CHALMERS

Deterministic Collision Free Communication Despite Continuous Motion

Master of Science Thesis in Computer Systems and Networks

ROSA TSEGAYE AGA

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, September 2014

Master‘s thesis in Evaluation of Deterministic Collision-free Schedule for Mobile Nodes protocol

Rosa Tsegaye Aga

Examiner: Tomas Olovsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden September 2014

Abstract

Mobile ad-hoc networks (MANETS) require robust and carefully designed Media Access Control (MAC) solutions to cope with all the inherent problems of wireless communications. Today, quite a lot of solutions, that promise to address these challenges, exist; nevertheless, their practical applicability to a particular situation has not been examined in real world scenarios and hence, many of them have remained on paper and in simulations. We believe it is important to evaluate the performance of the protocol in simulators and real platform.

In this study the performance of *deterministic collision-free despite continuous motion (DCFA)* algorithm has been evaluated in simulators and on a real platform. For the evaluation, the DCFA algorithm has been implemented in tinyos-2.1.1 using the nesc programming language and a special python script to handle the topology for simulation on TOSSIM simulator; and the C++ programming language for simulation on the OMNET++ and Castalia simulators. For the evaluation of the DCFA algorithm on simulators, we have used TOSSIM, OMNET++ and the Castalia simulators. To evaluate DCFA on a real platform, we have implemented it on Crossbow's MicaZ platform motes with a CC2420 radio. The performance of the DCFA algorithm has been measured in terms of throughput and success-rate. Since the evaluation of the DCFA algorithm is on mobile nodes, we have considered two mobility models to evaluate the performance of the DCFA algorithm. Two mobility models have been designed in this study in order to challenge the DCFA algorithm and evaluate it. When the nodes move in parallel and reach the boarder of the grid, the first mobility model lets the node move to the next line of the the grid and continue in the opposite direction; the other mobility model lets the node go back to the beginning where it started the mobility and continues in same direction. From the study, we have observed that the DCFA algorithm does not have self stabilisation when unexpected things happen in the network.

Acknowledgements

First of all, I would like to say the following to my supervisor Dr. Michael Elad Schiller. For all your valuable advice, patience, constant encouragement, the immense knowledge shared and appreciation throughout our work, I am eternally grateful. I would like to express my tremendous gratitude to Professor Tomas Olovsson for being available, providing useful feedback on my work and for the time spent reviewing drafts of my dissertation. Your comments have contributed a great deal to make the final document what it is today.

I am specially thankful to Mustafa Muhammed who has helped me with all the difficulties and technical details I needed, more like a partner than just a friend. Mustafa, you have enriched my time here at Gulliver project in a way that words cannot express and sharing us your valuable knowledge and experience.

Above all, I would really like to thank Yonas Tsegaye for being with me starting from the beginning till the end of this project with all support. I just can not thank you enough. I would like to dedicate a line to Abdi Abate who has been a part of this project at its early stage, and helped us to have a base of this work. And also sincere appreciation and thanks to all other members of the Gulliver project.

Final and most of all, I would like to express my heart-felt gratitude to my mother, Genet Seyoum. Her inspiration, guidance and unconditional love have been my greatest strength. Last but not least, I would like to thank my youngest brother Henok Tsegaye and sister Feven Tsegaye. They were always supporting me and encouraging me with their best wishes. I also have to give a special mention and appreciation for the support given by my uncle Endale Geberu. Thank you ALL my good people, I would probably not be where I am today without your inputs, guidance, encouragement and friendship.

Rosa Tsegaye Aga, Gothenburg, Sweden 01/09/14

Contents

1

Introduction

1.1 MANETs

MANETs are networks of self-controlling and configuring mobile devices, referred to as "nodes". Such nodes have no centralized aid and are responsible for scheduling themselves, creating and maintaining their network. MANET applications range from their early conception in military networks to more recent and modern applications such as wildlife tracking, intelligent transport systems and vehicular ad hoc networks, VANETs. Some of the key characteristics of general MANETs are energy efficiency, scalability, throughput, adaptability to changes and self stablilzation. It still remains a challenge to design such an ideal MAC protocol with all these desirable features, and practically one is achieved on the expense of the others. There is no standard MAC protocol that is best in all aspects [1] . Some protocols such as DMAC [2] are designed with the intent of reducing latency where-as others like TreeMAC [3] focus on improving throughput as well as energy consumption.

The implementation of MAC protocols can broadly be classified as contention based, as in the traditional IEEE 802.11 [4] specification that use CSMA, or a contention free like FDMA, CDMA, TDMA. Protocols that use a combination of the two also exit, like TRAM [5] that uses a combination of TDMA and CSMA.

This thesis is based on the paper entitled *'Deterministic Collision-free algorithm despite continuous motion'* [6]. In there work, they proposed a deterministic collision-free MAC solution applicable for general Mobile Ad-hoc Networks (MANETS) with a special recommendation for Vehicular Ad-Hoc networks (VANETS).

In this thesis, the DCFA algorithm [6] implementation is the combination of space division multiplexing (SDM) and time division multiplexing (TDM). As per [6] study, the goal of the DCFA algorithm is to have a collision free communication protocol for continuously mobile nodes. Hence, as claimed by the authors in [6], this type of protocol is an ideal choice for safety critical environments such as VANETs.

1.2 Objective

The main objective of this thesis is to evaluate the DCFA algorithm performance by simulation on simulators and platform. The performance has been measured in terms of throughput of the protocol and its success-rate with respect to the number of mobile nodes which have exchanged neighborhood information messages successfully.

This thesis objective is not limited to just only the evaluation of the performance of the algorithm. After meeting the above objective, the protocol has criticized. Protocol limitations have been pointed out.

1.3 Research Questions

These research questions help for further understanding the performance factors and efficiency of the DCFA algorithm. This study answers the following questions:

1. Is the DCFA algorithm a collision free communication protocol for continuously mobile nodes under all circumstances?
2. Does the DCFA algorithm fulfill the main MAC protocol properties to be called or categorized as a MAC solution applicable for general MANETs?

1.4 Hypothesis

In order to address these questions, the DCFA algorithm was implemented in TOSSIM, OMNET++ and the Castalia simulators and the micaZ platform. The nodes then have to exchange neighborhood information messages on the implemented simulators and platform without any collisions. The implementation of the algorithm on the platform has been done by other groups of the project. However the result from the platform has been evaluated in this study with the simulators result.

On the DCFA algorithm implementation, the nodes are located on a two-dimensional plane. For the SDM part of the DCFA algorithm solution, the plane is tiled by hexagons similar to Viquar et. al [6] study. However, in a study performed by Viquar et. al [6], they did not mention that there is a problem if the two-dimensional plane of the DCFA algorithm implementation is tiled in a different way other than with hexagons. Therefore, we have tiled the two-dimensional DCFA algorithm implementation plane with squares instead of the hexagons. The square tiled plane, called grid plane, has to have different colors in each square. If two squares have the same color, they have to be located far a part by using the given parameter in the DCFA algorithm to don't have interference in each others broadcast. The mobile nodes' broadcast slots are dynamically allocated depending on the tile color. The tile color and the interference will be explained in detail in the next chapters. Despite the plane tile, we have been following the same parameters and methods of the DCFA algorithm for the implementation in the simulators and in a real platform. The evaluation of the DCFA algorithm has focused on its performance; specifically on its throughput and success rate with respect to the number of mobile nodes which have exchanged neighborhood information successfully. To evaluate the DCFA algorithm, two mobility models have been designed and tested on the simulations. The first mobility model represents the real car mobility. It is used to see if the algorithm works like expected. The second mobility model has been designed to challenge the algorithm in an unexpected scenario and to see if it is still works like expected. Therefore the DCFA algorithm has been tested with the two mobility models, and its performance has been measured on both the simulators and the real platform.

1.5 Assumptions

The underling assumption of the evaluation of the DCFA algorithm's performance is that it is 100% successful when exchanging neighborhood messages under all circumstances.

To do the implementation and evaluation of the DCFA algorithm, there are many good simulators and platforms available. Between the simulators TOSSIM and OMNET++ which have used in this study, Castalia is the one which has been selected for further investigation of the DCFA algorithm evaluation. However, because of the shortage of enough documentation, it was a bit hard to collect the DCFA algorithm evaluation result from the Castalia simulator implementation.

Castalia has a GUI for monitoring the simulation flows, uses C++ development and visualization [7]. It has available models for MAC protocols, like TMAC, SMAC, Tunable MAC and it works on the platforms of OMNET++. However in this study, we has chosen TOSSIM and OMNET++ to implement DCFA algorithm. Even if Castalia has many interesting features, its in the state of active development and it has not made lager impact so far.

Protocols which are related with the DCFA protocol like TDMA and others, have many common features, drawbacks and good points. For this study, we choose the TDMA protocol properties as a baseline for our DCFA algorithm evaluation. We have assumed that TDMA is the best choice to evaluate and compare the DCFA algorithm with. One of the main reasons for selecting the TDMA protocol from other protocols for comparison is that TDMA is one very popular scheduling algorithm. Additionally, the DCFA algorithm is designed from some of the TDMA protocol features. For example the DCFA algorithm has deterministic scheduling for nodes in a MANET to avoid collisions by allowing nodes to maintain information about their neighboring nodes. This node's scheduling methods of the DCFA algorithm has been taken from the TDMA protocol features. This feature ensures that nodes in different colored squares (tiles) never broadcast at the same time.

1.6 Structure of the thesis

The rest of the thesis is organized as follows. In Chapter 2 related work is described to put our work in context. Then in Chapter 3 the algorithm description is presented. In Chapter 4 the details of the implementation in simulators is presented. Chapter 5 contains a detailed explanation of the DCFA implementation in MicaZ platform. Chapter 6 presents the detailed simulators results of the work. The discussion and the future work are described in Chapter 7. Finally, Chapter 8 presents the conclusion.

2

Literature Review

2.1 Introduction

Mobile ad hoc networks (MANETs) represent complex distributed systems that comprise of wireless mobile nodes which can freely and dynamically self-organize into arbitrary and temporary network topologies [5]. MANETs have some challenges for MAC protocol design such as mobility, radio link and so on, therefore MANETs have to have efficient and distributed MAC protocols and it is necessary to evaluate MAC protocols and see if they can give the right functionally for MANETs.

Many researches have been evaluating wireless sensor networks algorithms using different simulators, testbeds or using their own framework. In the past, some researches were criticizing simulators since they could lead to incorrect conclusions. From this motivation many researchers have been doing much research work on comparing simulators and testbeds for simulation and also criticising them. At the same time, some researchers developed frameworks to evaluate algorithms, like *'Towards Reliable Communication and Agreement in Mobile Ad-hoc Networks: Algorithms, Simulation and Testbed'* [8] research.

2.2 MAC protocols in MANETs

As mentioned in the introduction of this chapter, MANETs have some challenges to design MAC protocols. However, MAC protocols have to fulfill at least the basic characteristics of MANETs. Researchers from university of Florida [5] studied challenges that MAC design is facing and how they impact the performance of MANETs.

Some of the MAC protocol design challenges are: there is no centralized controlling system; due to hardware constraints, a node can not immediately detect collisions during the transmission which leads to channel inefficiency; the network topology may change from time to time since the nodes are in mobile, then each node may experience different degree of channel contention and collision. Other important MAC protocol issues are energy efficiency, fairness and quality of service provision [5]. During the design of MAC protocols for MANETs, these challenges need to be considered carefully.

The MAC protocol specifies how nodes in a sensor network access a shared communication channel [9]. The desired properties of MAC protocols are [9]: it should be distributed and contention-free; it should self-stabilize for changes in the network and these changes should be contained; it should not assume that nodes have a global time reference.

Many researches have been studying properties and challenges of the MAC protocols. In addition, some researches particularity highlight which properties of MAC protocols have to be fulfilled during MAC protocol design. These research studies have helped us to evaluate the coming new MAC protocols and determine whether they are efficient protocols for MANETs or not. Our study has also followed these research direction when evaluating the DCFA algorithm.

2.3 Simulations in MANETs

Even if simulation is useful for evaluating protocol performance, several recent studies underlined the lack of stability of the applications that threatens the credibility of the published claims. However some researchers were arguing against the idea of the lack of simulations on the stability of the applications. Ivan Stojmenovic [10] is one researcher who advocates for a different overall view on his *simulations in wireless sensor and ad-hoc networks: matching and advancing models, metrics, and solutions* article; even if he while he was agreeing with some of the criticism on simulation. As Ivan Stojmenovic mentioned in his study, the primary goal of simulation is to provide sufficient support for new concepts and protocols, for an overall combined contribution. He has stated that Proof of concept is basic (not thorough or testbed based) simulation using assumptions in designed protocol, including comparison with truly competing existing solutions. Ivan Stojmenovic concentrated on simulations as an independent task, and advised to have better reporting (repeatability), more reliability, more realism, and more scenarios in simulation. However in their study, they discussed the more general view of simulation as a support for new ideas and theories, providing a platform for their comparison with truly competing existing solutions, not for their validation. Then they support the use of models, matching assumptions and metrics in the problem statement and simulation to provide a basic ”proof of concept” and comparison with truly competing solutions.

As a conclusion, Ivan Stojmenovic et.al [10] recommend carrying out a proper and effective simulation activity for protocol design and evaluation. Having this study in mind as a support, we moved forward on our study to evaluate the DCFA algorithm by simulating on both simulators and on real platforms.

2.4 Simulators comparison for MANETs

Since simulators have differences between them, it would be good idea to study their behavior and impact on the algorithms used for simulation; instead of picking one randomly. Some studies compare simulators to choose the best one for the given task by studying their behavior. By taking this point in to consideration, we have been comparing and evaluating different simulators for our DCFA algorithm simulation.

On simulator selection, the NS-2 simulator has taken our attention to implement the DCFA algorithm. However, Nicolas Eude, Bertrand Ducourthial and Mohamed Shawky research work and findings using this simulator made us change our mind [11]. They used both experimentation and simulations to develop efficient algorithms for C2C (car to car) communication using the NS-2 simulator. However the propagation models of the NS-2 simulator was not sufficient on preciseness. Therefore they extended their study by adding a new functionality to NS-2 in order to achieve the required precision on car-to-car communication experiments. From their study, it's possible to see that the NS-2 simulator is not sufficient for preciseness for high mobility ad hoc networks. The NS-2 simulator needs additional functionality to have such preciseness in the simulation experiments.

TOSSIM is one of the simulators which has been studied and used in many researches. The TOSSIM simulator is a network emulator and can support thousands of nodes. TOSSIM emulates hardware componenets like clock, radio, potentiometer etc. It can simulate the real world situation more accurately [12] than other simulators like NS-2. One of the main point of it's accuracy, unlike NS-2, is that it has the functionality to have preciseness in the simulation experiments. This emulator may provide high speed simulation result at component levels because of its behaviour of compiling directly to native code [12]. However, TOSSIM has also some limitations. One of its limitation as presented in [12] study, is that it is designed to simulate behaviors and run applications only on TinyOS. Comparing TOSSIM simulator with the NS-2 simulator and others, it is a good simulator; especially when it comes to wireless and high mobility in ad-hoc networks.

Cavin [8] believed that simulations have some shortcomings. In order to avoid the shortcomings of simulations, he proposed to have an adequate software framework to ease the development and the deployment of applications and algorithms. He showed in his study that evaluation of an algorithm by simulation leads to incorrect conclusions. Then he has developed FRANC, a high weight Java framework dedicated to the implementation of real ad-hoc applications.

As a conclusion, some simulators are good and some are not good depending on the given problems and the algorithms. In our study, we found that the TOSSIM simulator is much a better simulator to simulate the DCAF algorithm. In addition, we have been using the OMNET++ simulator as well for more findings. A detailed explanation of the OMNET++ simulator is presented in implementation chapter.

2.5 Summary

First of all, the first two papers [5] and [9] showed the MAP protocol challenges and it's desired properties which an efficient MAC protocol needs to take into account during design. This helped us to evaluate DCAF's algorithm properties based on these MAC protocol properties. The second paper [10] advised to carry on a proper and effective simulation activity for protocol design which shows the goodness of simulation. This study helped us to go on for our simulation to evaluate the DCFA algorithm. The rest of the papers are comparing and showing the behavior of simulators on algorithms and also compare simulators with real environments. This helped us to choose the proper simulator to evaluate DCFA algorithm and gave us a clue about that it might not be enough to evaluate algorithms using only one simulator to tell the performance and efficiency of the protocol.

3

Description of the algorithm

3.1 Deterministic collision free despite continuous motion algorithm

The DCFA algorithm communication plane is developed on the hexagon network topology model. This communication plane is partitioned by hexagons as shown in figure 3.1. Each hexagonal region is called tile and each tile contains a bounded number of nodes. Tiles are grouped to form a super tile; the larger communication domain consisting of different colored tiles. The numbers 0-6 in figure 3.1 correspond to different colors of the tiles. Colors in a super tile are unique but they are repeated in each super tile. The transmissions of tiles with different colors are scheduled separately by TDMA; whereas all tiles which have similar color broadcast simultaneously. As shown in the figure 3.1, there are tiles with number 0 which have shaded color; such tile nodes in different super tiles can be able to transmit the neighbor information simultaneously as long as the distance between them is greater than the minimum required threshold at any moment of time. The information allows nodes to maintain information about their local neighborhood. This information in turn is used to keep the schedule collision-free. In our implementation, between 80 and 850 nodes have been used in the topology to run different experiments where we have studied the performance of the algorithm.

When the DCFA algorithm was developed, it was not mentioned why the hexagon network topology model was chosen as a communication plane. This situation opens the door for this thesis to choose any topology model which has a good performance on networks. Therefore, we have been using grid topology as a communication plane for the DCFA algorithm implementation. Different network topologies have different effects on the properties of the network [13], such as the reliability, energy consumption and latency of the network. From the list of the network topologies, grid topology has the best reliability comparing with the regular hexagon topology and the equilateral triangle topology models [13]. However, the definitions and properties, which are used

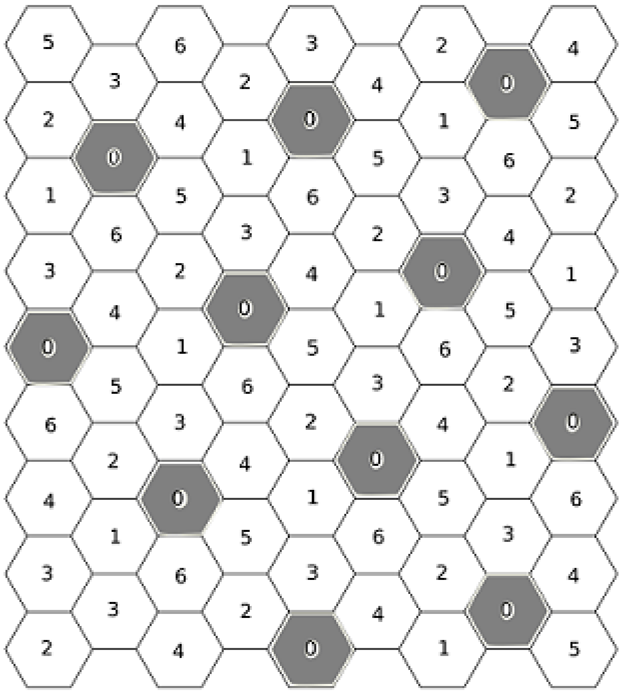

Figure 3.1: Partitioning of a plane in to hexagons

in the hexagon topology for the DCFA algorithm, are still used and applied in the grid topology in our study; like tiles, round, colors and the like. The topology change, which was made in this thesis, does not change the behaviour, requirements and parameters of the DCFA algorithm. The grid network topology is shown in figure 3.2.

The time allocated for a single node is called a broadcast slot. The group of broadcast slots denoted by U in a given tile is called a round. If there are M tiles in a super tile then MU broadcast slots are required for scheduling. It is assumed that there is an upper bound on the number of nodes that can exist in a given tile at any instant of time; this bound is denoted by V< U. Hence, V slots of a tile are used by the nodes; whereas the remaining U-V slots are used for the other protocol, see figure 3.3.

For scheduling, the orders of the rounds have been allocated on different colored tiles per super tile. Two scheduling models were used for the DCFA protocol by considering directional bias. The second scheduling model does not experience directional bias like the first scheduling model. To get a good understanding of these two scheduling models, they are explained in the next paragraphs.

The first model has information propagation in one direction. In this model, the node's time slot is allocated from right to left and top to bottom for each phase in a super tile. This model has directional bias; this means it favors the propagation of information in one particular direction. Consider the horizontal path b shown in Figure 3.4 In this scheduling each color is allocated exactly one round during one phase and the

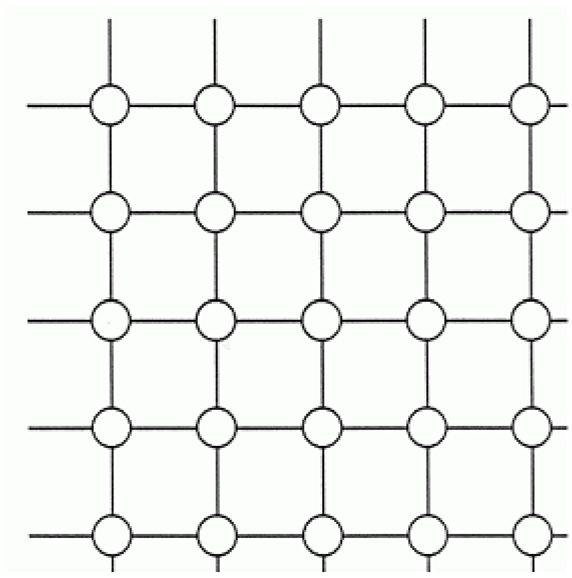

Figure 3.2: Grid topology

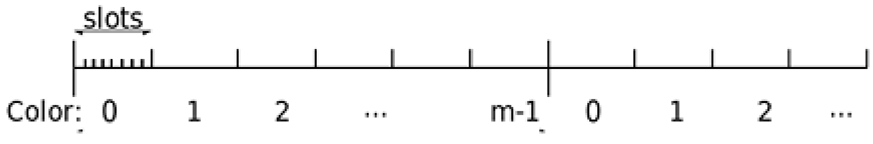

Figure 3.3: TDMA slot of M tiles with color from 0 to M-1

sequence of colors is the same for every phase [6].

The second model, the node's time slot allocation is done in a concentric circle in a super tile. As shown in figure 3.4 *c: multiple paths in different direction*, the information propagation schedule should propagate along all the six paths from the center equally [6]. Therefore the second model starts with allocating the slots from the center of the super tile to its boundary. Then the information propagates from the boundary of the super tile to the center. When this information propagation is done in the first phase, the next phase propagates information in the opposite direction by allocating the slots starting from the boundary to the center of the super tile. The periodicity of this schedule is shown in the figure 3.5. This model does not show any directional bias, all nodes get fair scheduling to propagate information in each phase.

In this thesis, the second scheduling model idea has been used for the implementation of the DCFA algorithm. Our study has not been following the exact steps of the second scheduling model to allocate slots. Mainly, we have been considering the idea of not

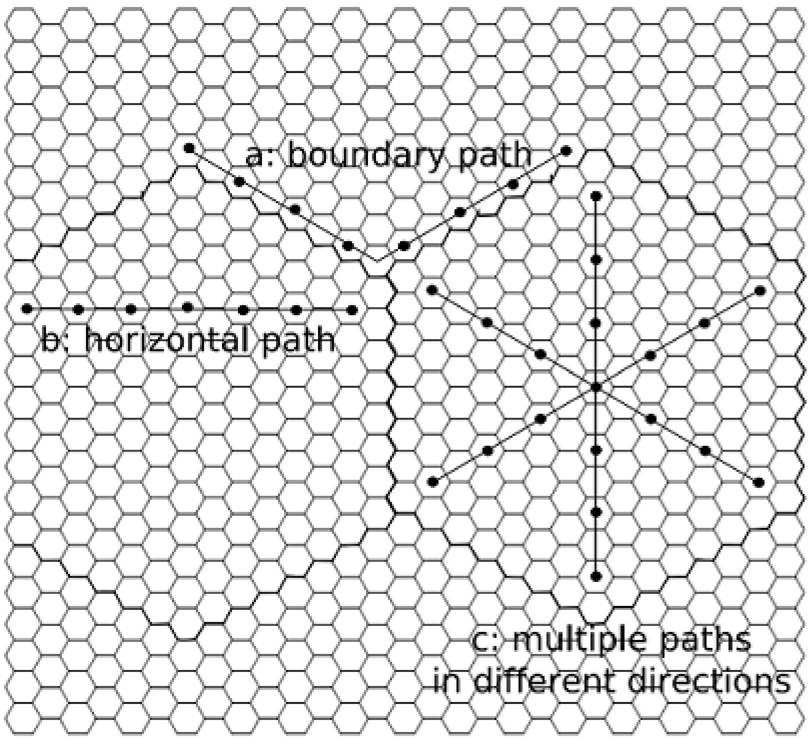

Figure 3.4: Information flow on all paths without directional bias

having directional bias by allocating slots starting from different directions in every phase instead of following only one direction. Since we have been using grid topology, the slots allocation are not exactly from the center to the boundary and from the boundary to the center just like the second scheduling model. However, we have been handling directional bias by allocating slots from the right to the left and from the left to the right direction in every phase.

3.2 Statistical Analysis

In the statistical analysis section, the success rate assumption and the algorithm's constraints which have explained in the [6] paper will been seen.

When a node successfully transmits a neighborhood message, all nodes within the nodes broadcast radius receive the message. However, logging all such received messages requires large amount of memory in the order of O(pq); where p is the number of senders and each senders send q messages. To avoid the problem of having large amount of memory, we have only considered to log the information of the ratio of the receivers for the propagated neighborhood information from sender. The ratio of the number of receivers to the sender is the success rate.

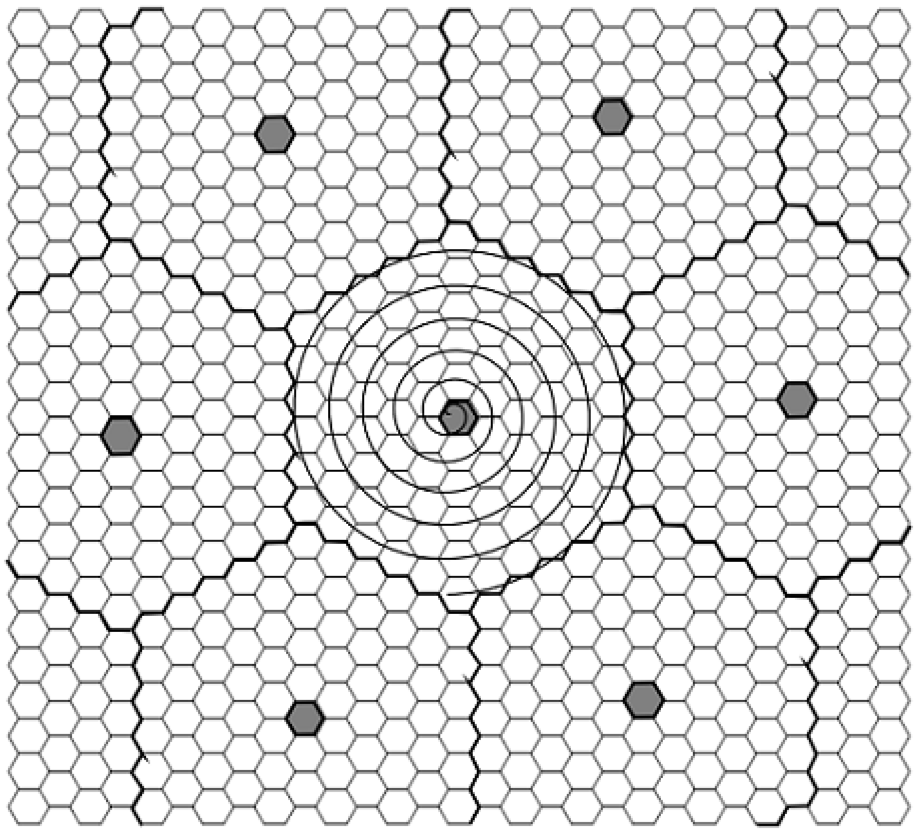
Figure 3.5: An example of the network division in to regions

The algorithm assumes two constraints on the mobility of the nodes:

- Constraint 1- In figure 3.6, there are two nodes *n1* and *m1* in the two far ends of the two neighboring tiles. At the beginning of the phase, they move away from each other by a distance $mu\sigma$. In the phase, *mu* stands for the total broadcast slot for one super tile and σ represents a fixed speed. They should still receive each other's broadcast throughout the phase based on the equation 3.1.

$$\rho + 2mu\sigma <= R \tag{3.1}$$

  The maintenance of this neighborhood knowledge is a part of [6] proposed solution and is interleaved with the collision-free schedule. This constraint guarantees the maintenance of the neighborhood knowledge with the assumptions that every nodes' broadcast is heard by all others within their broadcast radius *R*.

- Constraint 2- *n1* and *m1* are two concurrently transmitting nodes moving towards each other, as shown in the figure 3.7. The distance between these two nodes should

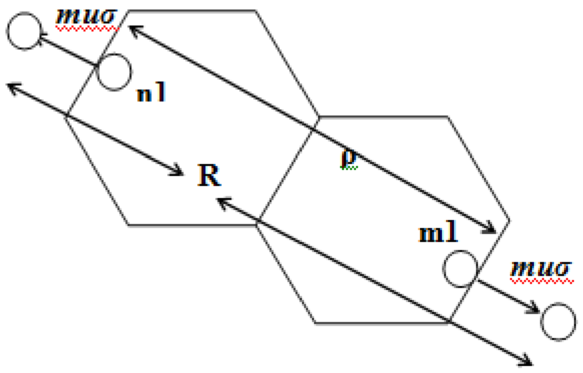

Figure 3.6: Two nodes *n1* and *m1* moving away from each other from the far ends of two adjacent tiles in a super tile

still be greater than the minimum threshold throughout the phase. Therefore their transmissions do not interfere. It's shown in equation 3.2.

$$\lambda - 2mu\sigma >= R + R' \tag{3.2}$$

R stands for broadcast radius and *R'* for interference radius which is roughly 2.2*R. This constraint ensures collision avoidance.

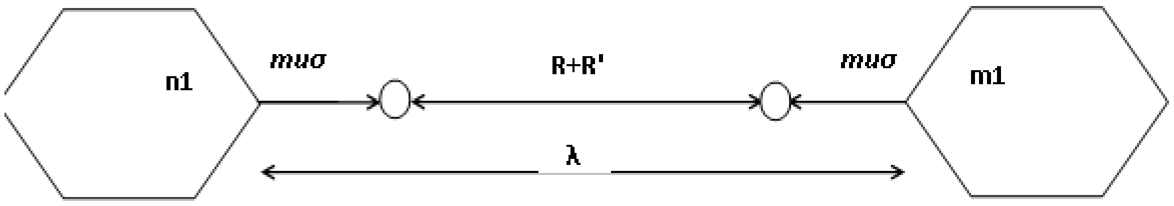

Figure 3.7: Two nodes *n1* and *m1* moving toward each other from the far ends of two adjacent tiles in a super tile

4

Implementation

4.1 Introduction

In our study, the implementation of the DCFA algorithm was done using the TOSSIM and OMNET++ simulators, and the micaZ platform implementation from another group of students. From the implementation, the performance and efficiency of the DCFA algorithm was studied.

The implementation started on the TOSSIM simulator with the TinyOS applications using the NesC programming language with the help of python to run the simulation and to setup the topology. In addition to the TOSSIM simulator, the OMNET++ simulator was used as another simulator to study the performance of the DCFA algorithm. The implementation of the DCFA algorithm was done on the OMNET++ simulator using the C++ programming language. Unlike the TOSSIM simulator, the OMNET++ simulator simulate the simulation process on graphical user interfaces which are highly useful for demonstration and debugging purposes. The implementation has continued on the micaZ platform by another group of students.

Before we did our implementation, the DCFA algorithm's parameters was studied and identified to use the correct values of parameters in the DCFA algorithm instead of using the default values from the simulator.

The implementation has been done on the two mobility models by letting the nodes follow these models during their mobility. The DCFA algorithm was then evaluated based on the result which has been collected from the two mobility models implementation. As explained in the previous chapters, the main reason of using the second mobility model is to evaluate the DCFA algorithm response in unexpected environments; especially to evaluate its self stabilization property if something unexpected happens in the network.

4.2 Tools

TinyOS [14] [15] is a light weight operating system designed for the domain of wireless sensor networks. The core OS takes a few hundred bytes of memory space which can be easily compiled into the motes memory. TinyOS applications make use of components which are software abstractions of the underlying hardware functionalities. Components either provide or use an an interface and interfaces contain commands that the application calls and events which are hardware interrupts that the caller must handle. There are two types of components called modules and configurations. Configuration contains declaration and wiring of components while modules do the actual implementation of the code written in the nesC[16] programming language. NesC is a variant of the C programming language with libraries supporting components based programming.

TinyOS uses TOSSIM as a library and it is one of the most widely used wireless sensor network simulators available. As described by the developers in[17], TOSSIM is a discrete event-driven simulator where all simulation events associated to a mote are timestamped and kept in a separate event queue in a global time order, and executed one by one.

In tinyOS, a call to a certain time taking operation such as sending a message, returns immediately; on compile time, the component that provides the service notifies the caller through a signal. Such an operation mode where the command and the signal are decoupled is called split-phase operation [17]. TinyOS splits the invocation and the compilation in two different phases. Commands and events are the main driving factors of an execution of a tinyOS application. TinyOS also uses operation tasks that could be deferred for later execution. In the TinyOS, tasks run at the background and can not be preempted by other tasks but by hardware event handler. The TinyOS commands or events can post a task whenever needed and return immediately which make them more responsive [17].

TOSSIM fulfills most of the required properties of a tinyOS simulator such as scalability, completeness, fidelity and bridging [17]. It can simulate a network of thousands of motes, with the current practical experience, in this study we have used up to 850 nodes, and also it captures the behavior of the network at high fidelity.

The command line execution of the TOSSIM has a debugging facility with which users can print debugging messages on the screen and use it in more suitable way. There is a recent tinyviz plug in, a Java based graphical user interface for enhanced user control and feedback. The original implementation of the TOSSIM does not model thread based execution and also lacks support for modeling the CPU time/energy [17], but there is a later extension called powerTOSSIM for the modeling power consumption [18].

Python is one programming interface which is supported by the TOSSIM library. TOSSIM does not run simulations by itself; it rather relies on a special python or C++ script that configures the network. It defines the underlying radio behavior and also runs the simulation. Such a script creates a set of properties to simulate the propagation strength between links. In this thesis, the DCFA algorithm implementation uses a python script to run the TOSSIM simulation.

The python script is responsible for creating the original topology and keeping track of the node's location when they move. Additionally it defines the radio properties and also controls the simulation. A special Java application is used to create the propagation strength between the nodes. Whenever the nodes change location, the application takes the new locations as a parameter then outputs the corresponding gain between every pair of locations. Additionally, this application sets a predefined noise floor value from a configuration file to each created motes. There is also a python radio object that can be created to define the gains between a pair of coordinates. But this radio object is less efficient; and in this study, we have decided to manually define the locations and the corresponding gains. The radio object is also used to monitor the status of the link between the two motes at any time in during the simulation. These include checking the existence of a link between a pair of nodes, and getting the corresponding gain between a pair of locations and the like.

The other simulator which has been used for the implementation is OMNET++. OMNET++ [19] is an object-oriented modular discrete event network simulation framework. It has a generic architecture, so it can be (and has been) used in various problem domains. It is useful for modeling and simulation of any system where the discrete event approach is suitable, and can be conveniently mapped into entities communicating by exchanging messages [19].

In OMNET++, modules are programmed using C++. These modules can be connected with each other via gates and combined to form compound modules. Modules communicate through messages passing along predefined paths via gates and connections, or directly to their destination [19]; the latter is useful for wireless simulations.

4.3 Simulators

4.3.1 Implementation on TOSSIM simulator

The DCFA algorithm implementation is a pure TDMA algorithm that completely ignores the default Carrier Sense Multiple Access (CSMA) properties of the TOSSIM simulator. Hence in this thesis, the DCFA algorithm implementation has to disable the default mac properties of the TOSSIM simulator in the TOSSIM mac object. This has been done through the use of the mac object or more efficiently using a tinyOS component that provides these services.

During the experiment, the initial topology creation was handled by python for the selected number of nodes. Once the nodes are placed in the topology and they get their initial coordinate, they know their color (tile) with the help of the function which is implemented in python. The nodes have mobility in every round; this means, when a node start a broadcasting message in a new round the nodes have moved since their previous location.

For simulating node mobility, the two mobility models described earlier have been used. The first mobility model represents real world vehicle movement with parallel mobility in different directions. Unlike the first model, the second mobility model does not exist in the real world vehicle movement. Since self stabilization is one of the main characteristics that MAC protocols have to have and shows their efficiency in MANETs, the second mobility model was chosen as a challenge. These mobility models are further explained in sub section *4.3.2*.

The DCFA algorithm does not handle initial discovery of the neighbors. It assumes that initially all nodes know all other nodes within their broadcast radius *R*. During the implementation of the DCFA algorithm in this thesis, the introduction phase has been denoted by phase -1; and in this phase all nodes broadcast their information consecutively based on their ID. Therefore, in the beginning of the phase 0, all nodes know all other nodes in their broadcast radius.

The nodes' slots have been allocated at the beginning of every phase based on the rank and the color of the nodes. Then nodes wait for their time-slot to broadcast the packet. Note that nodes may have the same time-slot but in different super tiles. The nodes with the same time-slot broadcast simultaneously without collision; since they are in different super tiles. Each node calculates their slot using formula 4.1; where color is the color of hexagon (in our case the square of a grid) and rank is identifier of the nodes from U slots in its round and it starts from one. The pseudo code of the algorithm is available in the original paper [6].

$$color + rank - 1 \tag{4.1}$$

In the DCFA algorithm, time synchronization on the real platform is handled by the token circulation which will be explained in explanation of MicaZ platform chapter. On a real platform implementation, a simplified way of centralized base station and sub-base stations that serve as a relay between the nodes and the base stations was used. However, in TOSSIM a global time has been assumed.

4.3.2 Mobility Models

Two mobility models have been used for nodes to implement movement in the network while they are propagating message in every round.

The first model represents real world node movement. It's used to see whether the algorithm is giving the expected result while the nodes are in movement. The nodes in this model move in parallel but in opposite directions. When nodes reach the border of the network, they turn to the other lane of the road and continue their movement. This movement has been repeated in every round. As figure 4.1 shows as an example, there are two parallel lanes and nodes move in opposite direction on the lanes. At the beginning and ending of the lane, there are arrows to show the direction of the next line of the road When the nodes reach the end of the lane, they move to the next lane of the road and continue the mobility. In figure 4.1, the nodes color does not represent anything.

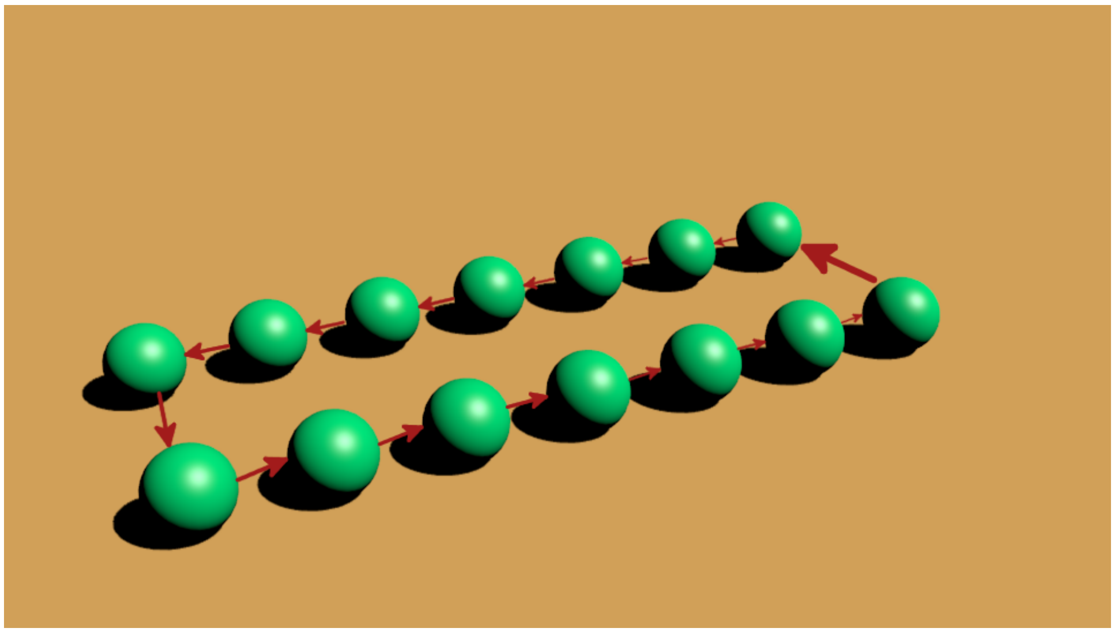
Figure 4.1: The first mobility model

In python, this model is handled as follows:

```
def move():

#This function handles the movement of nodes in every round;
#those nodes which are on y=0.375 and y=1.875 positions
#move in opposite directions in each other.

ftopo = open(topo_file , "w")
#open topology file for writing the new node position

for i in range(0,NODE):
    if(n[i].y==1.5): #nodes on y=1.5 position
       n[i].x = n[i].x + speed1
      if(n[i].x >=(Dimension*UnitDistance)):
         n[i].x = (Dimension*UnitDistance)-(n[i].x ..
                          \%(Dimension*UnitDistance))
         n[i].y=4.5
    elif(n[i].y==4.5):    #those on y=4.5
         n[i].x = n[i].x-speed2
         if(n[i].x < 0):
         #node come back on y=0.375
```

```
            n[i].x=-(n[i].x)
            n[i].y=1.5

n[i].color = findColor(n[i].x,n[i].y)
c[i]= str(n[i].id) + "    " + str('\%.3f'\%(n[i].x)) + "    " ..
            + str(n[i].y) + "    " + str(n[i].color) + "\n"
ftopo.write(c[i])
ftopo.close()
```

When the coordinate of the nodes changes, it is saved in the opened topology file and the link gain has been modified based on the new node location.

```
os.popen("java LinkLayerModel config.txt " + layout_file ..
     + "  " + topo_file)
```

After the DCFA algorithm's performance was studied on the first model, we continued to investigate how it handles some unexpected situations using the second model has proposed. This second model does not exist in the real world and does not represent real car mobility. Here when nodes reach the border of the network, they jump to the other side of the border and continue their movement. As figure 4.2 shows, nodes in parallel lanes move in different direction; when nodes reach the end of the lane, they jump to the beginning of the lane as the arrows show in the figure at the beginning and ending of the lanes. Just like figure 4.1, figure 4.2 nodes' color does not have special meaning or representation in the figure for the model.

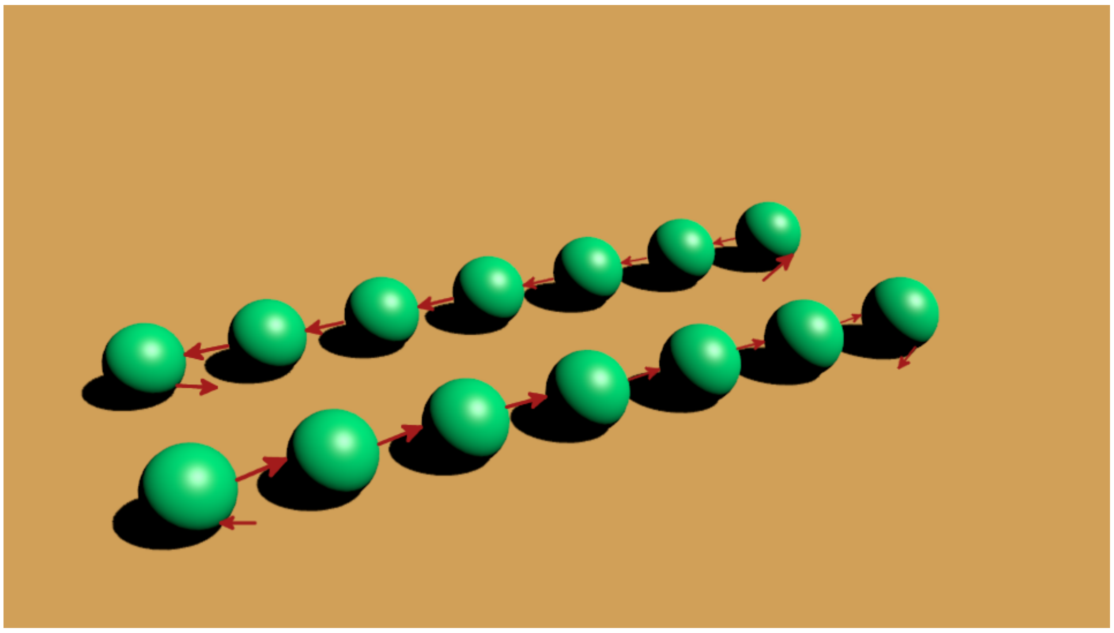

Figure 4.2: The second mobility model

The second mobility model is handled in python in the same way as in the first model, except some modification on *'move'* function. The modification of the *'move'* function and the node turning statement. The code is as follows:

```
def move():

# Almost all statement of the function is the same with
# the above first model code. The only difference is on
# the turning point.

......

ftopo = open(topo_file, "w")
    #open topology file for writing the new node position

for i in range(0,NODE):
    if(n[i].y==1.5):       #nodes on y=1.5 position
       n[i].x = n[i].x + speed1
      if(n[i].x >=(Dimension*UnitDistance)):
         n[i].x = (Dimension*UnitDistance))

    elif(n[i].y==4.5):    #those on y=4.5
         n[i].x = n[i].x-speed2
         if(n[i].x < 0):
         n[i].x = 0

n[i].color = findColor(n[i].x,n[i].y)
c[i]= str(n[i].id) + "   " + str('\%.3f'\%(n[i].x)) + "    " ..
         + str(n[i].y) + "   " + str(n[i].color) + "\n"
ftopo.write(c[i])

.....
```

4.3.3 Implementation on OMNET++ simulator

The first implementation of the DCFA algorithm was done on the TOSSIM simulator. For more accuracy, another implementation was done using the OMNET++ simulator. However the result from the OMNET++ simulator was not taken into consideration because of the simulator's signal-to-noise ratio (SNR) behavior. The implementation was done without any modification of the parameters and formats from the TOSSIM simulator implementation. For example, the topology model and scheduling model are the same. And the DCFA algorithm implementation has to allow an interference in the simulation environment to see the performance. Even if the OMNET++ simulator does not have a signal-to-interference-plus-noise ratio (SINR) interference model, we have implemented the interference environment by C++. This interference environment has been implemented by giving same time slot for each node which are found in different super tiles. TThough, it would be good to have real interference in the simulation environment for clear and precise evaluation.

The implementation of the DCFA algorithm on OMNET++ started by building modules using C++. An OMNET++ model is built from components (modules) which communicate by exchanging messages. By nesting modules, compound modules can be formed; that is, several modules grouped together. When a module is created, the system (each node) maps it into a hierarchy of communicating modules. As shown in below, the system keeps track of data rate and delay as well; *N* followed by a number represents the system (node), the arrow represents the connection between the systems, and the *out* and *in* arrays represent gates (ports).

```
connections:
 N0.out[0]-->{delay=0.1ms;  datarate=24bps;@display("ls=black,0");
    }-->N1.in[0];
 N0.out[1]-->{ delay=0.1ms;  datarate=24bps;@display("ls=black,0");
    }-->N2.in[0];
 N0.out[2]-->{delay=0.1ms;  datarate=24bps;@display("ls=black,0");
    }-->N3.in[0];
    ...
```

The modules structure was defined in the NED files using NED language. The NED files can be edited in a text editor or in the graphical editor in the Eclipse-based OMNeT++ Simulation IDE. But the active components of the model (simple modules) have to be programmed in C++, using the simulation kernel and class library. The NED file structure, the models with parameter, gates and connection is shown in the following box.

```

network SN
{
  parameters:
    double field_x;
    double field_y;
    double field_z;
    double numNodes;
        @display("bgb=2000,1000");
   submodules:

        N0: SensorNodes {
            parameters:
            // most parameters are set with the .ini file
                    load = 1.71987;
                    nodeColor = 0;
                    @display("i=block/circle_vs, blue;p=25,25");
            gates:
                    in[8];
                    out[8];
        }
        N1: SensorNodes {
            parameters:
            // most parameters are set with the .ini file
                    load = 1.64169;
                    nodeColor = 0;
                    @display("i=block/circle_vs, blue;p=50,25");
            gates:
                    in[10];
                    out[10];
                           ......
        connections:
                    ....
}
```

The OMNeT++ configuration and models parameter is provided in the configuration file which is called *omnetpp.ini.* This file contains settings which control execution of the simulation.

When the program starts, it reads all the NED files first which contains model structure. Then it reads the configuration file which is omnetpp.ini. The user interfaces is provided by the link which has created between the OMNeT++ simulation kernel and the code. The OMNet++ has both command line (batch) and interactive graphical user interfaces.

The simulation results have been written in output vector, output scalar files and also the user's own output files. By using the analysis tool in Simulation IDE, the results can be visualized. The result files are text-based; therefore they can be further processed by other ways like R, Matlab or other tools.

It would be interesting to extend this simulation by implementin it on the Castalia simulator to see the DCFA algorithms implementation result on SINR interference model. The Castalia simulator is a generic simulator intended for first order validation of high level algorithms.

4.4 Considerations and implications

Some of the main implementation procedures and timer set up which have been considered on the TOSSIM are presented in the following paragraphs with some brief descriptions and sample codes.

The configuration defined as TimerC and allows aggregation of components as shown below:

```
Configuration TimerC{
    components AppC as myApp, new TimerMilliC();
    //TimerMilliC() is a new instance component
    // that provides the Timer Interface
    myApp.Timer -> TimerMilliC;
    // Component wiring;App uses the Timer interface
    // provided by the  TimerMilliC
}
```

A Timer interface is shown below with two commands and one event. Each component is specified by an interface and it provides "hooks" for wiring components together; this increases runtime efficiency.

```
interface Timer {
        command void  startPeriodic(uint32_t interval);
        command void stop();
        event void fired();
}
```

A module structure which implements the application behavior is shown below:

```
module App{
        uses{
            Interface Timer<TMilli>
              // A timer interface with millisecond precision
        }
}
```

Now the implementation can use the timer interface by calling its command using the call keyword:

```
call Timer.startPeriodic(MLLLSEC_VALUE);
//the fire event is then handled using an event key word
event void MilliT.fired() {

}
```

Task functions has been used as deferred computation mechanism shown in the following sample code.

```


// Signaled by interrupt handler
event void Receive.receivemsg(message_t* msg) {
if (recv_task_busy) {
return; // Drop!
}
recv_task_busy = TRUE;
curmsg = msg;
post recv_task();
}
task void recv_task() {
// Process curmsg ...
recv_task_busy = FALSE;
}
```

5

Using the MicaZ platform for evaluation

The MicaZ platform implementation of the DCFA and analysis has been done by another group of students. The result of the DCFA algorithm on MicaZ platform experiments have been compared with the simulators results in this study to have robust conclusion of the DCFA algorithm behavior.

The testbed is shown in figure 5.1 with a base station sitting on a 45 cm high table with approximately the same distance from the two super-tiles, around 3.5m away on each side. The height is preferred so as to increase the transmission range. In total 11 MicaZ motes have been used. The DCAF algorithm was deployed in 8 of them, where each super-tile consists of only one tile with four nodes. One central Base Station (BS) is attached to the PC and two motes are used as Cluster Heads (CH) receiving messages from the nodes and passing it to the base station.

The nodes have been running on RFPower = 7 equivalent to -15db with the observed transmission range up to 3m on the floor; and hence the two tiles have been kept at around 7mts apart less likely to interfere. The cluster heads just listen for the packets from the nodes and transmit them to the BS, run at RFPower = 31, equivalent to 0db (the max range for cc2420 radio). These CHs transfer the received message to the BS on different times so as to avoid collision at the BS. The BS is used to synchronize the nodes in such a way that it sends a periodic start message in every 520 ms with one slot being 100ms; hence, all nodes follow its clock. Nodes run for one phase and wait for the start message from the BS before starting the next. This should guarantee that the BS message reaches all nodes, otherwise they will not start their transmission, or if they miss any of these start messages, they lag behind the others.

In the MicaZ platform experiment, the success rate of the DCFA algorithm on the static nodes was computed. In this experiment, two super tiles with two tiles and each tile with two static nodes without including the local BS have been considered. From the experiment, the success rate result of the algorithm in each phase is shown in graph 5.2. As can be seen, the DCFA algorithm is 100% successful in each phase.

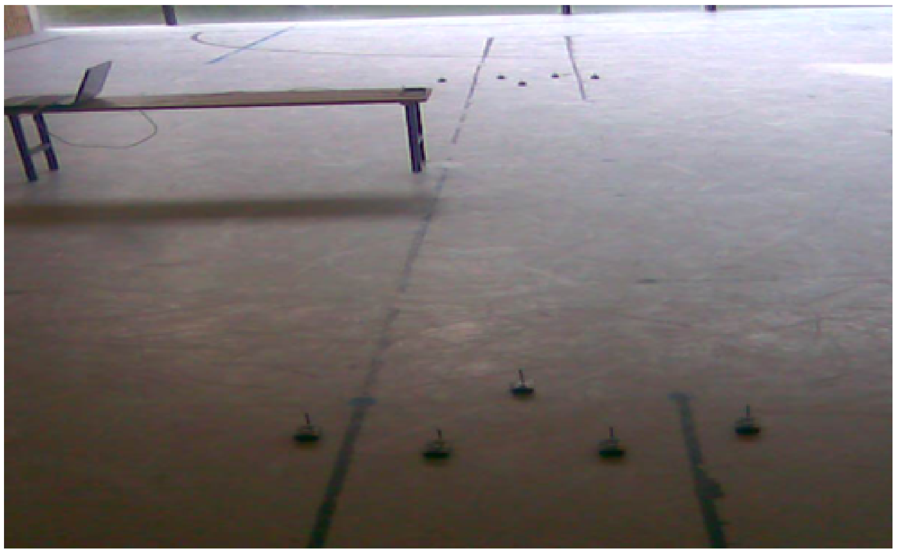

Figure 5.1: Testbed

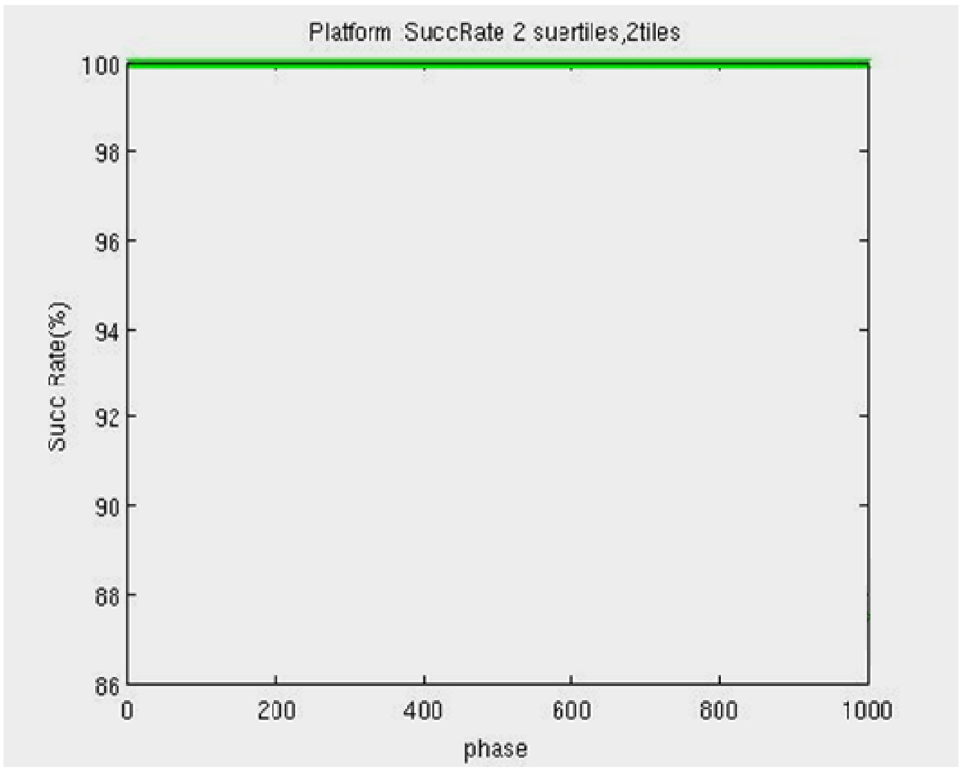

Figure 5.2: Success Rate of DCFA algorithm on Micaz platform

6

Results

DCFA algorithm experiments have been conducted on both simulators and using a real hardware platform. With the simulator, both static and mobile nodes have been considered, but only static nodes on the platform. During the simulator experiment, the two mobility models explained in the previous chapter, have been tested.

The first results have been collected from the TOSSIM simulator.The first result is the success rate of the DCFA algorithm computed at the end of each phase, which shows the success of the exchanged neighborhood information messages between the nodes. The second result is the throughput collected by counting the number of nodes which have reported the right neighborhood information. During the experiment 80 nodes have been considered. Each tile has 5 nodes; this means one node in one tile has four neighbors. In this case, when one node reports that it has 4 neighbors, it means this node knows its whole neighborhood correctly. The nodes report wrong neighbourhood information when they experience some problem in the propagation of the neighborhood information or receiving the neighborhood information. In our experiment, each tile has five nodes in total; even if the nodes are in mobility.

As shown in graph 6.1 the TOSSIM simulation result, the success rate of the DCFA algorithm when exchanging neighborhood information messages between the static nodes is 100% successful. The y-axis represents the success rate and the x-axis represents phases. The graph shows that the neighborhood messages between static nodes have been exchanged successfully without any collision or without any interruption. The second result on the static nodes is shown in graph 6.2. The y-axis represents the throughput and the x-axis represents the phases. It shows how many nodes have reported that they have four neighbors at the end of each phase, and it is possible to see that all the static nodes have reported the correct number of neighbors at the end of each phase.

We continued to investigate the two mobility models. Like the static nodes experiment, the two results have been collected from the first mobility model i.e. the nodes coming on the other lane (cyclic) model. We can see that the success rate of the DCFA algorithm on exchanging neighborhood information messages while the nodes are in mobility is 100% successful. As a conclusion, graph 6.3 shows that the messages between mobile nodes have been exchanged successfully with out any collision. The throughput is shown in graph 6.4. Like the static nodes experiment results, it shows how many nodes have reported that they have four neighbors at the end of each phase. It is possible to see that all mobile nodes have reported the correct number

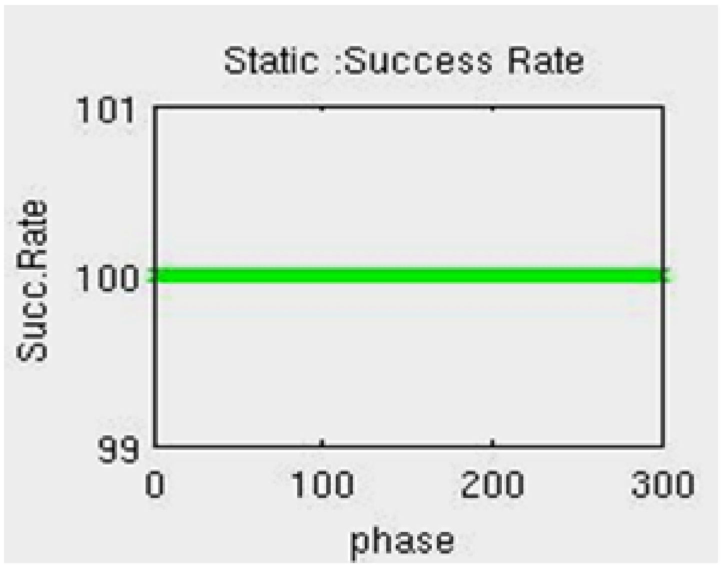

Figure 6.1: Success Rate of DCFA algorithm with the TOSSIM simulator on static nodes

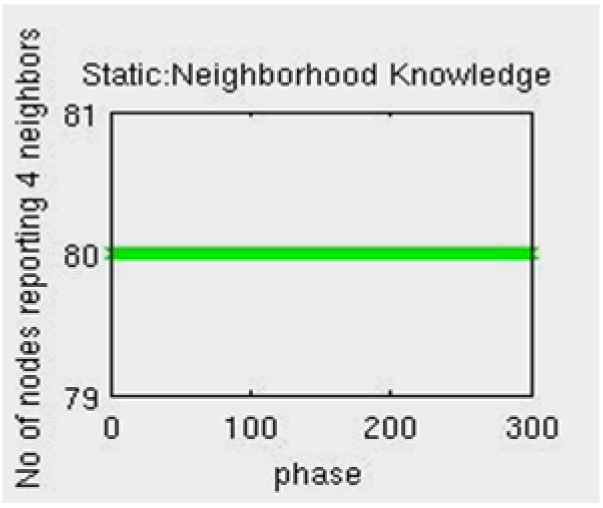

Figure 6.2: Number of nodes which reported their neighborhood information correctly using the TOSSIM simulator on static nodes

of neighbors at the end of each phase.

The results that have been collected in the previous experiments with both static nodes and mobile nodes (the first mobility model) showed that the DCFA algorithm is working correctly. However, the DCFA algorithm has not been working properly on mobile nodes using second mobility model, i.e node jumps to the other side when it reaches on the border of the network (plane or grid in our case). The DCFA algorithm could not stabilize itself immediately when the mobile node jumps to the other corner of the network. As shown in figure 6.5, the success rate of the DCFA algorithm with this model is far from 100% successful. When the mobile nodes

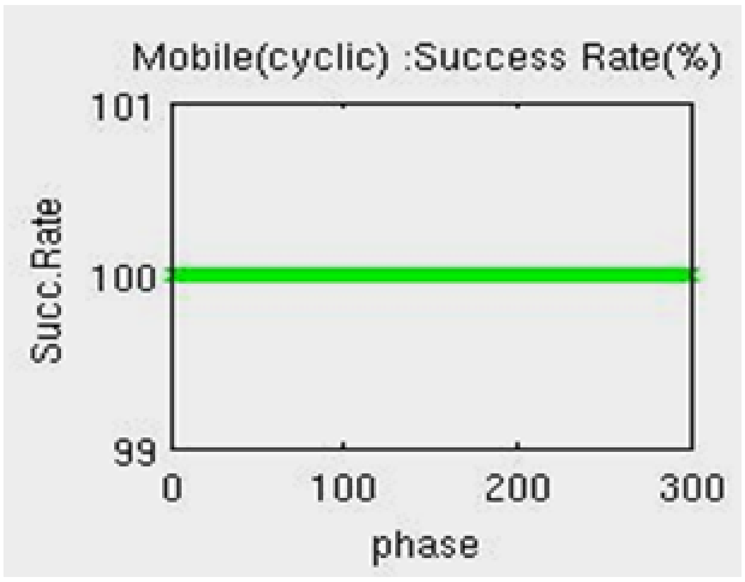

Figure 6.3: Success Rate of DCFA algorithm with the TOSSIM simulator with mobile nodes by following the first mobility model

jump to the other side of the grid, the DCFA algorithm takes time to stabilize and function properly. When it stabilizes after a few phases, another mobile node jumps to the other corner of the tile and disturbs the process. The problem is that when one mobile node jumps to the other corner tile, the mobile nodes in this tile do not have any information about this new node. They learn about this new node after exchanging some messages, and stabilize. However this stabilization depends on the speed of the nodes mobility. If the nodes are moving fast, its hard for the algorithm to stabilize.

When comparing the above results with the MicaZ platform results from the previous chapter, it is possible to conclude that the DCFA algorithm works property in real world situations. However, if an unusual situation happens in this real world environment, the DCFA algorithm does not stabilize itself. However, if this unusual situation happens only for a very short time, the algorithm may stabilize itself after a few neighborhood messages are exchanged between nodes, and then starts working properly.

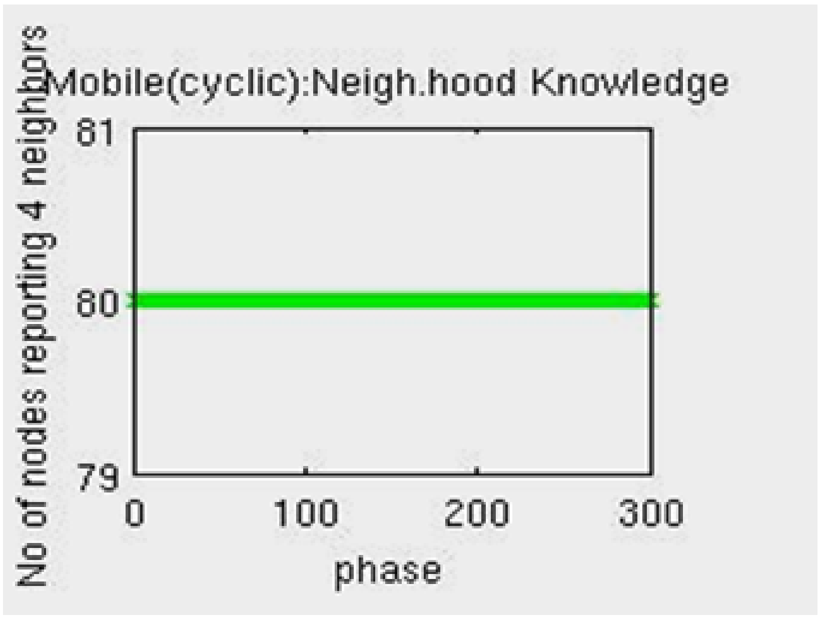

Figure 6.4: Number of nodes which reported their neighborhood information correctly using the TOSSIM simulator with mobile nodes by following the first mobility model

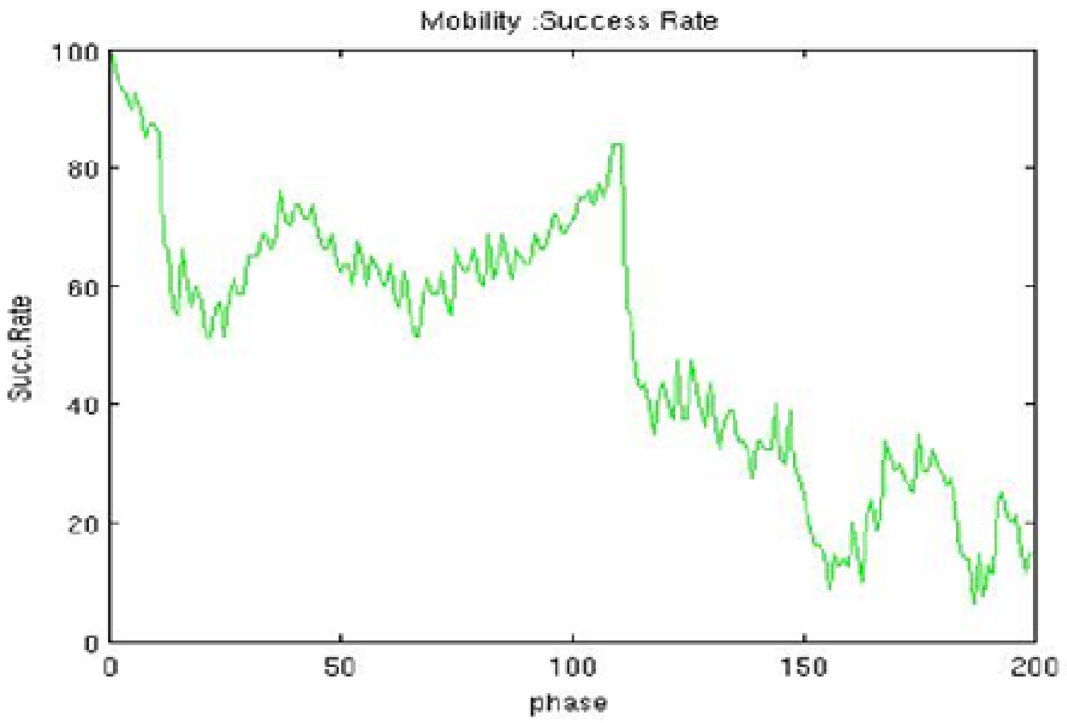

Figure 6.5: Success Rate of DCFA algorithm on TOSSIM simulator with mobile nodes by following jumping to the beginning model

7

Discussion

7.1 Statement of primary findings of the study

The primary finding of this study is that the DCFA algorithm gives the expected result on under certain circumstances. It has been experiencing some problems on unexpected situations which we have created in the experiment to challenge it, and see it's robustness and resistance on this kind of situations. The pros and cons of the algorithm will be discussed in the next paragraphs.

The DCFA algorithm maintains neighborhood knowledge when the nodes move in and out of each others broadcast range. This behavior works quite well in the first mobility model. The DCFA algorithm has reliable communication scheme when the nodes are in the static or mobility. However, the DCFA algorithm does not show good results in the second mobility model.

The DCFA algorithm has showed some problems to fully satisfy the required properties of the MAC protocols. The important and main properties of a well-defined MAC protocol are scalability and adaptability in dynamic networks. The changes in network size, node density and topology should be handled rapidly and effectively in the MAC protocols for a successful adaptation of nodes in the network [1]. Some of the reasons behind the network property changes are limited node lifetime, having additional new nodes in the network and varying interference which may alter the connectivity and hence the network topology [1].

7.2 Results from using the OMNET++ simulator

One of the main strengths of the DCFA algorithm is to do not have any interference between two nodes in rounds. However, SNR is not considered in its calculations. Therefore, the implementation result which is found on the OMNET++ simulator showed that all the messages are propagating between nodes without any interference. This means that the implementation is giving the expected result from the DCFA algorithm. However, it was necessary to test the DCFA algorithm by SINR interference model which was embedded in the simulator, since it has to be tested on how it handles the challenge of interference situations between the nodes. SINR considers interference with noise, and this helps to see the algorithms response on interference between nodes. Since OMNET++ does not implement SINR, the interference situation of the

DCFA algorithm implementation on the OMNET++ simulator was written in C++ code. However, this C++ code, does not have many interference challenges like SINR interference model to challenge the DCFA algorthm enough. Because of this reason, the results from the DCFA algorithm implementation on the OMNET++ simulator has not been added in the results since interference is one of the main properties which has to be seen in protocols. The implemented interference environment in C++ is not that challenging as SINR.

There is another simulator for wireless sensor networks called Castalia. It's based on the OMNET++ simulator. Castalia can be used to test distributed algorithms and/or protocols in realistic wireless channel and radio models with a realistic node behavior; especially relating to access of the radio. Unlike the OMNET++ simulator, Castalia uses the SINR interference model in its simulation. We had planned to continue working on the Castalia simulator, however, we found it difficult to continue because of the time and shortage of enough resources about Castalia. Therefore the implementation has remained on OMNET++ simulator. The Castalia extension work is left for the future.

From the experiments we have performed on the DCFA algorithm, all the required and expected properties of the algorithm have been met except one main property which many MAC protocols have to meet namely self stabilization which is not seen in the DCFA algorithm. The DCFA algorithm does not have the capacity to stabilize itself when unexpected things happen in the network.

The remaining findings have positive feedback for the DCFA algorithm except the above mentioned problem. It has a reliable communication schema for the mobile nodes with collision free and also it maintains neighborhood knowledge while the nodes are moving in and out of each others broadcast range; but it does not have self stabilization. Therefore, this study recommends for future study to modify the DCFA algorithm by adding self stabilization feature for any kind of situations in the algorithm.

8

Conclusion

The main objective of this thesis is to evaluate the performance of the DCFA algorithm. From the DCFA protocol experiments study, one of the criticisms has focused on the self stabilization. The self stabilization property of the DCFA algorithm has been studied using two mobility models. The first mobility model has followed the real world car mobility and the self stabilization of the DCFA algorithm has been giving the expected results. However the second mobility model has completely disturbed the slot assignment schema. In this second mobility model, when nodes reach the border of the network, it jumps to the other corner of the network and continue its mobility. When the jumped node leaves its neighbor nodes and moves to a new neighborhood, it does not has any knowledge of its new neighbors; and the slot will be disturbed. The second mobility model has been used to see how this algorithm can resist any kind of challenges and stabilize itself from any environmental change and continue its process without any problem.

The very nature of the wireless communications inherently exposed to a lots of long term and transient faults such as interference and message loss. Therefore protocols deployed on these areas should be highly robust to faults and should be able to self stabilize within a bounded period of time. The MAC protocols need evaluation on simulators to make sure that they are giving the expected objective and also to encourage modification / upgrade on the protocol. To have accuracy on the evaluation work of the protocols, it is a good idea to consider the real platform evaluation as well. The implementation of the DCFA algorithm on the TOSSIM simulator has shown that the DCFA algorithm never stabilizes unless it gets some lucky situations.

From the findings on the self stabilization, we have reached to the conclusion that the DCFA algorithm can not recover quickly from an error and continue its proper work if any thing goes wrong in the network. The implementation of the DCFA algorithm highly relies on the neighborhood knowledge and if by any means the nodes in a neighborhood happen to have inconsistent knowledge of each other due to neighborhood message loss, they might pick a colliding slot. And such a collision goes on reproducing itself in the remaining phases and never recovers unless by some lucky situations.

Bibliography

[1] I. Demirkol, C. Ersoy, F. Alagoz, MAC protocols for wireless sensor networks: a survey, IEEE Communications Magazine 44 (4) (2006) 115–121.
URL `http://dx.doi.org/10.1109/MCOM.2006.1632658`

[2] G. Lu, B. Krishnamachari, C. S. Raghavendra, An adaptive energy-efficient and low-latency mac for data gathering in wireless sensor networks, Parallel and Distributed Processing Symposium, International 13 (2004) 224a.

[3] W.-Z. Song, R. Huang, B. Shirazi, R. LaHusen, Treemac: Localized tdma mac protocol for real-time high-data-rate sensor networks, Pervasive Mob. Comput. 5 (6) (2009) 750–765.
URL `http://dx.doi.org/10.1016/j.pmcj.2009.07.004`

[4] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Standard 802.11 (Jun. 1999).

[5] I. Chlamtac, Mobile ad hoc networking: imperatives and challenges, Ad Hoc Networks 1 (1) (2003) 13–64.
URL `http://dx.doi.org/10.1016/S1570-8705(03)00013-1`

[6] S. Viqar, J. L. Welch, Algorithmic aspects of wireless sensor networks, Springer-Verlag, Berlin, Heidelberg, 2009, Ch. Deterministic Collision Free Communication Despite Continuous Motion, pp. 218–229.
URL `http://dx.doi.org/10.1007/978-3-642-05434-1_22`

[7] M. I. Miloš Jevtic, Nikola Zogovic, I. Goran Dimic, Member, Evaluation of Wireless Sensor Network Simulators , Communications Magazine, IEEE.

[8] D. Cavin, Towards reliable communication and agreement in mobile ad-hoc networks: Algorithms, simulation and testbed.

[9] C. Busch, M. Magdon-Ismail, F. Sivrikaya, B. Yener, Contention-free mac protocols for wireless sensor networks, in: IN DISC, 2004, pp. 245–259.

[10] I. Stojmenovic, Simulations in wireless sensor and ad hoc networks: matching and advancing models, metrics, and solutions, Communications Magazine, IEEE 46 (12) (2008) 102–107.
URL `http://dx.doi.org/10.1109/MCOM.2008.4689215`

[11] de La, R. Rebeyrotte, K. Runser, J.-M. Gorce, Enhancing ns-2 simulator for high mobility ad hoc networks in car-to-car communication context., IASTED International Conference on Antennas, Radar and Wave Propagation (ARP), Banff, Alberta, Canada, 2005.

[12] M. Korkalainen, M. Sallinen, N. Kärkkäinen, P. Tukeva, Survey of wireless sensor networks simulation tools for demanding applications, in: Proceedings of the 2009 Fifth International Conference on Networking and Services, ICNS '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 102–106.
URL `http://dx.doi.org/10.1109/ICNS.2009.75`

[13] J. Z. D. L. Ziqing Zhang, Hai Zhao, Research on wireless sensor networks toplogy models, Northeastern University, Shenyang, China, 2010.

[14] P. Levis, TinyOS Programming, 2006.

[15] C. Sienkiewcz, Porting TinyOS to an Amulet2e Sensor Mote, 2010.

[16] D. C. E. B. David Gay, Philip Levis, nesC 1.1 Language Reference Manual, 2003.

[17] P. Levis, N. Lee, M. Welsh, D. Culler, Tossim: accurate and scalable simulation of entire tinyos applications, in: Proceedings of the 1st international conference on Embedded networked sensor systems, SenSys '03, ACM, New York, NY, USA, 2003, pp. 126–137.
URL `http://doi.acm.org/10.1145/958491.958506`

[18] E. Perla, Art, R. S. Carbajo, M. Huggard, C. M. Goldrick, PowerTOSSIM z: realistic energy modelling for wireless sensor network environments, in: Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks, PM2HW2N '08, ACM, New York, NY, USA, 2008, pp. 35–42.
URL `http://dx.doi.org/10.1145/1454630.1454636`

[19] Oment++ - user manual.

[20] J. Brody, P. Yager, R. Goldstein, R. Austin, Biotechnology at low Reynolds numbers, Biophysical Journal 71 (6) (1996) 3430–3441.
URL `http://linkinghub.elsevier.com/retrieve/pii/S0006349596795383`