SIGMOD Rec. 31,* 3, 9–18.

YE, W., HEIDEMANN, J., AND ESTRIN, D. 2002. An energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of INFOCOM*.

Received February 2007; revised September 2007; accepted December 2007