THESIS
2
2003
56620985

LIBRARY
Michigan State
University

This is to certify that the
thesis entitled

COLLISION-FREE COMMUNICATION IN SENSOR NETWORKS

presented by

UMAMAHESWARAN ARUMUGAM

has been accepted towards fulfillment
of the requirements for the

M.S. degree in COMPUTER SCIENCE AND ENGINEERING

S Kulkarni

Major Professor's Signature

Oct 8, 2003

Date

MSU is an Affirmative Action/Equal Opportunity Institution

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.
MAY BE RECALLED with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE

6/01 c:/CIRC/DateDue.p65-p.15

COLLISION-FREE COMMUNICATION IN SENSOR NETWORKS

By

Umamaheswaran Arumugam

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Computer Science and Engineering

2003

ABSTRACT

Collision-Free Communication in Sensor Networks

By

Umamaheswaran Arumugam

Sensors networks are often constrained by limited power and limited communication range. If a sensor receives two messages simultaneously then they collide and both messages become incomprehensible. In this thesis, we present a simple time division multiple access (TDMA) algorithm for assigning time slots to sensors and show that it provides a significant reduction in the number of collisions incurred during communication. We present TDMA algorithms customized for different communication patterns that occur commonly in sensor networks. Our solution deals with several difficulties, e.g., unidirectional links, unreliable links, long links, failed sensors, sensors that are sleeping in order to save energy, and location errors. Our algorithms are self-stabilizing, i.e., TDMA is restored even if the system reaches an arbitrary state where the sensors are corrupted or improperly initialized. Further, we show that our algorithms ensure collision-freedom whereas collision-avoidance protocols like carrier sense multiple access (CSMA) suffer significant number of collisions. Moreover, as an application to our TDMA algorithms, in this thesis, we present transformation algorithms for the sensor network model of computation, called, *write-all-with-collision* model. Using the transformation algorithms, we transform programs written in this model into programs in other models considered in the literature, and vice versa.

To mom and dad

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Dr. Sandeep Kulkarni for his support and guidance during my master's program. He helped me understand what research is and the approach to solve research problems. He introduced relatively new areas of research and my interactions with him helped me understand new problems and design simple and efficient solutions. I would like to express my sincere thanks to him for the time and effort he spent in discussions that were extremely helpful in my research, writings and presentations.

Next, I would like to thank my guidance committee members Dr. Sandeep Kulkarni, Dr. Abdol Esfahanian, and Dr. Betty Cheng for their valuable comments/remarks of my thesis. I express my thanks to Dr. Cheng for reading my thesis and identifying errors in writing. Further, I express my thanks to Dr. Esfahanian for his interesting questions and remarks during my thesis presentation. Moreover, I am very thankful to Dr. Anil Jain, Dr. Betty Cheng, and Dr. Sandeep Kulkarni for providing recommendation letters to the department for Ph.D. admissions.

I would like to thank my lab mates Karun Biyani, Limin Wang, Bruhadeshwar Bezawada (Bru!), Ali Ebnenasir, Borzoo Bonakdarpour, and all the other SENS students for their valuable help during my master's program. Especially, I would like to thank Ali for proof reading my papers and my thesis with lots of care and dedication. Furthermore, I would like to acknowledge Karun and Limin for helping me out with the field experiments associated with the DARPA NEST project "A Line in the Sand".

I would like to thank Thangavelu Arumugam (Thangam!), Sridevi Srinivasan, and Janny Henry Rodriguez for sharing their valuable time and experiences with me. I

would also like to thank my friends S. Suresh Babu, G. R. Arun, and R. Naveen Kumar. I am grateful to have such good friends from my high school days. They are always very supportive of my decisions and endeavors. Thanks also goes to all my Anna University friends.

Finally, I would like to thank my parents for their love and affection. I thank them for their support in whatever efforts I make. Especially, I thank them for allowing me to switch my field of specialization, from Medicine to Computer Science. I would like to thank my mom for her strong belief in me. Also, I thank my parents for their constant words of encouragement and prayers.

TABLE OF CONTENTS

Page

List of Figures ix

List of Tables xiii

Chapters:

1. Introduction 1

   1.1 Communication in Sensor Networks 2
      1.1.1 Application: Model Conversions for Sensor Networks 3
   1.2 Thesis Contributions 4
   1.3 Organization of the Thesis 5

2. Model and Assumptions 7

3. Collision-Free Diffusion 9

   3.1 Version 1: Communicate 1, Interfere 1 11
   3.2 Version 2: Communicate x, Interfere x 13
   3.3 Version 3: Communicate 1, Interfere y 14
   3.4 Version 4: Communicate x, Interfere y 16
   3.5 Diffusion by an Arbitrary Sensor 16
   3.6 Observations about Our Algorithm 17

4. Time Division Multiple Access (TDMA) 19

   4.1 Simple TDMA Algorithm 20
   4.2 Stabilization of TDMA and Diffusion 21
   4.3 Extensions: Other Topologies 22
      4.3.1 TDMA Algorithm for Hexagonal Grids 23
      4.3.2 TDMA Algorithm for Triangular Grids 25
      4.3.3 Diffusion in Multi-Dimensional Grids 26
      4.3.4 Diffusion in Arbitrary Graphs 27
   4.4 Extensions: Dealing with Failed/Sleeping Sensors 29
      4.4.1 Diffusion in Imperfect Grids or Grids with Failed Sensors/Links 29
      4.4.2 Extending TDMA for Dealing with Failed Sensors 30

5. TDMA Service for Sensor Networks 33

5.1 Algorithms for TDMA Service 34
5.1.1 TDMA Service for Broadcast 34
5.1.2 TDMA Service for Convergecast 35
5.1.3 TDMA Service for Local Gossip 37
5.2 Implementation of TDMA Service 39
5.3 Simulation Model 41
5.4 Simulation Results 45
5.4.1 Broadcast 45
5.4.2 Convergecast 46
5.4.3 Local gossip 49
5.4.4 Effect of Location Errors 50
5.4.5 Effect of Grouping Constant 54

6. Application: Model Conversions for Sensor Networks 57

6.1 Atomicity Models, Preserving Stabilization and System Assumptions 60
6.2 Read/Write Model to WAC Model in Untimed Systems 65
6.2.1 Optimality Issues 67
6.2.2 Stabilization Issues 72
6.3 Read/Write Model to WAC Model in Timed Systems 73
6.3.1 Transformation for Grid Topology 74
6.3.2 Transformation for Arbitrary Topology 76
6.3.3 Preserving Stabilization during Transformation 78
6.4 WAC Model to Read/Write Model 80
6.4.1 Preserving Stabilization during Transformation 83
6.5 Discussion 84

7. Related Work 88

7.1 Sensor Networks 88
7.1.1 Challenges and Opportunities 89
7.1.2 Sensor Hardware, Operating System and Programming Language 90
7.2 Communication Protocols for Radio/Wireless Networks 93
7.3 MAC Protocols for Sensor Networks 95
7.3.1 CSMA Based MAC Protocols 96
7.3.2 Collision-Free MAC Protocols 97
7.3.3 Energy Efficient MAC Protocols 99
7.4 Time Synchronization 102
7.5 Self-Stabilization 106
7.5.1 Self-Stabilizing Algorithms 106

7.5.2 Stabilization Preserving Model Conversions 108

8. Conclusion and Future Work 111

Bibliography 116

LIST OF FIGURES

Figure **Page**

2.1 Sensor network topology where a sensor communicates with its distance 1 neighbors and interferes with its distance 2 neighbors. The numbers associated with a sensor indicates the grid position of the sensor. . . 8

3.1 Collision-Free Diffusion 10

3.2 Sample diffusion in networks where a sensor communicates with its distance 1 neighbors. The number associated with a sensor shows the slot in which it should transmit. 11

3.3 Version 1: Communicate 1, Interfere 1 12

3.4 Sample diffusion in networks where a sensor communicates with its distance 2 neighbors. The number associated with a sensor shows the slot in which it should transmit. 13

3.5 Version 2: Communicate x, Interfere x 14

3.6 Sample diffusion in networks where a sensor communicates with its distance 1 neighbors and interferes with its distance 2 neighbors. The number associated with a sensor shows the slot in which it should transmit. 14

3.7 Version 3: Communicate 1, Interfere y 15

3.8 Sample diffusion in networks with arbitrary initiator 17

4.1 Time Division Multiple Access (TDMA) 19

4.2 Simple TDMA Algorithm 20

4.3 Sample TDMA in networks where a sensor communicates with its distance 1 neighbors and interferes with its distance 2 neighbors. The numbers associated with a sensor shows the slots in which it could transmit. 21

4.4 TDMA slot assignment in hexagonal-grid network where communication range=1 and interference range=2. The number associated with each sensor denotes the time at which it can send a message. Slots for some sensors are not shown. 23

4.5 TDMA algorithm for hexagonal grids 24

4.6 Initial slot assignment in triangular-grid network where communication range=1 and interference range=2. The number associated with each sensor denotes the time at which it forwards the diffusion message. . 25

4.7 Three-dimensional grid network where a sensor communicates with its distance 1 neighbors in its x, y and z axes. The number associated with a sensor shows the slot in which it should transmit. 26

4.8 Diffusion in n-D grids 27

4.9 Mapping an arbitrary tree into a 2-dimensional grid 28

4.10 Diffusion in presence of failed/sleeping sensors 30

5.1 Sample TDMA in networks where a sensor communicates with its distance 1 neighbors and interferes with its distance 2 neighbors. The numbers associated with a sensor shows the slots in which it could transmit. 34

5.2 Sample diffusion for convergecast where communication range=1, interference range=2. The number associated with each sensor denotes the time at which it forwards the diffusion message. 35

5.3 TDMA algorithm for convergecast 36

5.4 TDMA slot assignment for convergecast where communication range=1, interference range=2. The number associated with each sensor denotes the time at which it can send a message. Some initial slots are not shown. 36

5.5 TDMA algorithm for local gossip 38

5.6 TDMA slot assignment for gossip where communication range=1, interference range=2. The number associated with each sensor denotes the time at which it can send a message. Some initial slots are not shown. 38

5.7 Sample data structure, interface, and component wiring definitions in nesC 42

5.8 Results for broadcast with communication range = 1, interference range = 1 46

5.9 Results for broadcast with communication range = 1, interference range = 2 47

5.10 Results for convergecast with communication range = 1, interference range = 1 48

5.11 Results for convergecast with communication range = 1, interference range = 2 49

5.12 Results for local gossip; (a) and (b) with communication range = 1, interference range = 1, (c) and (d) with communication range = 1, interference range = 2 51

5.13 Results for broadcast with location errors 53

5.14 Results for convergecast and local gossip with location errors 54

5.15 Effect of grouping constant; with communication range = 1, interference range = 1, for (a) convergecast and (b) local gossip, and with communication range = 1, interference range = 2, for (c) convergecast and (d) local gossip 56

6.1 In the transformed program, 'A' will execute actions of processes 0 and 6, 'B' will execute actions of processes 1, 3 and 5, and so on. 65

6.2 Read/write model to WAC model in untimed systems 66

6.3 Impossibility of executing more than one process in untimed systems 69

6.4 Executing more than one process in untimed systems. The numbers associated with each process denotes the number of writes the process executes in the corresponding round. 71

6.5 Time slots for processes in a grid. The numbers associated with a process show the first two slots in which it could execute. 74

6.6 Transformation for grid topology 75

6.7 Transformation using graph coloring. The number associated with each process denotes the color of the process. 77

6.8 Transformation for arbitrary topology 77

6.9 WAC model to read/write model 82

LIST OF TABLES

Table | Page

5.1 Results for $\mu = 0.4$, $\sigma = 0.2$, and interference range=2 55

7.1 Comparing different energy-efficient MAC protocols 102

7.2 Comparing different time-synchronization protocols 105

CHAPTER 1

Introduction

In recent years, sensor networks have become popular in the academic and industrial environments due to their applications in data gathering, active and passive tracking of unexpected/undesirable objects, environment monitoring and unattended hazard detection [1–3]. Due to their low cost and small size, it is possible to rapidly deploy them in large numbers. These sensors are resource constrained and can typically communicate with other (neighboring) sensors over a wireless network. However, due to limited power and communication range, they need to collaborate to achieve the required task.

One of the important issues in sensor networks is message collision: Due to the shared wireless medium, if a sensor simultaneously receives two messages then they collide and, hence, both messages become incomprehensible. Such collision is undesirable as it results in wastage of power to transmit the message that resulted in a collision. And, it also requires several transmissions to get a single message across. Moreover, collision detection is further complicated by the fact that a given message may collide at one sensor and be received correctly at another sensor.

Collision among messages is especially problematic in the context of *system-wide computations* where some sensors need to communicate some information to the entire network (respectively, a subset of the network that satisfies the geographic properties of interest). Such computations arise when a sensor needs to communicate the observed value to the *base station* or when we want to organize these sensors in a suitable topology (e.g., tree). In such system-wide computation (diffusion), every

sensor that receives a message transmits it to its neighbors (in the given direction). It follows that at any time multiple sensors may be forwarding the sensor values to their respective neighbors. Therefore, the possibility of collision increases.

Challenges in communication. One of the important issues in sensor networks is the scenario where two sensors can communicate with each other only with a very low probability. Also, sensor networks suffer from unidirectional links where one sensor can communicate with another with a high probability although the probability of the reverse communication is very low. In a situation where sensor j can only communicate occasionally with sensor k, it is expected that the signal strength received by k is so small that k cannot correctly determine all the bits in that message. However, the signal strength that k receives is often strong enough that it can corrupt another message that was sent to k at the same time. The above discussion suggests that we need to consider the situation where two sensors cannot effectively *communicate* with each other although they can effectively *interfere* with each other.

Now, we briefly outline the collision-free communication algorithms proposed for sensor networks. Then, in Section 1.1.1, we state an application of our collision-free communication algorithms. Subsequently, in Section 1.2, we list the contributions of this thesis.

1.1 Communication in Sensor Networks

In this thesis, we propose collision-free diffusion and time division multiple access (TDMA) algorithms for sensor networks. A collision-free diffusion is advantageous in obtaining clock synchronization and TDMA. More specifically, when a sensor receives the diffusion message, it can uniquely determine its clock by considering the clock value when the diffusion was initiated and the path that the diffusion took to reach

that sensor. If clocks are synchronized, we can assign slots to each sensor such that simultaneous message transmissions by two sensors do not collide.

A closer inspection of diffusion and TDMA shows that these two problems are interdependent. More specifically, if clocks are not synchronized then the diffusion may fail in the following scenario: the clock values of two sensors differ, one of these sensors transmits the diffusion message and the other sensor transmits an unrelated message at the same time. It follows that a collision in this situation will prevent the diffusion message from reaching all the desired destinations. Moreover, if the diffusion does not complete successfully then the clocks may remain unsynchronized forever. It is therefore necessary that diffusion be stabilizing fault-tolerant [4, 5], i.e., starting from an arbitrary state, the system should recover to states from where subsequent diffusing computation is collision-free. A stabilizing fault-tolerant solution also deals with the case where the sensors are inactive for a long time and subsequently become active, although at slightly different times.

1.1.1 Application: Model Conversions for Sensor Networks

We observe that the sensor networks pose a new model of computation. We can view this model as a *write-all-with-collision* (WAC) model. Intuitively, in this model, in one atomic action, a sensor (process) can update its own state and the state of all its neighbors. However, if two sensors (processes) simultaneously try to update the state of a sensor (process), say k, then the state of k remains unchanged.

Designing model conversion algorithms is one of the important research areas in distributed systems. It allows one to write programs in one model (typically a specific/restrictive model) and then later, transform them into another model (typically a general/less restrictive model). We apply our collision-free diffusion and

TDMA algorithms to transform a program in other distributed models of computation into a program in the WAC model for timed/synchronous systems. Specifically, for timed/synchronous systems with grid-based communication topology, we propose a transformation algorithm that transforms a program in the read/write model, where in one atomic action, the program can either write its own state or read the state of one of its neighbors, into a program in the WAC model.

Additionally, we identify other transformations, where a program in the read/write model is transformed into a program in the WAC model for untimed/asynchronous systems. Furthermore, we present an algorithm that transforms a program in the WAC model into a program in the read/write model.

1.2 Thesis Contributions

In this thesis, we concentrate on designing collision-free communication algorithms for sensor networks. Also, as an application to our collision-free communication algorithms, we design model conversion algorithms, where programs in the WAC model are converted into programs in other distributed computing models, and vice versa. The main contributions of the thesis are as follows:

- We present TDMA algorithms with a deterministic startup algorithm (i.e., collision-free diffusion) for commonly occurring communication patterns in senor networks. Further, we show how stabilizing fault-tolerance [4, 5] can be added to our TDMA algorithms. Thus, starting from an arbitrary state, the TDMA algorithm recovers to states from where collision-free communication among sensors is achieved. Additionally, we show that the TDMA algorithms can deal with unreliable sensors/links.

- We verify that the proposed TDMA algorithms are collision-free (using simulations). Furthermore, we compare the TDMA algorithms with collision-avoidance protocols like carrier sense multiple access (CSMA) and show that CSMA suffers from significant number of collisions. Additionally, we show that our TDMA algorithms can tolerate errors in the location, i.e., even if the sensors are moved slightly from their ideal location, the percentage of collisions is within the application requirements.

- We outline the middleware architecture of our TDMA algorithms. Towards this end, we show how our algorithms are implemented as a middleware service. We identify the application programming interfaces (APIs) of the proposed service.

- We present an application of collision-free diffusion and TDMA algorithms in model conversions for sensor network. We consider the *write all with collision* (WAC) model that commonly occurs in sensor networks, and present transformations from (respectively, to) programs in the WAC model to (respectively, from) programs in other models of computation. Further, for timed systems, we show that if the original program is stabilizing fault-tolerant then the transformed program is also stabilizing fault-tolerant. In other words, we show that the transformations are stabilization preserving for timed systems.

1.3 Organization of the Thesis

The thesis is organized as follows. In Chapter 2, we describe the sensor network model and state the assumptions made in this thesis. In Chapter 3, we present the collision-free diffusion algorithm. Specifically, we present four versions of the collision-free diffusion algorithm based on the ability of sensors to communicate with each other and their ability to interfere with each other. Subsequently, in Chapter

4, we discuss a simple TDMA algorithm for sensor networks. Further, we show how our algorithms can be made stabilizing. Also, we provide extensions to the collision-free diffusion and TDMA algorithms, where some of the sensors in the network are failed/sleeping, or the communication topology is different. In Chapter 5, we present the TDMA algorithms customized for different communication patterns that occur commonly in sensor networks. Further, we show how our TDMA algorithms can be implemented as a middleware service for sensor networks, especially, for MICA motes [3,6] developed by University of California, Berkeley. In Chapter 6, we present stabilization preserving transformation algorithms for transforming a program in the sensor network model into a program in other models, and vice versa. Finally, in Chapter 7, we identify the related work, and in Chapter 8, we present the concluding remarks and the scope for future research.

CHAPTER 2

Model and Assumptions

In this chapter, we present the sensor network model and state the assumptions made in our algorithms.

Topology. Initially, we assume that sensors are arranged in a grid where each sensor knows its location in the network (geometric position). Each message sent by a sensor includes this geometric position. Thus, a sensor can determine the position, direction and distance (with respect to itself) of the sensors that send messages to it. We assume that the sensor network has a perfect grid topology (cf. Figure 2.1) and no sensors have failed or are in sleeping state. By making these assumptions, we can design algorithms for perfect grid-based sensor networks. Then, we extend the algorithms to deal with the case where sensors (other than the initiator) have failed.

Communication and interference ranges. We assume that each sensor has a communication range and an interference range. Communication range is the distance up to which a sensor can communicate with certainty/high probability. Interference range is the distance up to which a sensor can communicate, although the probability of such a communication may be low. Thus, given two sensors, j and k if k is in the interference range of j but k is not in the communication range of j then k receives messages sent by j with a low probability. However, if k receives another message while j is sending a message, it is possible that collision between these two messages can prevent k from receiving either of those messages. Based on the definition of the interference range, it follows that it is at least equal to the communication range. For example, in Figure 2.1, the communication range is 1 and interference range is

2. Also, we assume that the sensors are aware of their communication range and interference range.

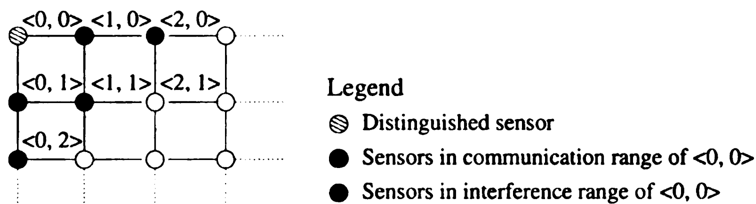


Figure 2.1: Sensor network topology where a sensor communicates with its distance 1 neighbors and interferes with its distance 2 neighbors. The numbers associated with a sensor indicates the grid position of the sensor.

Distinguished sensor (process). We assume that there is exactly one initiator that initiates the diffusion. For simplicity, initially, we assume that the sensor at the left-top (at location $\langle 0, 0 \rangle$) is the initiator (cf. Figure 2.1). This assumption is made since we can view the diffusion as propagating from this sensor to all the sensors in the network in one single (south-east) quadrant. Later, we remove this assumption and provide an algorithm where the initiator is not at the left-top position.

Clock speed. Additionally, we assume that one clock tick of the sensor corresponds to the propagation time of a message. Though the sensors can have high-precision clocks, for communication, we use only the higher-order bits that correspond to the propagation time of a message.

CHAPTER 3

Collision-Free Diffusion

In this chapter, we provide an algorithm for collision-free diffusion for sensors arranged in a two-dimensional grid. As mentioned in Chapter 1, whenever a sensor receives a diffusion message for the first time, it retransmits the message to its neighbors. However, if two (nearby) sensors transmit the diffusion message at the same time then the messages collide. The collision becomes even more problematic in sensor networks where it is often not possible to detect whether collision has occurred or not. Also, it is possible that a message sent by one sensor collides at one receiver whereas another receiver correctly receives it.

To deal with these problems, we define the problem of collision-free diffusion as follows. The first requirement for collision-free diffusion is that the diffusion message should reach every sensor. The second requirement is that collisions should not occur. More specifically, when sensor j transmits a message, it is necessary that a collision should not occur at any sensor that is expected to receive the message from j, i.e., a sensor in the communication range of j. Thus, if sensor k transmits concurrently then the set of sensors in the communication range of j should be disjoint from the set of sensors in the interference range of k. Thus, the problem statement is defined as follows:

Problem Statement: Collision-Free Diffusion
Given a sensor grid; if a sensor initiates diffusion then the following properties should be satisfied:

1. Diffusion message should reach every sensor.
2. If two sensors j and k transmit at the same time,
   (Sensors in communication range of j) $\bigcap$
   (Sensors in interference range of k) = $\emptyset$.

Figure 3.1: Collision-Free Diffusion

We present four versions of our collision-free diffusion algorithm based on the ability of the sensors to communicate with each other and their ability to interfere with each other. For simplicity, in Sections 3.1-3.4, we assume that the sensor at $\langle 0, 0 \rangle$ initiates the diffusion. In Section 3.1, we discuss the algorithm for diffusion in networks where a sensor can communicate only with its distance 1 neighbors. In Section 3.2, we extend this version for diffusion in networks where a sensor can communicate with its distance $x, x \geq 1$, neighbors. In both these algorithms, we assume that the interference range of a sensor is same as its communication range. We weaken this requirement in Sections 3.3, and 3.4. Specifically, in Section 3.3, we extend the first version (cf. Section 3.1) to deal with the case where a sensor can communicate with its distance 1 neighbors and interfere with its distance $y, y \geq 1$, neighbors. And, in Section 3.4, we extend the third version (cf. Section 3.3) to deal with the case where a sensor can communicate with its distance $x, x \geq 1$, neighbors and interfere with its distance $y, y \geq x$, neighbors. In Section 3.5, we provide an algorithm for collision-free diffusion initiated by an arbitrary sensor. Although in Sections 3.1-3.5 we assume that the communication range (respectively, interference range) of all sensors are identical, observations made in Section 3.6 show that the algorithm can be applied even if they are different.

3.1 Version 1: Communicate 1, Interfere 1

Consider a simple grid network where a sensor can communicate with sensors that are distance 1 away (cf. Figure 3.2). (Note that, in these examples we have used the Manhattan distance between sensors where the distance between two sensors $\langle x_1, y_1 \rangle$ and $\langle x_2, y_2 \rangle$ is $|x_1 - x_2| + |y_1 - y_2|$. Our algorithms can be applied even if we consider the geographic distance between sensors. See the first observation in Section 3.6.)

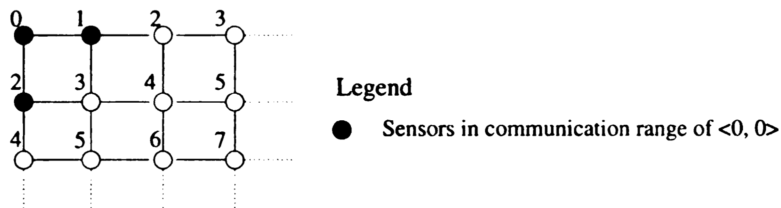


Figure 3.2: Sample diffusion in networks where a sensor communicates with its distance 1 neighbors. The number associated with a sensor shows the slot in which it should transmit.

From this figure, we observe that sensors $\langle 1, 0 \rangle$ and $\langle 0, 1 \rangle$ should not transmit at the same time as their messages will collide at sensor $\langle 1, 1 \rangle$. The following algorithm provides a collision-free diffusion in networks where sensors can communicate only with their distance 1 neighbors.

```
when sensor j receives a diffusion message from sensor k
      if (k is west neighbor at distance 1)
        transmit after 1 clock tick.
      else if (k is north neighbor at distance 1)
        transmit after 2 clock ticks.
      else // duplicate message received from east/south neighbor
        ignore
```

Figure 3.3: Version 1: Communicate 1, Interfere 1

Theorem 3.1 The above algorithm satisfies the problem specification of collision-free diffusion.

Proof. Let us assume that the source sensor $\langle 0,0\rangle$ starts transmitting at time $t=0$. By induction, we observe that sensor $\langle i,j\rangle$ will transmit at time $t=i+2j$. Now, we show that collisions will not occur in this algorithm.

Consider two sensors $\langle i_1,j_1\rangle$ and $\langle i_2,j_2\rangle$. Sensor $\langle i_1,j_1\rangle$ will transmit at time $t_1=i_1+2j_1$ and $\langle i_2,j_2\rangle$ will transmit at time $t_2=i_2+2j_2$. Collision is possible only if $i_1+2j_1=i_2+2j_2$, i.e., $(i_1-i_2)+2(j_1-j_2)=0$. Also a collision can occur only if the distance between $\langle i_1,j_1\rangle$ and $\langle i_2,j_2\rangle$ is at most 2, i.e., $|i_1-i_2|+|j_1-j_2|\leq 2$. Moreover, a collision occurs only if $\langle i_1,j_1\rangle$ and $\langle i_2,j_2\rangle$ are distinct, i.e., $|i_1-i_2|+|j_1-j_2|\geq 1$. Thus, the conditions for a collision are as follows:

- $(i_1-i_2)+2(j_1-j_2)=0$.
- $|i_1-i_2|+|j_1-j_2|\leq 2$.
- $|i_1-i_2|+|j_1-j_2|\geq 1$.

From the first condition, we conclude that $|i_1-i_2|$ is even. Combining this with the second condition, we have $|i_1-i_2|=0$ or $|j_1-j_2|=0$. However, if $|i_1-i_2|=0$

(respectively, $|j_1 - j_2| = 0$) then from the first condition $(j_1 - j_2)$ (respectively $(i_1 - i_2)$) must be zero. If both $(i_1 - i_2)$ and $(j_1 - j_2)$ are zero then the third condition is violated. Thus, collisions cannot occur in this algorithm. □

3.2 Version 2: Communicate x, Interfere x

Consider a grid network where a sensor can communicate with sensors that are distance $x, x \geq 1$ away (cf. Figure 3.4, where $x = 2$).

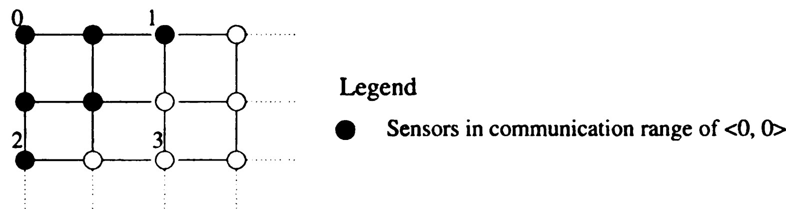


Figure 3.4: Sample diffusion in networks where a sensor communicates with its distance 2 neighbors. The number associated with a sensor shows the slot in which it should transmit.

From Figure 3.4, we observe that sensors $\langle 2, 0 \rangle$ and $\langle 0, 2 \rangle$ should not transmit at the same time as these messages will collide at $\langle 2, 2 \rangle$. Since the sensor $\langle 0, 0 \rangle$ can communicate at a larger distance and the sensors $\langle 0, 2 \rangle$, $\langle 2, 0 \rangle$ propagate the diffusion, sensors $\langle 0, 1 \rangle$, $\langle 1, 0 \rangle$ and $\langle 1, 1 \rangle$ need not transmit. The following algorithm provides a collision-free diffusion in networks where sensors can communicate with their distance $x, x \geq 1$, neighbors.

```
when sensor j receives a diffusion message from sensor k
      if (k is west neighbor at distance x)
        transmit after 1 clock tick.
      else if (k is north neighbor at distance x)
        transmit after 2 clock ticks.
      else //duplicate message from east/south neighbor or too close to source
        ignore
```

Figure 3.5: Version 2: Communicate x, Interfere x

Theorem 3.2 The above algorithm satisfies the problem specification of collision-free diffusion. $\square$

3.3 Version 3: Communicate 1, Interfere y

Consider a grid network where a sensor can communicate with sensors that are distance 1 away and interfere with sensors that are distance $y, y \geq 1$ away (cf. Figure 3.6, where $y = 2$).

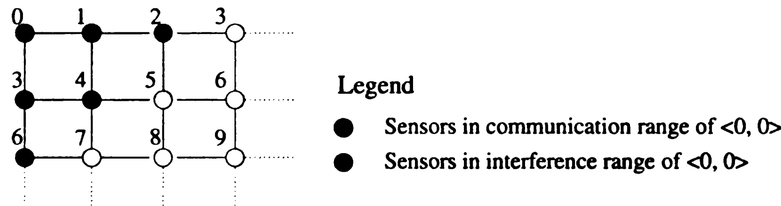


Figure 3.6: Sample diffusion in networks where a sensor communicates with its distance 1 neighbors and interferes with its distance 2 neighbors. The number associated with a sensor shows the slot in which it should transmit.

Once again, we observe that sensors $\langle 1, 0 \rangle$ and $\langle 0, 1 \rangle$ should not transmit at the same time. Also, sensors $\langle 2, 0 \rangle$ and $\langle 0, 1 \rangle$ should not transmit at the same time as

their messages will collide at $\langle 1,1\rangle$ and $\langle 2,1\rangle$. The third version of the algorithm is as follows:

```
when sensor j receives a diffusion message from sensor k
      if (k is west neighbor at distance 1)
        transmit after 1 clock tick.
      else if (k is north neighbor at distance 1)
        transmit after y + 1 clock ticks.
      else // duplicate message received from east/south neighbor
        ignore
```

Figure 3.7: Version 3: Communicate 1, Interfere y

Theorem 3.3 The above algorithm satisfies the problem specification of collision-free diffusion.

Proof. Let us assume that the source sensor $\langle 0,0\rangle$ starts transmitting at time $t=0$. By induction, we observe that sensor $\langle i,j\rangle$ will transmit at time $t = i+(y+1)j$. Now, we show that collisions will not occur in this algorithm. Consider two sensors $\langle i_1,j_1\rangle$ and $\langle i_2,j_2\rangle$. Sensor $\langle i_1,j_1\rangle$ will transmit at time $t_1 = i_1+(y+1)j_1$ and $\langle i_2,j_2\rangle$ will transmit at time $t_2 = i_2+(y+1)j_2$. Collision is possible only if the following conditions hold:

- $t_1 = t_2$, i.e., $(i_1-i_2)+(y+1)(j_1-j_2) = 0$.
- $|i_1-i_2|+|j_1-j_2| \le y+1$.
- $|i_1-i_2|+|j_1-j_2| \ge 1$.

From the first condition, we conclude that (i_1-i_2) is a multiple of $(y+1)$. Combining this with the second condition, we have $|i_1-i_2|=0$ or $|j_1-j_2|=0$. However,

if $|i_1 - i_2| = 0$ (respectively, $|j_1 - j_2| = 0$) then from the first condition $(j_1 - j_2)$ (respectively, $(i_1 - i_2)$) must be zero. If both $(i_1 - i_2)$ and $(j_1 - j_2)$ are zero then the third condition is violated. Thus, collisions cannot occur in this algorithm. □

3.4 Version 4: Communicate x, Interfere y

Consider a grid network where a sensor can communicate with sensors that are distance $x, x \geq 1$ away and interfere with sensors that are distance $y, y \geq x$ away. This network can be viewed as a modified network where the intermediate sensors are removed. Hence, in this modified network, a sensor can communicate with its distance 1 neighbors and interfere with its distance $\lceil \frac{y}{x} \rceil$ neighbors. Now, we apply version 3 of our algorithm with parameters *communicate* 1, *interfere* $\lceil \frac{y}{x} \rceil$.

3.5 Diffusion by an Arbitrary Sensor

If sensor k (other than, $\langle 0, 0 \rangle$) initiates the diffusion, we split the network into four quadrants with sensor k at the intersection of x and y axes. For each quadrant, we can use the algorithm similar to that in Sections 3.1-3.4; we simply need to ensure that messages in different quadrants do not collide (on x and y axes). For the case where communication range = interference range = 1, we can achieve this as follows: (Extensions for other values of communication and interference range are also similar.)

Sensors in the south-east quadrant transmit the diffusion message as before (i.e., a sensor $\langle i, j \rangle$ will transmit the diffusion at $|i| + 2|j|$). As shown in Figure 3.8, sensors in the north-east and south-west quadrants (including the $-$ve x-axis and $+$ve y-axis but excluding the $+$ve x-axis and $-$ve y-axis) transmit the diffusion similar to the south-east quadrant, but with 2 clock ticks delay. This is to ensure the diffusion messages do not collide at the x and y axes. Specifically, a sensor $\langle i, j \rangle$ in the north-east quadrant

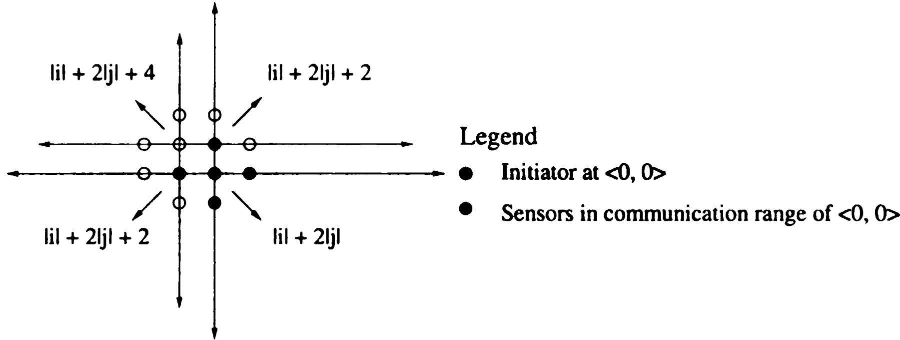


Figure 3.8: Sample diffusion in networks with arbitrary initiator

or in the south-west quadrant transmits the diffusion at $|i| + 2|j| + 2$. Sensors in the north-west quadrant (excluding the axes) transmits the diffusion similar to the other quadrants except that the delay here is 4 clock ticks (cf. Figure 3.8). In other words, a sensor $\langle i, j \rangle$ in the north-west quadrant transmits the diffusion at $|i| + 2|j| + 4$.

3.6 Observations about Our Algorithm

We make the following observations about our algorithm:

1. In Sections 3.1-3.5, we considered the Manhattan distance between sensors, i.e., if the interference range is y then we said that sensors $\langle i_1, j_1 \rangle$ and $\langle i_2, j_2 \rangle$ interfered with each other only if $|i_1 - i_2| + |j_1 - j_2| \leq y$. We note that our algorithm works correctly even if we say that sensors $\langle i_1, j_1 \rangle$ and $\langle i_2, j_2 \rangle$ interfere only if $|i_1 - i_2| \leq y$ and $|j_1 - j_2| \leq y$. It follows that our algorithm works correctly even if we consider the geographic distance between sensors and say that two sensors $\langle i_1, j_1 \rangle$ and $\langle i_2, j_2 \rangle$ interfere with each other if the geographic distance between them, $\sqrt{|i_1 - i_2|^2 + |j_1 - j_2|^2}$, is less than or equal to y.

2. Even if the interference range is overestimated, our algorithm works correctly.

3. Even if the communication range is underestimated, as long as it is at least 1, our algorithm works correctly.

4. We can apply our algorithm even if the communication and interference ranges of different sensors vary. We can use the minimum of the communication range of each sensor (underestimate) and the maximum of the interference range of each sensor (overestimate).

CHAPTER 4

Time Division Multiple Access (TDMA)

In this chapter, we present an algorithm for time-division multiple access (TDMA) in sensor networks using the collision-free diffusion algorithm discussed in Chapter 3. Time-division multiplexing is the problem of assigning time slots to each sensor. Two sensors j and k can transmit in the same time slot if j does not interfere with the communication of k and k does not interfere with the communication of j. In this context, we define the notion of *collision-group*. The *collision-group* of sensor j includes the sensors that are in the communication range of j and the sensors that interfere with the sensors in the communication range of j. Hence, if two sensors j and k are alloted the same time slot then j should not be present in the collision-group of k and k should not be present in the collision-group of j. Thus, the problem of time-division multiple access is defined as follows:

Problem Statement: Time Division Multiple Access
Assign time slots to each sensor such that,
If two sensors j and k transmit at the same time then
($j \notin$ collision-group of sensor k).

Figure 4.1: Time Division Multiple Access (TDMA)

Now, we present the algorithm for allotting time slots to the sensors. In Section 4.1, we present the algorithm for TDMA in perfect grids. In Section 4.2, we discuss how stabilization is achieved starting from an improperly initialized state.

4.1 Simple TDMA Algorithm

In this section, we present our simple TDMA algorithm that uses the third version of the diffusion algorithm (cf. Section 3.3) where communication range is 1 and interference range is y. Let j and k be two sensors such that j is in the collision group of k. Let t_j (respectively, t_k) be the slots in which j (respectively, k) transmits its diffusion message. We propose an algorithm where the slots assigned for j are $t_j + c * MCG$ where $c \geq 0$ and MCG captures information about the maximum collision group in the system.

From the correctness of the diffusion computation, we know that $t_j \neq t_k$. Now, future messages sent by j and k can collide if $t_j + c_1 * MCG = t_k + c_2 * MCG$, where $c_1, c_2 \geq 0$. In other words, future messages from j and k can collide iff $|t_j - t_k|$ is a multiple of MCG. More specifically, to ensure collision-freedom, it suffices that for any two sensors j and k such that j is in the collision group of k, MCG does not divide $|t_j - t_k|$. We can achieve this by choosing MCG to be $\max(\{\forall j : j \text{ is in the collision group of } k : |t_j - t_k|\}) + 1$.

In the third version of our algorithm, if j is in the collision group of k then $|t_j - t_k|$ is at most $(y+1)^2$; such a situation occurs if j is at distance of $y+1$ in north/south of k. Hence, the algorithm for time division multiplexing is as follows:

If sensor j transmits a diffusion message at time slot t_j, j can transmit at time slots, $\forall c : c \geq 0 : t_j + c * ((y+1)^2 + 1)$.

Figure 4.2: Simple TDMA Algorithm

The algorithm assigns time slots for each sensor based on the time at which it transmits the diffusion. Thus, a sensor (say, j) can transmit in slots: t_j, $t_j + ((y +$

$1)^2 + 1)$, $t_j + 2((y+1)^2 + 1)$, ..., etc. Figure 4.3 shows a sample allocation of slots to the sensors.

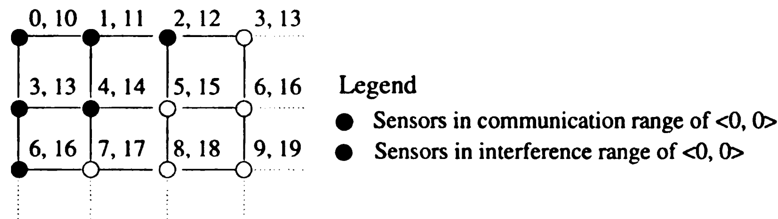


Figure 4.3: Sample TDMA in networks where a sensor communicates with its distance 1 neighbors and interferes with its distance 2 neighbors. The numbers associated with a sensor shows the slots in which it could transmit.

Theorem 4.1 The above algorithm satisfies the problem specification of TDMA. □

4.2 Stabilization of TDMA and Diffusion

We now add stabilization to the TDMA algorithm discussed in Section 4.1, i.e., if the network is initialized with arbitrary clock values (including the case where there is a phase offset among clocks), we ensure that it recovers to states from where collision-free communication is achieved. The simple TDMA algorithm relies on the collision-free diffusion algorithm discussed earlier (cf. Section 3.3). Whenever a sensor does not get the diffusion message for certain consecutive number of times, the sensor shuts down, i.e., it will not transmit any message until it receives a diffusion message. The network will eventually reach a state where the diffusion message can be received by all sensors. From then on, the sensors can use the TDMA algorithm to transmit messages across different sensors. Moreover, if there are no faults in the network and the links are reliable then no sensor will ever shut down.

Dealing with unreliable links. Now, we show that in the absence of faults, a sensor rarely shuts down even if the link between the neighboring sensors are unreliable. Let p be the probability that a sensor receives a message from its neighbor. Also, let n be the number of diffusion periods a sensor waits before shutting down. Now, consider a sensor j that receives a diffusion message after l intermediate transmissions. The probability that this sensor does not receive the diffusion message is $1 - p^l$ and the probability that this sensor shuts down in the absence of faults is $(1 - p^l)^n$. Note that this is an overestimate since a sensor receives the diffusion message from more than one sensor. If we consider $p = 0.90$, $l = 10$ and $n = 10$, the probability that the sensor j will incorrectly shut down is less than or equal to 0.0137.

Observations about our stabilizing fault-tolerant algorithms. We make the following observations about our stabilizing fault-tolerant algorithms:

1. If there are no failures in the network and the links are reliable then no sensor will ever shut down.

2. If there are no failures in the network and the links are unreliable then sensors may shut down rarely. However, the probability that a sensor shuts down incorrectly due to unreliable links can be made as small as possible.

4.3 Extensions: Other Topologies

In this section, we provide algorithms for collision-free diffusion and TDMA for other communication graphs. Specifically, in Section 4.3.1, we present the TDMA algorithm for hexagonal grids and, in Section 4.3.2, we present the TDMA algorithm for triangular grids. In Section 4.3.3, we show how collision-free diffusion is performed on a multi-dimensional rectangular grids. Finally, in Section 4.3.4, we provide algorithms for collision-free diffusion in arbitrary communication graphs.

4.3.1 TDMA Algorithm for Hexagonal Grids

Consider a hexagonal grid network where a sensor can communicate with its distance 1 neighbors and interfere with its distance 2 neighbors (cf. Figure 4.4). We assume that the initiator of the diffusion, which assigns the initial slots to the sensors, is located at the left-most corner on the left-top hexagon in the network (cf. Figure 4.4).

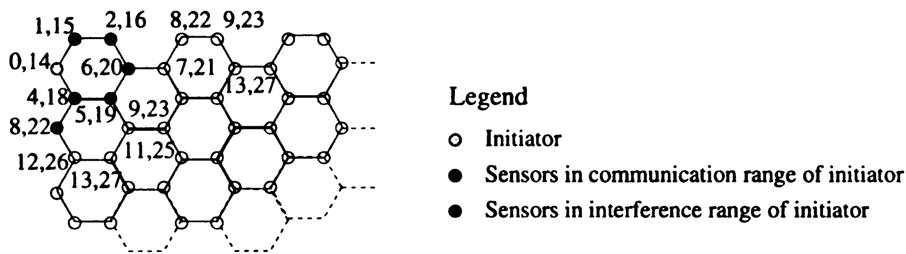


Figure 4.4: TDMA slot assignment in hexagonal-grid network where communication range=1 and interference range=2. The number associated with each sensor denotes the time at which it can send a message. Slots for some sensors are not shown.

From Figure 4.4, we observe that whenever the initiator transmits, sensors located at the top (say, j) and bottom (say, k) of the initiator at geometric distance 1 from the initiator can transmit next. However, if both these sensors transmit simultaneously then collision occurs at the initiator. Hence, we proceed as follows: whenever j receives the diffusion message from the initiator, it retransmits the message after 1 clock tick. Likewise, whenever k receives the diffusion message from the initiator, it retransmits the message after $2y$ clock ticks, where y is the interference range of the sensors. Further, whenever a sensor receives a message from its neighbor on the straight edge (cf. Figure 4.4), it forwards the message after 1 clock tick.

Once the initial slots are assigned, each sensor can determine future slots based on the time it forwards the diffusion message. In this context, we use the notion of collision-group. As discussed earlier in this chapter, the collision group of sensor j includes the sensors that can interfere with the communication of j. In this communication topology, the collision group of j includes the sensors that are within distance $y+1$ from j. Now, j can transmit again once the sensors in the collision group of j transmit the diffusion message. For a hexagonal grid, the period between successive slots, $P=2y(y+1)+\lfloor\frac{y}{2}\rfloor+1$ suffices. Thus, the TDMA algorithm for hexagonal grids is as follows:

```
const P=2y(y+1)+⌊y/2⌋+1;
// Initial slot assignment for hexagonal grids
when sensor j receives a diffusion message from k
    if (k is at distance 1 in the same level
    (i.e., j − k is a straight edge))
      transmit after 1 clock tick.
    else if (k is at distance 1 in the lower level)
      transmit after 1 clock tick.
    else if (k is at distance 1 in the upper level)
      transmit after 2y clock ticks.
    else // duplicate message
      ignore

// TDMA algorithm for hexagonal grids
If sensor j transmits a diffusion message at time slot t_j,
    j can transmit at time slots, ∀c : c ≥ 0 : t_j + c * P.
```

Figure 4.5: TDMA algorithm for hexagonal grids

Theorem 4.2 The TDMA algorithm for hexagonal grid guarantees collision-freedom. □

4.3.2 TDMA Algorithm for Triangular Grids

We now show how the TDMA algorithm for hexagonal grid network can be used in a triangular grid network. Consider a triangular grid network, where the communication range is 1 and interference range is 2 (cf. Figure 4.6).

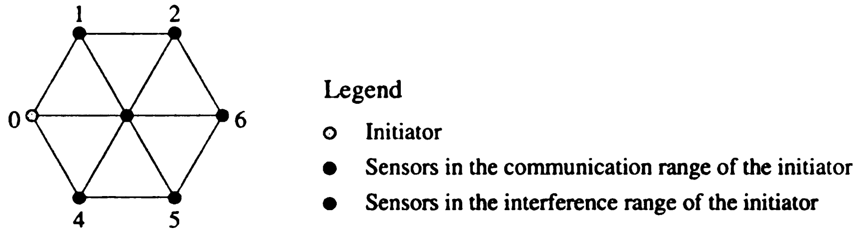


Figure 4.6: Initial slot assignment in triangular-grid network where communication range=1 and interference range=2. The number associated with each sensor denotes the time at which it forwards the diffusion message.

From Figure 4.6, we observe that it is possible to convert a triangular grid into a hexagonal grid. Once the hexagonal grid is obtained, the TDMA algorithm for the hexagonal grid network can be used to allot time slots to different sensors. In this algorithm, the *intermediate* sensors within the hexagons (cf. Figure 4.6) will not get time slots. However, we can allow the sensors in the boundary of the hexagon (called *boundary* sensors) to share their slots with the intermediate sensors. In order to allow boundary sensors to share their TDMA slots with the intermediate sensors in the grid, we need to increase the collision group size. Specifically, collision group of a sensor, say j, should include the sensors that are within distance $y + 2$. This ensures that the communication of the intermediate sensors in the slots assigned to the boundary sensors does not collide.

The sharing scheme can be designed based on the application requirements and fairness issues. Since each boundary sensor appears in 3 hexagons, a simple sharing scheme is as follows: for every 3 TDMA slots assigned to boundary sensor j, j allows the intermediate sensor in one of 3 hexagons to transmit alternatively. Note that the slot assignments to the intermediate sensors can be communicated using the TDMA slots of the boundary sensors.

4.3.3 Diffusion in Multi-Dimensional Grids

Consider a multi-dimensional grid where each sensor has n neighbors, one in each dimension. Figure 4.7 illustrates collision-free diffusion in a cube (3-dimensional grid) where each sensor can communicate with its distance 1 neighbors. (Similar extension is also possible for other values of communication range and interference range.)

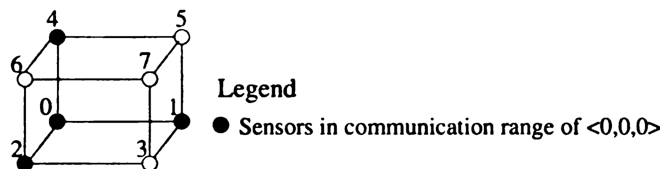


Figure 4.7: Three-dimensional grid network where a sensor communicates with its distance 1 neighbors in its x, y and z axes. The number associated with a sensor shows the slot in which it should transmit.

In two-dimensional grid topology, diffusion is done such that the transmissions in each dimension do not interfere or collide with each other. Our algorithm for collision-free diffusion in multi-dimensional grids is similar; we use the algorithm in Section 3 for each plane while ensuring that the messages from two planes do not collide. Specifically, the algorithm for n-dimensional grid is as follows:

Let sensor k be a neighbor (in the direction towards the initiator) of sensor j in its i-th dimension.
When sensor j receives a diffusion message from sensor k
    j transmits after 2^i clock ticks.

Figure 4.8: Diffusion in n-D grids

Theorem 4.3 The above algorithm satisfies the problem specification of collision-free diffusion.

Proof. The above algorithm provides collision-free diffusion in each plane. Let us consider the different planes in a multi-dimensional grid. The algorithm for a given plane is similar to the algorithm in Section 3, which is collision free. The algorithm is collision-free across different planes since the diffusion is separated by a sufficient time delay. □

4.3.4 Diffusion in Arbitrary Graphs

In this section, we extend our algorithm for collision-free diffusion in sensor networks where the underlying graph is not a grid. Specifically, we consider arbitrary graphs and apply our algorithms by embedding a grid in it.

Embedding a grid in arbitrary graphs. Collision-free diffusion in other graphs can be achieved by embedding a (partial) grid in that graph. (Note that the goal of this solution is to show the feasibility of such extension. If additional information about the topology is available then it is possible to improve the efficiency of the diffusion on the transformed graph.) To show one approach for embedding such a partial grid, we begin with the observation that, an arbitrary tree can be mapped into a (complete) binary tree. Also, a complete binary tree can be mapped on a 2-dimensional grid with *dilation* (i.e., longest path to which any edge of the original

graph is mapped) $\lceil (k-1)/2 \rceil$ where k is the depth of the tree [7]. If the degree of a node is more than 3, we split that node to construct a binary tree (cf. Figure 4.9).

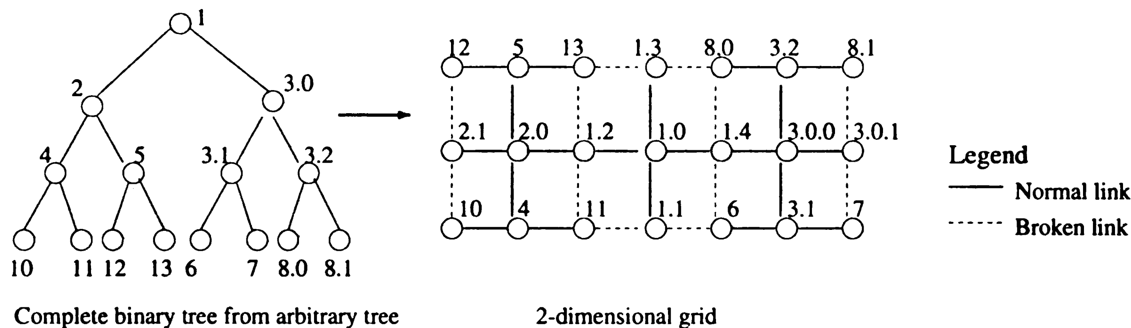


Figure 4.9: Mapping an arbitrary tree into a 2-dimensional grid

We can observe from Figure 4.9 that node 1 is split into 5 nodes, 1.0...1.4. Hence, node 1 will get 5 different time slots for communication. Also, nodes in the 2-dimensional grid can communicate with the nodes that are at distance 1 (except in the case where the link drawn is a broken link). Some of the nodes in the 2-dimensional grid can communicate with nodes that are at a larger distance. For the purpose of collision-free diffusion, we can treat this communication as interference (e.g., in Figure 4.9, the communication between 1.1 and 3.1 can be treated as interference). It follows that given an arbitrary tree, we can embed a partial grid in it; in this partial grid, the communication range is 1. And, the interference range is determined based on the way in which nodes of degree more than 3 are split.

Finally, for an arbitrary graph, we can use its spanning tree, and embed a partial grid in it. Then, we can add the remaining edges to this partial grid and treat them as interference-only. With such an approach, it is possible to apply the collision-free diffusion algorithm to arbitrary graphs.

4.4 Extensions: Dealing with Failed/Sleeping Sensors

In this section, we discuss extensions that remove some of the assumptions made in Section 2. In Section 4.4.1, we extend the algorithm to deal with the case where sensors are subject to fail-stop faults. Then, in Section 4.4.2, we extend our TDMA algorithm to deal with failed sensors. While the solutions presented in Sections 4.4.1 and 4.4.2 are discussed with respect to rectangular grids, we note that it can also be applied in other grid-based topologies.

4.4.1 Diffusion in Imperfect Grids or Grids with Failed Sensors/Links

In this section, we consider the case where sensors can fail, links between sensors can fail, or the grid can be improperly configured (with some sensors missing).

Based on the extension in Section 3.5, without loss of generality, assume that the left-top sensor initiates the diffusion. In the absence of failure of sensors or links between them, the sensors receive the diffusion messages from their north or west neighbors before receiving duplicate messages from their east or south neighbors. Hence, if a sensor receives the diffusion message for the first time from its east or south neighbor, it can conclude that some of the sensors in the network are missing or failed. When a sensor receives such a message from the south/east neighbor, it updates its clock based on the time information in the diffusion message. Based on its geographic location, it then determines the slot in which it would have transmitted the diffusion if no sensor had failed. Finally, it uses the TDMA algorithm (cf. Section 4.1) to find the next slot when it can transmit a message; this slot would be used for retransmitting the diffusion message. Thus, the algorithm for collision-free diffusion in the presence of failed/sleeping sensors is as follows:

```
when sensor j receives a diffusion message for the first time from sensor k
    update local clock;
    determine the ideal diffusion slot;
    find the TDMA slots using the algorithm in Section 4.1;
    transmit in ideal diffusion slot or next TDMA slot, whichever is earlier;
```

Figure 4.10: Diffusion in presence of failed/sleeping sensors

Remark. We can embed a multi-dimensional grid in a non-planar graph using the above algorithm by considering the thickness [8] of the given graph. With this embedding, we can obtain collision-free diffusion and time-division multiplexing by combining the above algorithm and the algorithm for multi-dimensional grid (cf. Section 4.3.3).

4.4.2 Extending TDMA for Dealing with Failed Sensors

In this section, we focus on providing TDMA service in the presence of failed or sleeping sensors. We assume that the initiator does not fail and that the network remains connected.

In the TDMA algorithm discussed in this chapter, a sensor normally receives the diffusion message for the first time from a sensor that is closer to the base station. In presence of failed/sleeping sensors, a sensor may receive the diffusion message for the first time from the sensor that is (physically) farther away from the base station. The TDMA algorithm in Section 4.1 ignore such message as duplicate. However, in presence of failed/sleeping sensors, a sensor should forward such a diffusion message (cf. Section 4.4.1). This ensures that the diffusion message reaches all the active sensors.

This modification, however, also assigns slots to failed/sleeping sensors. To reassign the slots assigned to failed/sleeping sensors, we consider the notion of collision

group. Recall that the collision group of sensor j includes the sensors that interfere with the communication of j. To improve the bandwidth utilization, we consider the problem of reducing the collision-group size to deal with the sensors that are sleeping or have failed. Our solution involves three tasks: (1) allowing each sensor to determine its collision-group, (2) computing the maximum collision group size (MCG) in the network, and (3) communicating the MCG to all sensors

Determining collision group of each sensor. Regarding the first part, if a sensor, say j, plans to be inactive for a long time, it should inform the sensors in its collision-group before it becomes inactive. This can be achieved as follows: When j wants to become inactive, it informs its neighbors (in a slot allotted by the TDMA algorithm). These neighbors, in turn, inform their neighbors until the information reaches all sensors in the collision-group of j. Alternatively, if j fails (or becomes inactive without informing its neighbors), its neighbors can detect this fact by observing that no communication was received in the slot allotted to j. This information, in turn, will be communicated to the sensors in the collision-group of j. A sensor, say k, updates its collision-group to max(collision-group of k, $\{\forall i : i$ is in the collision of $k : |t_i - t_k|\}$) where t_i (respectively, t_k) is the time slot at which sensor i (respectively, k) transmits a diffusion message.

Computing the MCG. Regarding the second part, we use the initial slot assignment algorithm (cf. Section 3.3) where the communication range is 1 and the interference is greater than or equal to 1. Once again, for simplicity, we assume that left-top sensor ($\langle 0, 0\rangle$) communicates the size of the collision-group when it initiates the diffusion. Whenever a sensor, say j, propagates the diffusion, it sets the collision-group to max (collision-group included in a message that was received by j, collision-group of j). It follows that the sensor in the right-bottom corner will be able

to obtain the MCG in the network. This MCG can then be communicated to the left-top sensor using the current collision-free TDMA algorithm. If the right-bottom sensor has failed, this responsibility can be delegated to other sensors.

Communicating the MCG. Finally, regarding the third part, once the left-top sensor learns the new collision-group, it can include this when it initiates the next diffusion. This diffusion will allow the sensors to learn the size of the new collision-group that will then be used by the TDMA algorithm.

If the size of the collision-group for a sensor, say j, is less than MCG then the bandwidth in the neighborhood of j is underutilized. There are several approaches to deal with this difficulty. For one, j can choose to use the unassigned slots at the risk of causing collision. For two, j can use the collision-free slot allotted to it to request access to future slots that are unassigned.

We would like to note that the above description is intended to show that it is possible to change the size of the collision-groups to ensure optimal bandwidth utilization. The parameters involved in changing the collision-group are the frequency with which sensors update their collision-group and the frequency with which the sensor(s) at the right-bottom communicate the group change information. Also, it is possible to accelerate the change using the distributed reset [9,10].

CHAPTER 5

TDMA Service for Sensor Networks

In this chapter, we present the TDMA service for sensor networks for different communication patterns. Our TDMA service lets one customize the assignment of time slots to different sensors by considering the commonly occurring communication patterns in sensor network applications. Specifically, we consider three commonly occurring communication patterns: *broadcast*, *convergecast*, and *local gossip*. In broadcast, a message is sent to all the sensors in the network. Broadcast is useful when a base station wants to transmit some information (e.g., program capsules for reprogramming the sensors [11]) to all the sensors in the network. We also consider two other communication patterns, convergecast and local gossip. These algorithms are based on our experience with *Line in the Sand* demonstration [12]. In this demonstration, the sensors are arranged in a thick line (grid). When an intruder crosses this line, the sensors detect it. Now, to classify the intruder, the sensors that observed the intruder communicate with each other. We consider two ways of classification, internal and external. In an internal classification, the sensors that detect the intruder communicate with each other locally. Based on this motivation, we consider the communication pattern, local gossip, where a sensor sends a message to its neighboring sensors within some distance. In an external classification, the sensors send their data to the base station that exfiltrates the data outside the sensor network. Based on this motivation, we consider the communication pattern, convergecast, where a group of sensors send a message to a particular sensor.

We present the TDMA algorithms customized for different communication patterns in Section 5.1 and their implementation in Section 5.2. Then, we introduce the simulation model in Section 5.3 and present simulation results in Section 5.4.

5.1 Algorithms for TDMA Service

In this section, we present the algorithms for TDMA service for different communication patterns. In Section 5.1.1, we discuss the algorithm for broadcast. Then, in Section 5.1.2, we discuss the algorithm for convergecast and subsequently, in Section 5.1.3, we discuss the algorithm for local gossip.

5.1.1 TDMA Service for Broadcast

The TDMA algorithm in Section 4.1 is more suitable for broadcast where the base station sends some data to all sensors in the network. To see this, observe that, when sensor j transmits the broadcast data, the sensors that receive that data (in the southeast quadrant of j) can transmit it immediately; slots assigned to the sensors in the southeast quadrant of j are just *after* the slots assigned to j (cf. Figure 5.1).

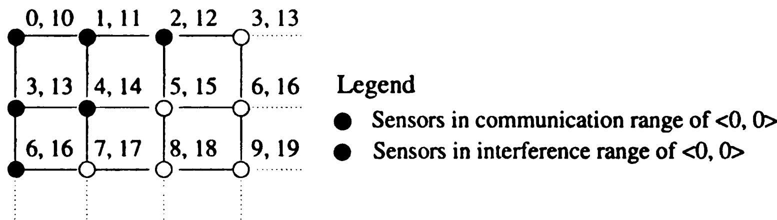


Figure 5.1: Sample TDMA in networks where a sensor communicates with its distance 1 neighbors and interferes with its distance 2 neighbors. The numbers associated with a sensor shows the slots in which it could transmit.

5.1.2 TDMA Service for Convergecast

The algorithm in Chapter 4 introduces a significant delay for convergecast, where a group of sensors send data (for example, information about the activities of an intruder in the field [12]) to the base station. Hence, in this section, we present the TDMA algorithm customized for convergecast.

To reduce the delay for convergecast, we change the slot assignment as follows: If j receives a message from its left neighbor then it chooses to transmit the diffusion in $(-1)^{th}$ slot (in circular sense). In other words, j transmits in the $(P-1)^{th}$ slot, where P $(=(y+1)^2+1)$ is the interval between slots assigned to a sensor and y is the interference range of the sensors. If j receives a message from its top neighbor then it transmits in the $(-(y+1))^{th}$ slot. (For example, see Figure 5.2 for slot assignment for the case where $y=2$.) After the first slot is determined, the sensors can then transmit once in every P slots.

Figure 5.2: Sample diffusion for convergecast where communication range=1, interference range=2. The number associated with each sensor denotes the time at which it forwards the diffusion message.

As we can see from Figure 5.4, with the above slot assignment, delay for convergecast is reduced. Specifically, when a sensor transmits a message that is to be relayed by sensors closer to the base station (left-top sensor), such a relay introduces

only a small (respectively, no) delay. Thus, the TDMA algorithm customized for convergecast is as follows:

const $P = (y+1)^2 + 1$;
// Initial slot assignment for convergecast
When sensor j receives a diffusion message from k
    if (k is west neighbor at distance 1)
        transmit in the $P + (-1)^{th}$ slot.
    else if (k is north neighbor at distance 1)
        transmit in the $P + (-(y+1))^{th}$ slot.
    else // duplicate message
        ignore

// TDMA algorithm for convergecast
If sensor j transmits a diffusion message at time slot t_j,
    j can transmit at time slots, $\forall c : c \geq 0 : t_j + c * P$.

Figure 5.3: TDMA algorithm for convergecast

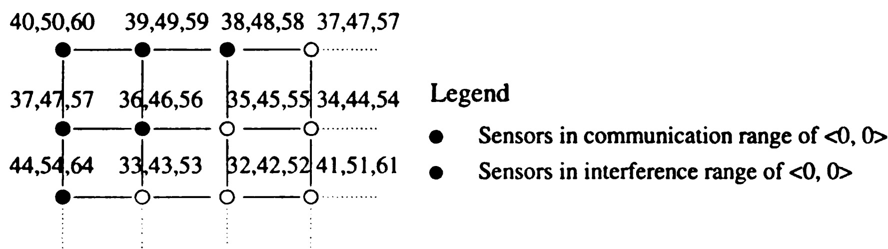


Figure 5.4: TDMA slot assignment for convergecast where communication range=1, interference range=2. The number associated with each sensor denotes the time at which it can send a message. Some initial slots are not shown.

Theorem 5.1 The above algorithm satisfies the problem specification of TDMA. □

Remark. We note that the above algorithm is designed for a rectangular grid. We can also customize the TDMA service on a hexagonal/triangular grid for convergecast.

Towards this end, we need to change the initial slot assignments and the TDMA period P similar to the modifications discussed for convergecast on rectangular grids. Specifically, for hexagonal/triangular grid, whenever sensor j receives the diffusion message, it forwards the message in its negative slot (i.e., it forwards the message in $(P-t_j)^{th}$ slot, where t_j is the time at which it is supposed to forward according to the original algorithm).

5.1.3 TDMA Service for Local Gossip

For local gossip, the communication is in all directions. Hence, we need an approach that combines the slot assignment for broadcast and convergecast. We proceed as follows: We increase the value of the period (P) to $2((y+1)^2+1)$, twice the previous value. With this increased value, each sensor gets two slots (even and odd) in this period. Let the slots assigned to the initiator be 0 and $P-1$. To simplify the presentation, let us assume that the initiator starts a diffusion in its even or the 0th slot. When j receives the diffusion from its left neighbor, it chooses the slot that is 2 higher than that used by the left neighbor. Likewise, when j receives the diffusion from its top neighbor, it chooses the slot that is $2(y+1)$ higher than that used by the top neighbor. (For example, see Figure 5.6 for slot assignment for the case where $y=2$.) Note that the diffusion messages are forwarded in the even slots. In our solution for gossip, whenever sensor k transmits in the even slot, say t_k, it can also transmit in $((P-1)-t_k)$ *mod* P, the odd slot. Thus, the TDMA algorithm customized for local gossip is as follows:

const $P = 2((y+1)^2+1)$;
// Initial slot assignment for local gossip
When sensor j receives a diffusion message from k
    if (k is west neighbor at distance 1)
        transmit after 2 clock ticks.
    else if (k is north neighbor at distance 1)
        transmit after $2(y+1)$ clock ticks.
    else // duplicate message
        ignore

// TDMA algorithm for local gossip
If sensor j transmits a diffusion message at time slot t_j,
    j can transmit at time slots,
        $\forall c : c \geq 0 : t_j + c * P$,
            $(((P-1)-t_j) \ mod \ P) + c * P$.

Figure 5.5: TDMA algorithm for local gossip

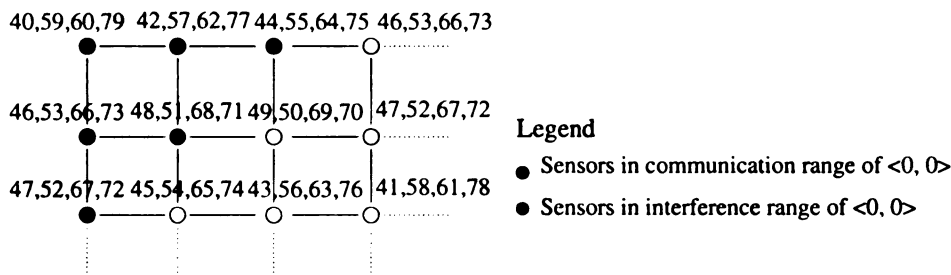


Figure 5.6: TDMA slot assignment for gossip where communication range=1, interference range=2. The number associated with each sensor denotes the time at which it can send a message. Some initial slots are not shown.

Theorem 5.2 The above algorithm satisfies the problem specification of TDMA. □

Based on Figure 5.6, in the case where TDMA is customized for local gossip, the interval between two successive slots of a sensor can be twice as much as in the case where TDMA is customized for broadcast/convergecast. Thus, if a sensor needs to transmit a message then the worst case delay is larger when the TDMA service is

customized for local gossip. In spite of this deficiency, the TDMA service provides substantial benefits for broadcast and convergecast even if it is customized for local gossip. To see this, observe that once the base station sends the broadcast message in its even slot, any sensor receiving it can forward it with a small delay (cf. Figure 5.6). Likewise, if a sensor transmits a convergecast message in the odd slot, any sensor receiving it can forward it with a small delay. In fact, as seen from Figure 5.6, in the TDMA service customized for local gossip, if a sensor wants to transmit a message in any given direction (east, west, north, south, southeast, southwest, northeast, or northwest) then the sensor that receives that message can forward it with a small delay.

Based on the above discussion, if the most common communication pattern is known to be broadcast or convergecast, we can customize the TDMA service accordingly. Even if the communication pattern is unknown or varies with time, customizing the TDMA service for local gossip provides significant benefits for other communication patterns.

Remark. We note that the above algorithm is designed for a rectangular grid. We can also customize the TDMA service on a hexagonal/triangular grid for local gossip. Towards this end, we need to change the initial slot assignments and the TDMA period P similar to the modifications discussed for local gossip on rectangular grids.

5.2 Implementation of TDMA Service

The TDMA service includes APIs for initialization, send and receive. We discuss each of these APIs and their internal details, next.

Initialization. As discussed in Section 5.1, one of the parameters to the service is the interference range used by sensors. We assume that the interference range of

all sensors is identical. For initialization, the TDMA service assumes that once the sensor network is deployed, there is a delay before the application begins. This delay is used to perform a diffusing computation and to assign initial slots. Additionally, the diffusion is performed periodically to (re) validate the slots and to deal with clock drift among sensors. Thus, one of the parameters to the service is the period after which diffusion is used to (re) validate the slots that sensors need to use for TDMA.

Yet another parameter for TDMA service is the time slot (in physical time) that should be assigned for sending a message. Until now, we made a simplifying assumption that it takes one unit of time to send a message. The time slot parameter for TDMA service determines the length of this unit. Hence, we choose the slot time so that it is larger than the time (including preamble, CRC, etc.) required to send a message of maximum length.

The TDMA service also takes the parameter that identifies the communication pattern for which the service should be customized. The application can use this parameter to customize the communication that occurs most frequently. As discussed in Section 5.1, if the commonly occurring communication pattern is not known then customizing TDMA for local gossip is beneficial.

Send. Although TDMA service ensures that when two sensors, say j and k, transmit simultaneously, neighbors of j (respectively, k) receive messages from j (respectively, k) without collision, we still use CSMA. Thus, if j is about to transmit in its TDMA slot and it observes that the channel is busy then j backs off until the next TDMA slot. Although in our simulations and in experiments with small number of motes such a back off never occurred, it is expected that it may occur in a larger experimental setup. We expect that using CSMA in addition to TDMA will reduce the collisions that may occur due to unsynchronized clocks, larger interference range

than that used by the TDMA algorithm or interference range that varies with time or other environment factors.

The send is non-blocking. Hence, if the TDMA service receives more than two messages and the sum of their lengths is less than the maximum message length, we combine those messages and send them in the next time slot.

Receive. There are no special tasks performed when TDMA service receives a message. All received messages are forwarded to upper layer. Additionally, if the received message includes multiple embedded messages then the receive action separates them.

Figure 5.7 shows the partial nesC [13,14] code of the TDMA component designed in this chapter. Specifically, Figure 5.7 shows the application programming interfaces (APIs) of the TDMA service, structure of a TDMA message and the wiring logic. The TDMA component *TDMAC* is wired with the implementation *TDMAM* of the TDMA service and other required TinyOS components.

5.3 Simulation Model

In this section, we discuss the simulation model of the experiments. We use a probabilistic wireless network simulator, *prowler* [15] which is a simulation environment for embedded systems especially for MICA motes [3,6]. The simulator has a modular design. Each layer of the system architecture is designed as a separate module.

Using prowler, one can prototype different sensor network applications, communication models, propagation models and topology. For our TDMA simulations, we use the radio/communication model that is based on the algorithms in Section 5.1. To compare our algorithms with the existing implementation, we use the default radio

```
typedef struct TDMA_Msg {
                uint8_t type; // Diffusion or TDMA message
                uint8_t senderID;
                uint8_t size;
                uint8_t seqNo;
                char data[TDMA_MESSAGE_LENGTH];
} TDMA_Msg;
typedef TDMA_Msg * TDMA_MsgPtr;

interface TDMA {
        command result_t init(uint8_t ID, uint8_t baseStationID,
                uint8_t gridX, uint8_t gridY, uint8_t interferenceRange);
        command result_t send(uint8_t messageSize, TDMA_MsgPtr data);
        event result_t sendDone(TDMA_MsgPtr sent, result_t success);
        event result_t receive(TDMA_MsgPtr data);
}

configuration TDMAC {
                provides { interface TDMA; interface StdControl; }
}
implementation {
                components TDMAM, GenericComm as TComm, ...;
                TDMA = TDMAM;
                StdControl = TDMAM;
                ...
}
```

Figure 5.7: Sample data structure, interface, and component wiring definitions in nesC

models provided by prowler. These models include CSMA and a primitive model that uses no access protocol. Finally, we use the rectangular grid as the underlying topology since it reflects the topology used in [12]. Specifically, in [12], the sensors are arranged in a rectangular grid and the base station is placed at one corner in this grid.

Now, we discuss the simulations we performed in the context of these communication patterns. Then, we discuss the simulations we performed to study the effect of location errors.

Broadcast. The base station (sensor at left-top corner) initiates a broadcast. It sends the broadcast message to its neighbors in the communication range. Whenever a sensor receives the broadcast message for the first time, it relays it (for sensors farther from the base station). We conduct the broadcast simulations for different network sizes. In these simulations, we consider the following metrics: maximum delay incurred in receiving the broadcast message, number of sensors that receive the broadcast message, and number of collisions. Since CSMA (respectively, no MAC layer) does not guarantee reception by all sensors, we also consider the delay when a certain percentage of sensors receive the broadcast message. Regarding collisions, we compute the ratio of the number of collisions to the number of messages. Note that this ratio can be greater than 1 as one message can potentially collide at several sensors.

Convergecast. For convergecast, a set of sensors send a message to the base station (approximately) at the same time. In our experiments, we keep the network size fixed at 10x10. We choose a subgrid of varying size; sensors in this subgrid transmit the data to the base station. We make the worst case assumption in these experiments and assume that the subgrid that sends the data to the base station is in the opposite corner from the base station, i.e., the subgrid is farthest from the base station. For these simulations, we compute maximum delay incurred for receiving messages at the base station, the percentage of sensors whose messages are received by the base station and the number of collisions. Once again, as in broadcast simulations, we

compute the delay for the case where a certain percentage of messages are received by the base station.

Local gossip. In local gossip, a subgrid of nodes send the data. The goal is to transmit the data from these sensors to the sensors in the subgrid and the neighbors of the sensors in the subgrid. Thus, local gossip is applicable in *locally* determining the set of sensors that observed a particular event. In our experiments, we keep the network size fixed at 10x10. And, we choose different sizes of subgrids; sensors in this subgrid transmit the gossip messages. For these simulations, we compute the average delay incurred for receiving messages at the nodes that are expected to receive the local gossip and number of collisions.

Location errors. One of the important concern in any communication protocol for sensor networks is the errors in sensor location. Errors are introduced in sensor location due to misplacement of sensors or external factors like wind, vehicle movement, etc. Communication protocols that depend on sensor location should be able to tolerate such errors.

To model location errors, we randomly perturbed the sensors. In our simulation, the error in sensor location is determined using a normal distribution, $N(\mu, \sigma)$, where μ is the mean error distance and σ is the standard deviation of the error. The direction of perturbation was randomly selected from 0 degrees to 360 degrees. To ensure that the grid remains connected in spite of perturbations, during these simulations we increased the communication range. We note that this is a reasonable assumption in that if we need to tolerate location errors then the distance between two neighboring sensors should be less than or equal to the communication range.

5.4 Simulation Results

In this section, we present our simulation results that compare our TDMA algorithm from Section 5.1 with the case where CSMA is used and with the case where no MAC layer is used. We first present our results for broadcast. Then, we consider convergecast and local gossip. Finally, we consider the issue location errors. Based on the values used in [12], in convergecast and local gossip, the TDMA service groups up to 4 messages in the queue into a single message before transmitting. In Section 5.4.5, we discuss the effect of grouping on the performance of the TDMA algorithms.

5.4.1 Broadcast

In Figure 5.8, we present our simulation results for broadcast for the case where communication range and interference range is 1. Figure 5.8(a) identifies the number of collisions that occur in different algorithms. As expected, TDMA is collision free for all network sizes. By contrast, in CSMA, about 10% of messages suffer from collisions. However, if no MAC layer is used then the number of collisions is more than the number of messages sent. This is due to the fact that one transmitted message often collides at more than one sensor.

Figure 5.8(b) identifies the maximum delay incurred in receiving broadcast messages. Since all sensors may not receive the broadcast message when CSMA is used, we consider the delay when a certain percentage, 80-100%, of sensors receive the broadcast message. As we can see, the delay in TDMA based schemes is only slightly higher.

Figure 5.8(c) identifies the number of sensors that receive the broadcast message. We find that with CSMA/TDMA, all sensors receive the message. However, without the MAC layer, the number of sensors that receive the message is less than 50%.

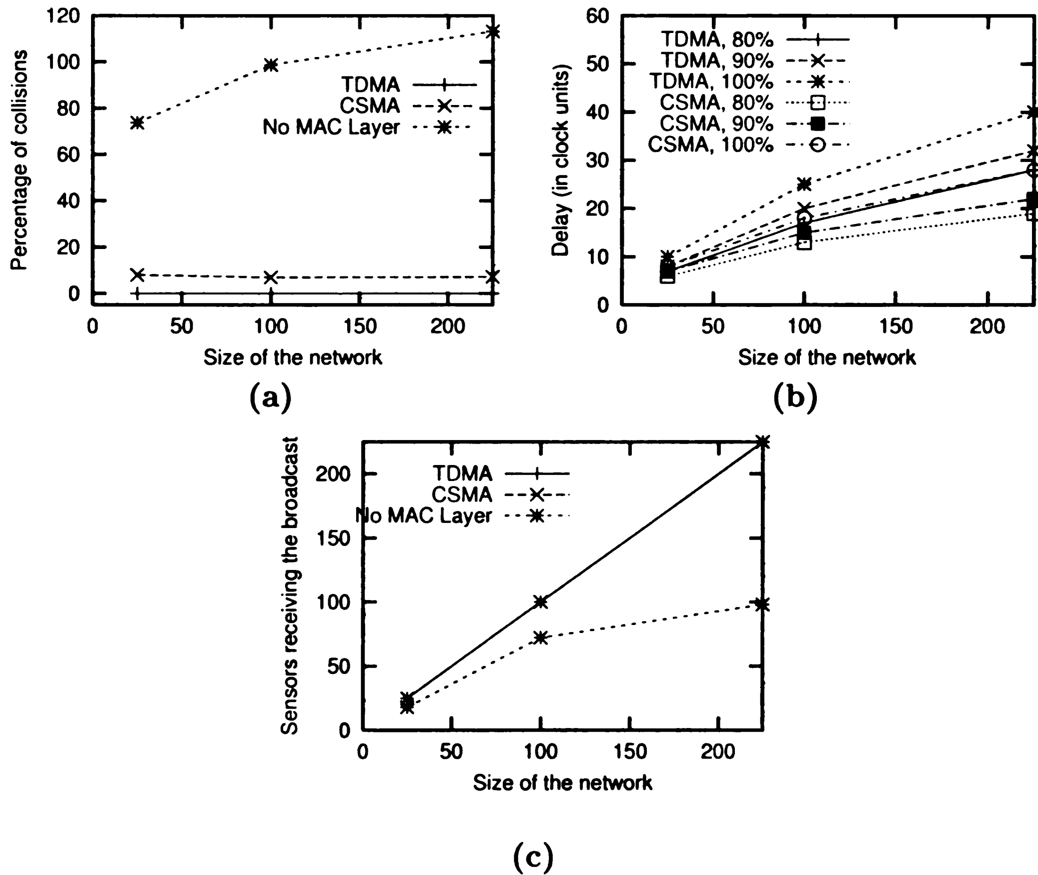


Figure 5.8: Results for broadcast with communication range = 1, interference range = 1

In Figure 5.9, we present the simulation results for broadcast for the case where communication range is 1 and interference range is 2. As we can see, these results are similar to those in Figure 5.8.

5.4.2 Convergecast

In Figure 5.10, we present our simulation results for convergecast for the case where communication range and interference range is 1. Figure 5.10(a) identifies the number of collisions that occur in different algorithms. As we can see from Figure

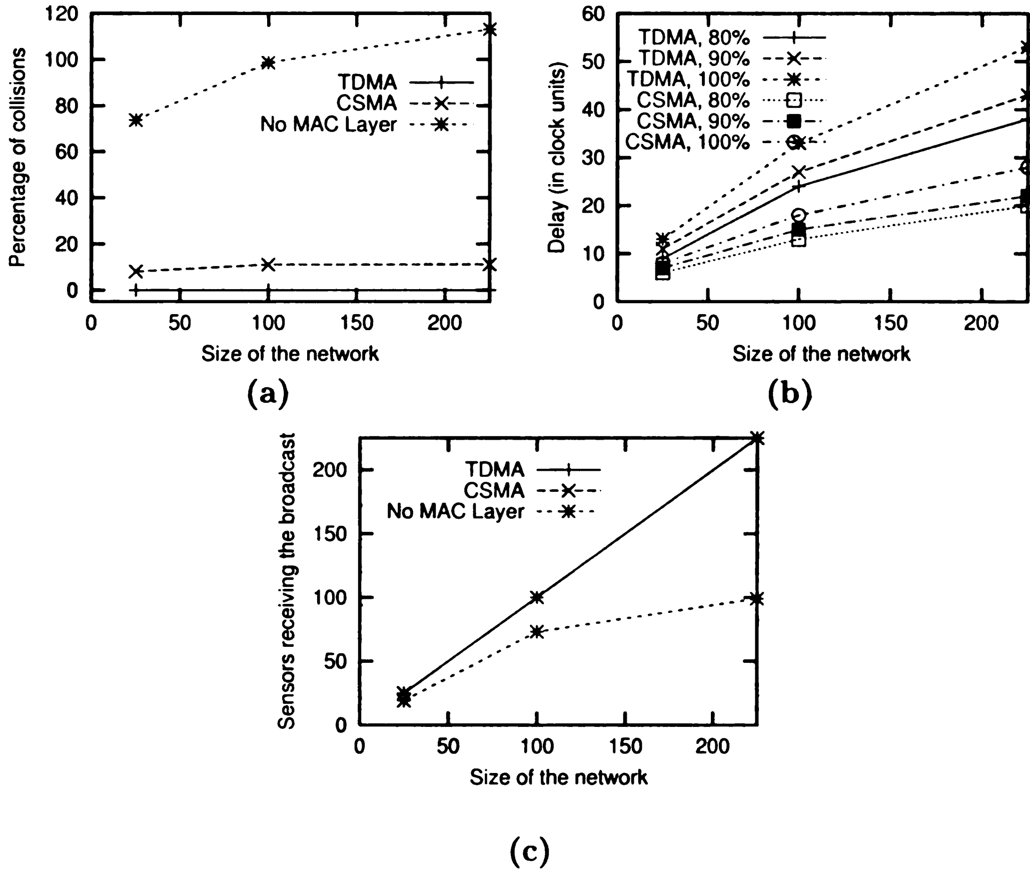


Figure 5.9: Results for broadcast with communication range = 1, interference range = 2

5.10(a), although the TDMA based solution is collision free, there are a significant number of collisions with CSMA. Regarding delay, as we can see from Figures 5.10(b) and (c), the delay incurred by TDMA is reasonable and that the base station receives all the messages sent by the sensors. By contrast, with CSMA, approximately 50% of the messages are received when the number of sensors sending the data to the base station increases.

We note that the number of collisions without the MAC layer is significantly more than that for CSMA/TDMA. The number of collisions decrease as the size of the field

sending data increases. This anomaly occurs due to the fact that when the number of sensors that start the convergecast is more, many of their messages fail on the first few links. Effectively, this reduces the number of collisions as collisions occur only on initial links. In fact, as we can see from Figure 5.10(c), without MAC layer, no data reaches the base station.

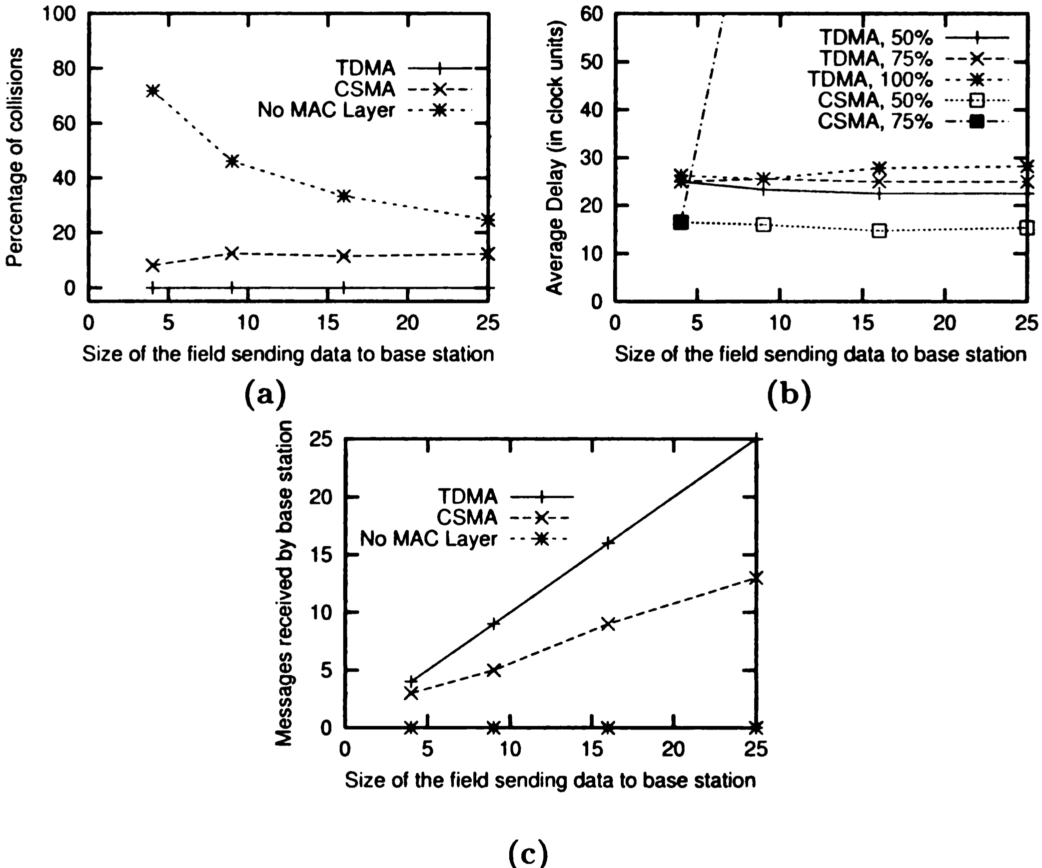


Figure 5.10: Results for convergecast with communication range = 1, interference range = 1

In Figure 5.11, we present the simulation results for convergecast for the case where communication range is 1 and interference range is 2. As we can see, these results are similar to those in Figure 5.10.

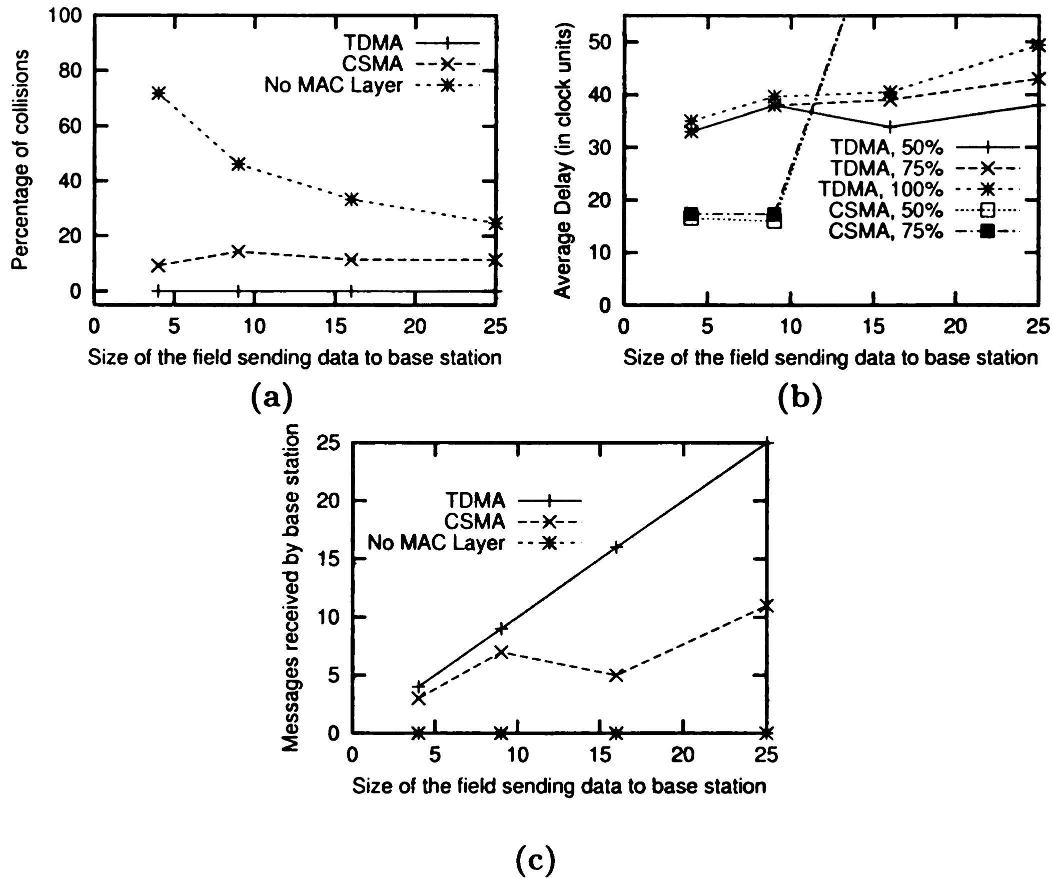


Figure 5.11: Results for convergecast with communication range = 1, interference range = 2

5.4.3 Local gossip

In Figures 5.12(a) and 5.12(b), we present our simulation results for local gossip for the case where communication range is 1 and interference range is 1. Figure

5.12(a) identifies the number of collisions as the size of the group performing local gossip increases. As we can see, CSMA based solution suffers significant collisions whereas TDMA based solution is collision free. Note that the percentage of collisions decrease as the size of the group performing local gossip increases. As discussed in the case of convergecast, this is due to the fact that with increased communication, many messages create collisions in the initial part of the network and then they are lost. Also, as seen from Figure 5.12(b), the delay in TDMA is somewhat more than that in CSMA. However, unlike TDMA where all sensors receive the necessary messages, in CSMA, the sensors receive approximately 50% of messages.

Once again, the results are similar for the case where interference range is increased to 2 (cf. Figures 5.12(c) and 5.12(d)).

5.4.4 Effect of Location Errors

In our location error experiments, even if the sensors are perturbed from their ideal position, as long as the perturbation is small and the communication range is increased so that the network remains connected, the results are close to the those presented earlier. We introduce location errors in the sensors as follows. Let $\langle a, b \rangle$ be the ideal location of a sensor. Let e_d be the distance a sensor is perturbed from its ideal location, and θ_d be the angle of perturbation. The error distance e_d is determined using the normal distribution $N(\mu, \sigma)$, where μ is the mean error distance and σ is the standard deviation of e_d. Thus, the error in location on 95% of the sensors is in the range $(-\mu-2\sigma, \mu+2\sigma)$. Hence, to determine the topology, we increase the *physical communication range* by $\mu+2\sigma$. However, the communication and interference range used by the algorithm is 1. For small perturbations (i.e., $\mu \leq 0.2$), increasing the physical communication range is sufficient to ensure that the network is connected.

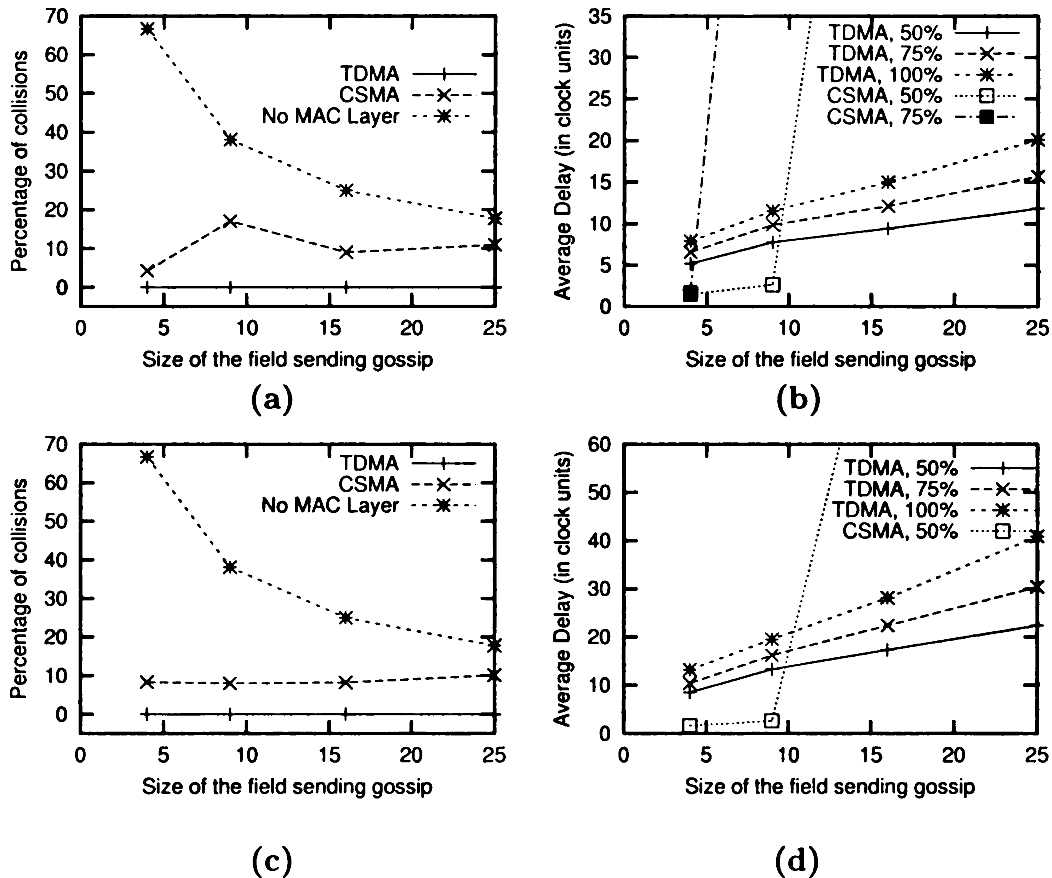


Figure 5.12: Results for local gossip; (a) and (b) with communication range = 1, interference range = 1, (c) and (d) with communication range = 1, interference range = 2

However, for larger perturbations (i.e, $\mu > 0.2$), if the communication and interference range used by the algorithm is 1, number of collisions increase significantly. Hence, we need to increase the interference range that the algorithm uses, say, to 2. Additionally, if the predicted μ is less than the actual mean error, the algorithm can increase its interference range when it observes significantly higher number of collisions using the approach to change the collision group size (cf. Section 4.4.2).

In our simulations, μ takes the following values: $0.0-0.4$ and σ takes the following values: $0.0-0.2$. And, θ_d is determined using the uniform distribution $U(0, 2\pi)$. Thus, the actual location of the sensor is $\langle a+e_d\cos(\theta_d), b+e_d\sin(\theta_d)\rangle$.

Broadcast. In Figure 5.13, we present the simulations results for broadcast with location errors. Figure 5.13(a) identifies the percentage of collisions during broadcast. As we can see, when μ increases, the number of collisions increases. However, the collisions are within 2%. Figure 5.13(b) identifies the maximum delay involved in delivering the broadcast message to all the sensors. We can note that the delay is within 10% when compared to the case where no location errors are introduced. Finally, Figure 5.13(c) identifies the number of sensors receiving the broadcast message. As we can observe, all the sensors receive the message except for the case where $\mu = 0.2$ and $\sigma = 0.1$. Even in this case, more than 96% of the sensors receive the broadcast message. Table 5.1 shows the percentage of collision for the case where $\mu = 0.4$ and interference range=2. As we can observe, the percentage of collisions is small with increased interference range.

Convergecast. In Figures 5.14(a) and 5.14(b), we present the simulation results for convergecast with location errors. Figure 5.14(a) identifies the number of collisions during the message communication. We note that, as the error in sensor location increases, collisions increase. Further, as observed earlier, the collisions are within acceptable limits, i.e., within 8%. Figure 5.14(b) identifies the average delay involved in delivering the convergecast messages to the base station; the average delay is within 2% when compared to the case where were no location errors are introduced. Further, similar to the case where no location errors are present, the base station receives all the convergecast messages. Moreover, if the mean error distance increases, we can

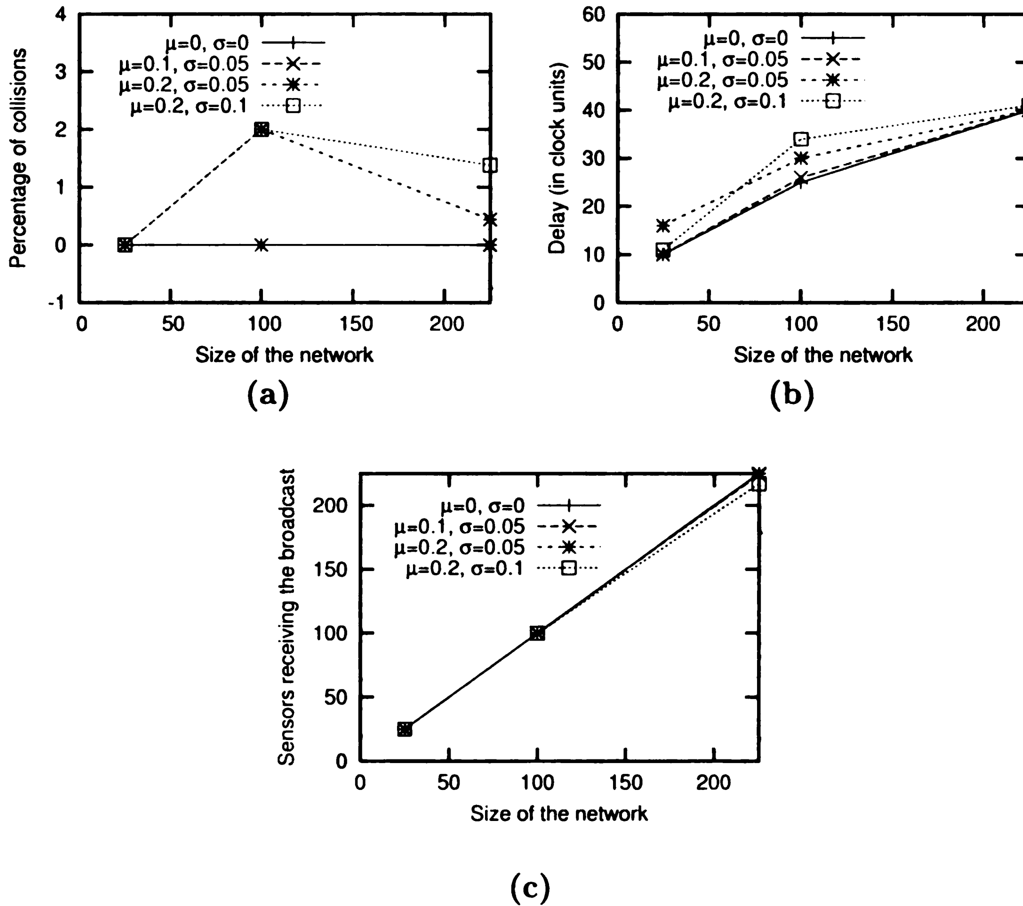


Figure 5.13: Results for broadcast with location errors

keep the percentage of collisions small by increasing the interference range (cf. Table 5.1).

Local gossip. In Figures 5.14(c) and 5.14(d), we present the simulation results for local gossip with location errors. Similar to the observations made earlier in this section, from Figure 5.14(c), we observe that the number of collisions during message communication is small (i.e., within 4%). Further, the delay involved in delivering the local gossip messages is within 15% when compared to the case where no location errors are introduced. Finally, all the local gossip messages are delivered to the group

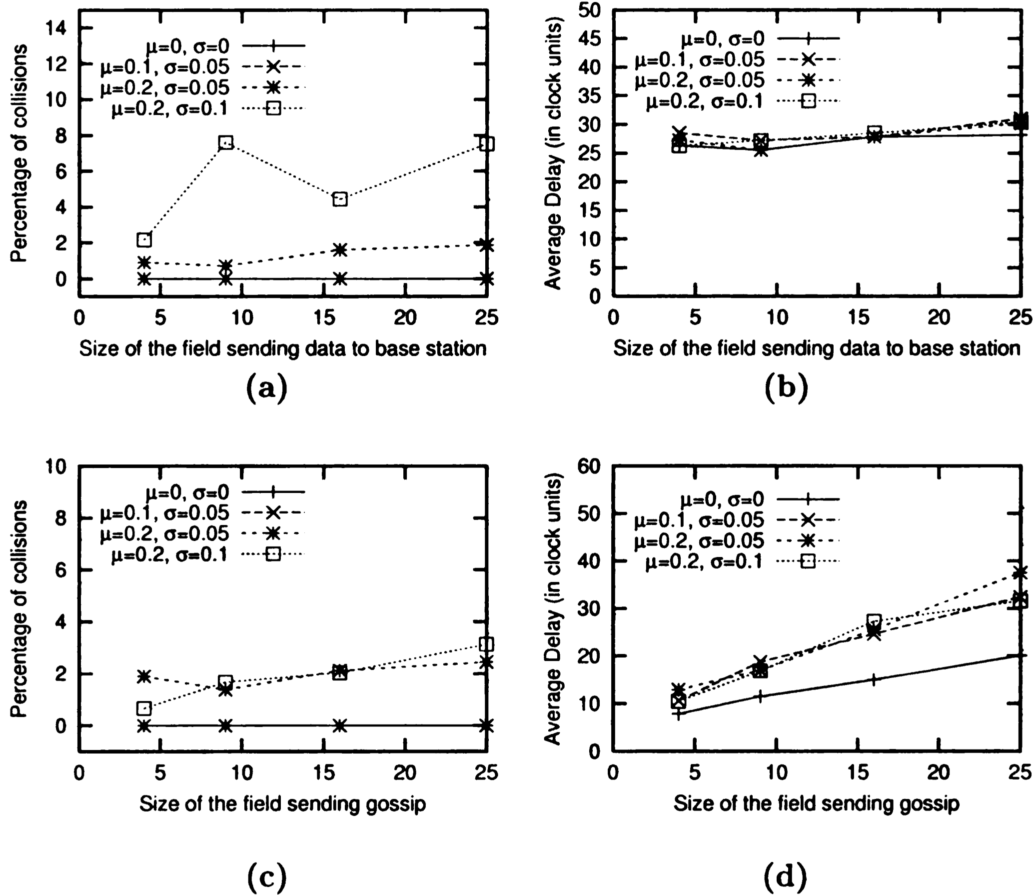


Figure 5.14: Results for convergecast and local gossip with location errors

that expects such messages. Moreover, if the mean error distance increases, we can keep the percentage of collisions small by increasing the interference range (cf. Table 5.1). From these simulations, we conclude that the location errors do not significantly affect the performance of our TDMA service.

5.4.5 Effect of Grouping Constant

In the proposed TDMA service for sensor networks, if the TDMA service receives two or more messages and the sum of the message lengths is less than the maximum

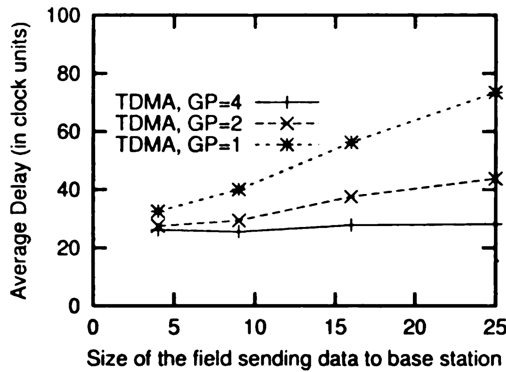
Table 5.1: Results for $\mu = 0.4$, $\sigma = 0.2$, and interference range=2

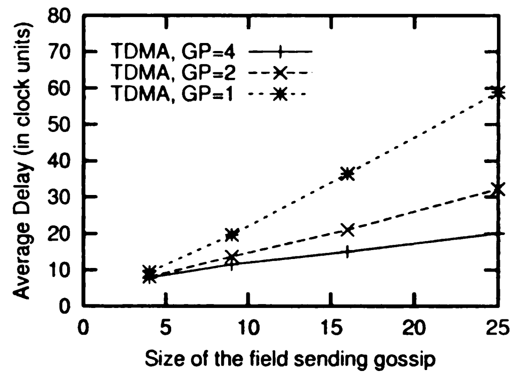
Network Size	Broadcast % of Collisions
25	0
100	5
225	6.8

Field Size	Convergecast % of Collisions	Local Gossip % of Collisions
4	9.6	4.9
9	8.8	7.1
16	11.7	7.5
25	11.5	6.6

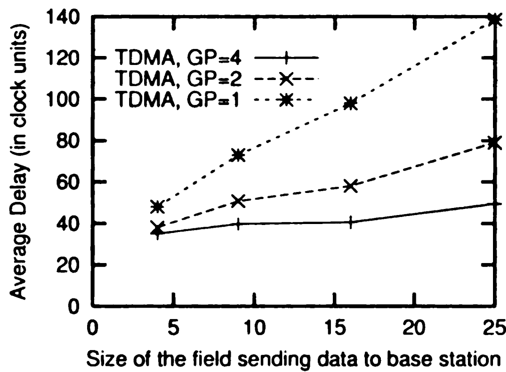
message length of a TDMA message, TDMA service combines these messages into a single TDMA message. In this section, we study the effect of grouping. Specifically, we study the effect of varying the number of messages grouped into a single TDMA message for convergecast and local gossip. Note that in the simulations for broadcast only one message is transmitted, and hence, we do not consider the issue of grouping for broadcast. Further, in this section, we present results only for the delay in delivering the messages. In our simulations, the base station (respectively, the group expecting the gossip messages) receives all the convergecast (respectively, gossip) messages. Also, the percentage of collisions is zero.

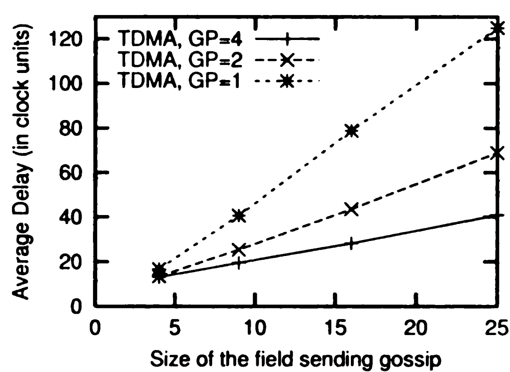
In Figures 5.15(a) and 5.15(b), we present the simulation results for convergecast and local gossip where the communication range is 1 and interference range is 1. We consider the following values for grouping constant (GP): 1, 2, and 4. In Figure 5.15(a), we can observe that as the size of the field sending convergecast message to base station increases, the delay increases. Further, delay increases when the number of messages grouped into single TDMA message decreases. With GP as 1 (i.e., no grouping), we observe that the delay increases significantly as size of the network increases.





(a) (b)





(a) (b)

Figure 5.15: Effect of grouping constant; with communication range = 1, interference range = 1, for (a) convergecast and (b) local gossip, and with communication range = 1, interference range = 2, for (c) convergecast and (d) local gossip

In Figure 5.15(b), we present the results for local gossip. We observe that the results are similar to convergecast. Specifically, as GP decreases, delay in delivering the local gossip messages increases.

Once again, the results for the case where the communication range is 1 and interference range is 2 are similar (cf. Figures 5.15(c) and 5.15(d)).

CHAPTER 6

Application: Model Conversions for Sensor Networks

The ability of modeling abstract distributed programs and transforming them into concrete programs that preserve the properties of interest is one of the important problems in distributed systems. Such transformation allows one to write a program in one (typically a simpler/restrictive) model and run it on another (typically a general/less restrictive) model. Hence, several algorithms (e.g., [16–21]) have been proposed for enabling such transformation.

As discussed in [22] (cf. Chapter 1), the model of computation in sensor networks can be viewed as *write all with collision* (WAC) model. Intuitively, in this model, in one atomic action, a sensor (process) can update its own state and the state of all its neighbors. However, if two sensors (processes) simultaneously try to update the state of a sensor (process), say k, then the state of k is unchanged.

Moreover, in sensor networks, detecting such collisions is difficult due to several reasons. For example, it is possible that a sensor succeeds in updating the state of one of its neighbors even though its update causes collision at another neighbor. Hence, we assume that collisions are not detectable. We would like to note that most of our results are also applicable for the case where collisions can be detected. We discuss this issue in Section 6.5.

While previous literature has focused on transformations among other models of computation (e.g., [16–21]), the issue of transformation from (respectively, to) WAC model to (respectively, from) other models has not been considered. To redress this deficiency, we focus on the problem of identifying the transformations that will

allow us to transform programs in the WAC model to other models considered in the literature, and vice versa

Existing models and semantics for distributed programs. Some of the commonly encountered models of computations include a message passing model, a read/write model, and a shared memory model. In these models, a program consists of a set of processes and each process consists of a set of *actions.* And, the tasks that are performed in an action depend upon the model of computation. In message passing model, processes share no memory and they communicate by sending and receiving messages. Thus, in each action, a process can perform one of the following tasks: send a message, receive a message, or perform some internal computation. A read/write model reduces the complexity in modeling message passing programs. In read/write model, the variables of a process are split into public variables and private variables. In each action, the process can either (1) read the state of one of its neighbors (and update its private variables), or (2) write its own variables (public and private) using its own variables (public and private). Thus, read/write model allows one to hide the complexities of message queues, message delays, etc. The shared memory model simplifies the read/write model further in that in one action it allows a process to atomically read its state as well as the state of its neighbors and write its own state. Thus, the shared memory model hides the intermediate states, where a process has read the state of a subset of its neighbors, that occur in read/write model.

For shared memory model, there are different types of semantics that are often used. Some of the commonly encountered semantics include, interleaving (also known as central daemon), maximum-parallelism and power-set semantics (also known as distributed daemon). In interleaving semantics, given a set of enabled actions, i.e.,

actions whose execution will change the state of some process, one of those actions is non-deterministically chosen for execution. In maximum-parallelism, all enabled actions (one from each process) are executed concurrently. And, in power-set semantics, any non-empty subset of enabled actions (at most one from each process) is executed concurrently.

In this chapter, we focus on transformation from WAC model under power-set semantics into read/write model, and vice versa. We show that previously studied concepts such as graph coloring (e.g., [23–25]), local mutual exclusion (e.g., [19, 20]) and collision-free diffusion [26] can be effectively used for obtaining these transformations. The main results are as follows:

- For untimed (asynchronous) systems, we present an algorithm for the transformation of programs in read/write model into programs in WAC model. We also show the optimality of this transformation; specifically, we show that if the transformed program is deterministic, cannot use time and cannot perform redundant writes (see Section 6.2.1 for definition) then at most one process can execute at a time. Also, we argue that these transformations cannot be made stabilization preserving.

- For timed systems, we present algorithms for the transformation of programs in read/write model into programs in WAC model for a grid topology and also, for arbitrary graphs. These transformations permit concurrent execution of multiple processes. We also show that if the given program in read/write model is stabilizing fault-tolerant [4,5], i.e., starting from an arbitrary state, it recovers to states from where its specification is satisfied, then, for a fixed topology, the transformed program in WAC model is also stabilizing fault-tolerant.

- We present an algorithm for the transformation of programs in WAC model into programs in read/write model. We show that this transformation is also stabilization preserving. This transformation does not assume that the topology is fixed or known in advance.

- We show how to transform programs in shared memory/message passing model to WAC model, and vice versa.

In Section 6.1, we present the formal definitions of the atomicity models and our approach for proving that our transformation algorithms are stabilization preserving and also, identify the system assumptions. In Section 6.2, we present an optimal approach for the transformation of programs in read/write model into programs in WAC model under power-set semantics for untimed systems. Subsequently, in Section 6.3, we present the transformation for timed systems. Then, in Section 6.4, we present an approach for the transformation of programs in WAC model under power-set semantics into programs in read/write model. Finally, in Section 6.5, we discuss some of the questions raised by the transformations.

6.1 Atomicity Models, Preserving Stabilization and System Assumptions

In this section, we first precisely specify the structure of the programs in read/write model and in WAC model. Then, we define stabilizing fault-tolerance and discuss our approach to show that our transformation algorithms preserve stabilization. Finally, we present the assumptions made about the underlying system.

The programs are specified in terms of guarded commands; each guarded command (respectively, action) is of the form:

guard $\longrightarrow$ *statement*,

where *guard* is a predicate over program variables, and *statement* updates program variables. An action $g \longrightarrow st$ is enabled when g evaluates to true and to execute that action, st is executed atomically. A computation of this program consists of a sequence $s_0, s_1, \ldots$, where s_{j+1} is obtained from s_j by executing actions (one or more, depending upon the semantics being used) in the program.

A computation model limits the variables that an action can read and write. Towards this end, we split the program actions into a set of processes. Each action is associated with one of the processes in the program. We now describe how we model the restrictions imposed by the read/write model and the WAC model.

Read/Write model. In read/write model, a process consists of a set of public variables and a set of private variables. In the read action, a process reads (one or more) public variables of one of its neighbors. For simplicity, we assume that each process j has only one public variable $v.j$ that captures the values of all variables that any neighbor of j can read.

Furthermore, in a read action, a process could read the public variables of its neighbor and write a different value in its private variable. For example, consider a case where each process has a variable x and j wants to compute the sum of the x values of its neighbors. In this case, j could read the x values of its neighbors in sequence. Whenever j reads $x.k$, it can update a private variable $sum.j$ to be $sum.j + x.k$. Once again, for simplicity, we assume that in the read action where process j reads the state of k, j simply copies the public variables of k. In other words, in the above case, we require j to copy the x values of all its neighbors and then use them to compute the sum later.

Based on the above discussion, we assume that each process j has one public variable, $v.j$. It also maintains $copy.j.k$ for each neighbor k of j; $copy.j.k$ captures

the value of $v.k$ when j read it last. Now, a read action by which process j reads the state of k is represented as follows:

$$true \quad \longrightarrow \quad copy.j.k = v.k$$

And, the write action at j uses $v.j$ and $copy.j$ (i.e., copy variables for each neighbor) and any other private variables that j maintains to update $v.j$. Thus, the write action at j is as follows:

$$predicate(v.j, copy.j, other_private_variables.j) \quad \longrightarrow \quad \text{update } v.j,\ other_private_variables.j;$$

Remark. Note that the above representation assumes that the read action is always enabled. If this is not the case then j can ignore the value it read when the read action was not enabled. This can be achieved as follows: In addition to maintaining $copy.j.k$, j also maintains $rcopy.j.k$. When the read action in the given program is enabled, j executes the above read action to set $copy.j.k$ to $v.k$ and then set $rcopy.j.k$ to $copy.j.k$. Since $copy.j.k$ and $rcopy.j.k$ are local variables, these two actions can be trivially serialized. With these modifications, when the read action at j is not enabled, any update to $copy.j.k$ is ignored.

Write-All-With-Collision (WAC) model. In WAC model, each process consists of write actions (to be precise, write-all actions). Each write action at j writes the state of j and the state of its neighbors. Similar to the case in read/write model, we assume that each process j has a variable $v.j$ that captures all the variables that j can potentially write to any of its neighbors. Likewise, process j maintains $l.j.k$ for each neighbor k; $l.j.k$ denotes the value of $v.k$ when k wrote it last. Thus, an action in WAC model is as follows:

$predicate(v.j, l.j, other_private_variables.j) \longrightarrow$ update $v.j$, $other_private_$
$variables.j$;
$\forall k : k$ is a neighbor of j :
$l.k.j = v.j$;

Remark. In the rest of the chapter, we leave the additional private variables considered above implicit.

Preserving stabilization. Since our transformations are stabilizing fault-tolerant, we discuss our approach in proving stabilization now. Towards this end, we define the notion of equivalence between a computation of the given program and computation of the transformed program. This notion is based on the definition of refinement [27,28] and simulation [28].

Consider the transformation of program p in read/write model into program p' in WAC model. Note that in WAC model, multiple writes can occur at once whereas in read/write model, at most one read/write can occur at a time. Hence, each step of the program in WAC model would be simulated in read/write model by multiple steps. Hence, if $c' = \langle s_0, s_1, \ldots \rangle$ is a computation of p' and c is a computation of p, we say that c and c' are equivalent if c is of the form $\langle t_{00}, t_{01}, \ldots, t_{0f_0}(= t_{10}), t_{11}, \ldots, t_{1f_1}(= t_{20}), \ldots \rangle$, where $\forall j : s_j$ and t_{j0} are identical (subject to renaming of variables) as far as the variables of p are concerned. For the transformation of p' in WAC model into p in read/write model, the definition of equivalence is similar. We require that c can be expressed as $\langle t_{00}, t_{01}, \ldots \rangle$ by commuting the read/write actions in c in such a way that the effect of these transitions does not change due to such commutation.

To show that our transformations are stabilization preserving, we proceed as follows: Let p be the given program, say, in read/write model, and let p' be the transformed program in WAC model. We show that given any computation of p', there

exists a suffix of that computation such that there is a computation of p that is equivalent to that suffix. If the given program, p, is stabilizing fault-tolerant then any computation of p is guaranteed to reach legitimate states and satisfy the specification after reaching legitimate states. It follows that eventually, a computation of p' will reach legitimate states from where it satisfies its specification.

System assumptions. We assume that the set of processes in the system are connected. If the set of processes are partitioned then the algorithms in this chapter can be executed for each partition. Further, we assume that in the given program in read/write model, for any pair of neighbors j and k, j can never conclude that k does not need to read the state of k. In other words, we require that the transformation should be correct even if each process executes infinitely often. Further, in our transformation from read/write model to WAC model, we assume that the topology remains fixed during the program execution, i.e., failure or repair of processes does not occur. Thus, while proving stabilization, we disallow corruption of topology related information. This assumption is similar to assumptions in previous stabilizing algorithms where the process IDs are considered to be incorruptible. We note that our transformation from WAC model to read/write model does not assume that the topology is known up front and the topology can change at run time.

We consider two types of systems, timed and untimed. In a timed system, each process has a clock variable. We assume that the rate of increase of the clocks is same for all the processes. In an untimed system (also known as asynchronous systems), processes do not have the notion of time or the speed of processes.

6.2 Read/Write Model to WAC Model in Untimed Systems

In this section, we discuss the first of our transformations where we transform a program in read/write model into a program in WAC model. Recall that in the read/write model, each program action is either a read action or a write action. In the WAC model, there is no equivalent of a read action. Hence, an action by which process j reads the state of k in the read/write model can be modeled in the WAC model by requiring process k to write the appropriate value at process j. Of course, when k writes the state of j in this manner, it is necessary that no other neighbor of j is writing the state of j at the same time. Finally, a write action in read/write model can be executed in WAC model as is.

To obtain the transformed program that is correct in WAC model, we organize the processes in the given program (in read/write model) in a ring. Such a ring can be *statically* embedded in any arbitrary graph by first embedding a spanning tree in it and then using an appropriate traversal mechanism to ensure that each process appears at least once in the ring (cf. Figure 6.1).

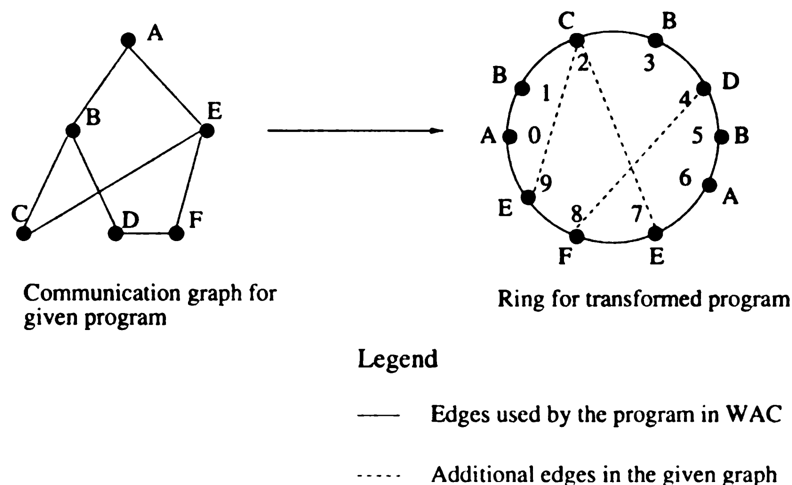


Figure 6.1: In the transformed program, 'A' will execute actions of processes 0 and 6, 'B' will execute actions of processes 1, 3 and 5, and so on.

Let the processes in this ring be numbered $0 \dots n-1$; note that if a process from the original graph is repeated in the ring then that process gets multiple numbers in this ring. Now, in the transformed program, process 0 executes first. When process 0 executes and writes the state of process 1 (and any other processes that are neighbors of process 0 in the original communication graph), process 1 is enabled and permitted to execute. When process 1 executes, it allows process 2 to execute, and so on. The actions of process j in the transformed program are as follows: (If a process in the original graph has multiple numbers in the ring, it executes the actions corresponding to all those values.)

```
process j
const
        n;                      // number of processes in the ring
var
        v.j, l.j, counter.j;
initially
        set v.j according to the initial value of v.j in read/write model;
        set l.j according to the initial value of copy.j in read/write model;
        counter.j = 0;
begin
        counter.j = j  --> if(predicate(v.j, l.j)) update v.j;
                           counter.j = (j + 1) mod n;
                           ∀k : k is a neighbor of j : l.k.j, counter.k = v.j, (j + 1)mod n;
                         // this write action will enable process numbered j + 1
end
```

Figure 6.2: Read/write model to WAC model in untimed systems

Theorem 6.1 For every computation of the transformed program in WAC model under power-set semantics there is an equivalent computation of the given program in read/write model.

Proof. The main idea in this proof is that every step of j in the transformed program is equivalent to the following computation of the original program in read/write model: (write action by j, followed by read action by each neighbor of j).

Consider a computation $s_0, s_1, s_2, \ldots$ of the transformed program in WAC model. Based on the initial values of the counters, (s_0, s_1) is a transition of process 0. Let $x_1, x_2, \ldots, x_{f_0-1}$ be the neighbors of 0. Now, for the transition (s_0, s_1), we construct a computation $\langle t_{00}, t_{01}, t_{02}, \ldots, t_{0f_0}\rangle$ of the original program. Since t_{ab} is a state of the program in the read/write model, we identify the v and *copy* values for t_{ab}.

- For state t_{00}: $\forall j :: v.j(t_{00}) = v.j(s_0)$, $\forall j, k :: copy.j.k(t_{00}) = l.j.k(s_0)$.
- State t_{01} is obtained from t_{00} by executing the write action at process 0 (i.e., the process that was numbered 0 during transformation).
- State $t_{0w}, 1 < w \leq f_0$, is obtained from $t_{0(w-1)}$ where process x_{w-1} reads the value of $v.0$.

Now, in $t_{10} = t_{0f_0}$, we have $\forall j :: v.j(t_{10}) = v.j(s_1)$, $\forall j, k :: copy.j.k(t_{10}) = l.j.k(s_1)$. Further, by induction, if $s_0, s_1, s_2, \ldots$ is a computation of the transformed program in WAC model then there exists an equivalent computation $t_{00}, t_{01}, \ldots, t_{0f_0}(= t_{10}), t_{11}, \ldots$ that is a computation of the given program in read/write model. □

6.2.1 Optimality Issues

In this section, we discuss the optimality of the above transformation. Towards this end, we first identify the notion of redundant writes. Based on the definition of WAC model, the guard of any action, say of process j, in WAC model depends only on the local variables of j. Now, consider the case where process j executes its action (and writes its state as well as the state of its neighbors) and an action of j is still

enabled after this execution. It follows that j can again execute and write the state of its neighbors. In this scenario, one of the writes of j is redundant if the system was untimed and no collision had occurred. To show the optimality of the above transformation, we assume that processes do not perform such redundant writes.

Redundant writes. We say that process j does not perform redundant writes if after the execution of any action by j, all actions of j are disabled until a neighbor of j executes and writes the state of j.

Theorem 6.2 An algorithm for transforming programs in read/write model to programs in WAC model can allow at most one process to execute at a time, if the system is untimed and processes are deterministic, cannot detect collisions, and cannot perform redundant writes.

Proof. As discussed in Section 6.1, in this proof, we assume that the transformation algorithm does not have any information about the fact that some processes execute only finite number of times in the given read/write program. In other words, the transformation should succeed even if each process executes infinitely often in the given program. We also assume that the transformation algorithm should work correctly even if communication among neighbors is bidirectional, i.e., it should not assume that communication between some neighbors be only unidirectional.

Further, we observe that in WAC model, if the system is untimed and processes are deterministic then a process can go from being disabled to being enabled only if one of its neighboring processes writes its state. Now, consider a program in read/write model that is designed for the system in Figure 6.3. In the transformed program, assume that j executes at some step. This could allow processes in the left part and the processes in the right part to execute concurrently. Now, we show that if such

concurrent execution is permitted then it is impossible to ensure that j can execute once more.

Since c_1 and c_2 may need to write the state of j, the transformation algorithm must ensure that j is disabled at some point after it executes. This will allow one of these neighbors to write the state of j. However, for the write to j to succeed, either c_1 should execute or c_2 should execute, but not both. As we can see, in an untimed, deterministic system, there is no way to ensure that exactly one process from c_1 and c_2 executes if they write the state of j once. Thus, for the system in Figure 6.3, only one process can execute at a time.

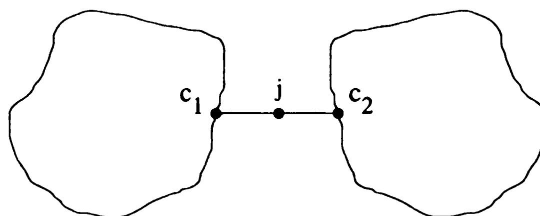


Figure 6.3: Impossibility of executing more than one process in untimed systems

Moreover, even if there are additional edges (other than $c_1 \leftrightarrow j, c_2 \leftrightarrow j$) among processes in the left network (left of process j), processes in the right network (right of process j), and process j, the above argument still holds. Thus, the theorem holds for all arbitrary connected graphs with at least 3 processes. Note that the proof is trivial for graphs with only 1 or 2 process(es). Thus, an algorithm for transforming programs in read/write model into programs in WAC model, where the system is untimed and processes are deterministic, cannot detect collisions, and cannot perform redundant writes, can allow at most one process to execute at a time. $\square$

There are several ways to improve the performance of the transformation if we weaken some of the assumptions made above. Specifically, we can remove the assumption that processes cannot perform redundant writes in order to allow concurrency in the programs in WAC model. Next, we present a solution that provides potential concurrency if processes are allowed to perform redundant writes. Another approach would be to remove the assumption about untimed systems. If the processes are allowed to use time, we can design transformation algorithms that allow more concurrency (cf. Section 6.3).

Concurrent executions. We show that it is possible to design transformation algorithms that allow concurrent executions of processes with redundant writes. Specifically, in this section, we present an algorithm where a concurrency of 2 is potentially possible. In this example, initially, processes are mapped onto a logical ring. The communication graph of the original program can have additional links.

For simplicity of presentation, we consider a ring with 6 processes as shown in Figure 6.4. Let s and t be two special processes. Process s is initially enabled. Whenever s executes, it passes a *token* to the process chains a_1, a_2 (top processes) and b_1, b_2 (bottom processes), allowing them to execute concurrently. In this execution, we guarantee that the write actions of the top processes always succeed. Whenever process t is enabled due to the write actions of the top processes, process t reverses the token circulation direction. In other words, when t executes, it passes the token to the process chains b_2, b_1 and a_2, a_1, thereby allowing them to execute concurrently. In the reverse pass, we guarantee that the write actions of the bottom processes always succeed.

Now, we describe how we ensure that the write action of top (respectively, bottom) processes succeed in the forward (respectively, reverse) circulation of token. In the

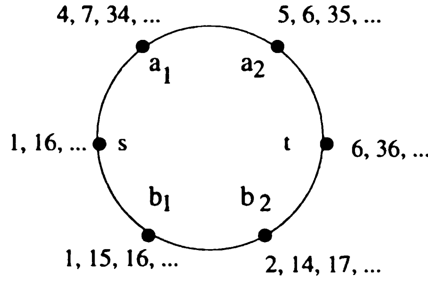


Figure 6.4: Executing more than one process in untimed systems. The numbers associated with each process denotes the number of writes the process executes in the corresponding round.

forward pass, when s executes, it writes the state of a_1 and b_1, thereby enabling them. Now, processes a_1 and b_1 can execute concurrently. Since their write actions may collide, we need to ensure that at least one of their writes succeed. To ensure that the writes of the processes in top chain (i.e, a_1, a_2) succeed, when a_1 executes, it writes more than the sum of number of write actions of the processes in the bottom chain (i.e., b_1, b_2). In our solution, when b_1 executes, it writes once, and when b_2 executes, it writes twice. Hence, when a_1 executes, it writes four times, and when a_2 executes, it writes five times. Thus, one or more of the write actions of a_1 and a_2 succeed, thereby enabling successive processes a_2 and t respectively. When the write actions of both a_1 and b_1 succeed at s in the forward pass, s receives confirmation for its write, and hence s cancels its pending writes. When a_2 (respectively, t) writes its state to a_1 (respectively, a_2) successfully, a_1 (respectively, a_2) cancels its pending writes. Now, process t will be enabled since one of the write actions of a_2 would succeed. Note that process t is enabled only by the write action of process a_2, even though the writes of process b_2 may succeed at t. When process t executes, it reverses the token circulation direction. Further, it initiates the next round, and hence, the number of write actions by each process change accordingly (cf. Figure 6.4). Moreover, if the write action of t

succeeds at b_2 before b_2 executes all its forward writes, b_2 cancels its pending forward writes. And, process t cancels its pending writes when it receives confirmation about its successful writes at b_2 and a_2.

In the reverse pass, we ensure that the writes of the processes in bottom chain succeed. Thus, similar to the above discussion, when a_2 executes, it writes six times, and when a_1 executes, it writes seven times. Hence, when b_2 executes, it writes 14 times, and when b_1 executes, it writes 15 times. Further, process s is enabled only by the write action of b_1, even though the write actions of a_1 may succeed at s. Continuing thus, we can allow two processes to execute concurrently.

We observe that it is not possible to bound the number of write actions by each process in this solution. Further, in this solution, it is possible to have a scenario where only one process is executing at a time. For example, consider the following scenario. Initially, process s executes, and allows process chains a_1, a_2 and b_1, b_2 to execute. However, the bottom chain is slow, and hence, process t becomes enabled by the execution of the top processes. If t executes, writes of the bottom processes may be canceled. Therefore, it is possible to have a scenario where only one process executes at a time. However, as we have discussed above, there is a potential for concurrency.

6.2.2 Stabilization Issues

Based on the assumption that processes do not perform redundant writes, for each process, say j, there exists a local state of that process where none of its actions are enabled. Also, since the guard of an action at process j depends only on local variables of j, we can perturb the given program to states where none of the actions

are enabled. It follows that it will not be possible for the program to reach legitimate states from such a state.

In the context of the above result, a reader may wonder if a stabilizing token ring circulation algorithm could be used to achieve stabilization in the above transformation. To add stabilization to the transformed program in WAC model, we need a stabilizing token ring circulation algorithm that is correct under the WAC model. By contrast, existing stabilizing token ring circulation algorithms are correct under read/write model.

Note that the above impossibility result depends on the assumption that the system is untimed and processes are deterministic, cannot detect collisions, and cannot perform redundant writes. If the assumption about untimed system is removed then it is possible to preserve stabilizing fault-tolerance. We present such stabilization preserving algorithms in Section 6.3.

6.3 Read/Write Model to WAC Model in Timed Systems

In this section, we present the algorithm for the transformation of a program in read/write model into a program in WAC model for timed systems. We present two transformations; one for a grid topology (cf. Section 6.3.1) and the other for an arbitrary topology (cf. Section 6.3.2). For these two transformations, we assume that the clocks of the processes are initialized to 0 and the rate of increase of the clocks is same for all processes. In Section 6.3.3, we present an approach to deal with uninitialized clocks; this approach enables us to ensure that if the given program in read/write model is stabilizing fault-tolerant then the transformed program in WAC model is also stabilizing fault-tolerant.

6.3.1 Transformation for Grid Topology

This algorithm is based on the collision-free diffusion algorithm presented in [26]. Let the processes be arranged in a 2-d grid topology as shown in Figure 6.5. Also, assume that there is a distinguished process at the left-top position (process $\langle 0,0 \rangle$). This distinguished process starts the computation in the transformed program. First, this distinguished process executes and writes the state of processes $\langle 1,0 \rangle$ and $\langle 0,1 \rangle$. Now, either $\langle 1,0 \rangle$, $\langle 0,1 \rangle$ or both can execute. However, if both $\langle 1,0 \rangle$ and $\langle 0,1 \rangle$ execute simultaneously, their write actions collide at process $\langle 1,1 \rangle$. Hence, we use the following algorithm to ensure collision-freedom during write actions: process $\langle 1,0 \rangle$ executes one cycle (i.e., time required to execute one action in the WAC model) after $\langle 0,0 \rangle$ writes its state. And, process $\langle 0,1 \rangle$ executes after two cycles. In general, if the distinguished process starts the computation when its *clock* is equal to 0, then process $\langle a,b \rangle$ executes at $clock = a + 2b$.

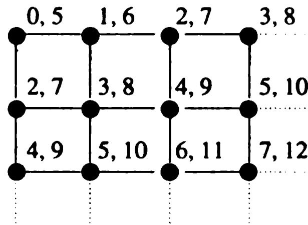


Figure 6.5: Time slots for processes in a grid. The numbers associated with a process show the first two slots in which it could execute.

Further, in the above algorithm, the distinguished process can execute again when its clock equals 5 (cf. Figure 6.5). In this case, the write action of $\langle 0,0 \rangle$ does not collide with simultaneous write actions of other processes. Hence, in general, process j located at $\langle a,b \rangle$ can execute and write its state to its neighbors at $\forall c : c \geq 0 :$

$slot.j + c * 5$, where $slot.j (= a + 2b)$ is the initial slot assignment. The actions of process j at location $\langle a, b\rangle$ are as follows:

```
process j                    // located at ⟨a, b⟩
const
        slot.j = a + 2b;     // by symmetry, slot.j could also be initialized to 2a + b
        period = 5;
var
        v.j, l.j, clock.j;
initially
        set v.j according to the initial value of v.j in read/write model;
        set l.j according to the initial value of copy.j in read/write model;
        clock.j = 0;
begin
        (∃c : clock.j = slot.j + c * period) ⟶ if(predicate(v.j, l.j)) update v.j;
                                                ∀k : k is a neighbor j : l.k.j = v.j;
end
```

Figure 6.6: Transformation for grid topology

Theorem 6.3 For every computation of the transformed program in WAC model under power-set semantics there is an equivalent computation of the given program in read/write model.

Proof. Consider the program in the WAC model. Based on the initial values of the clocks, the distinguished process (process $\langle 0, 0\rangle$) starts the computation. Further, the initial values of the variables of the program are assigned according to the program in the read/write model.

Similar to the proof of Theorem 6.1, a write action by process j in the transformed program is equivalent to the following computation of the original program in read/write model: a write by process j, followed by the read actions by the neighbors of j.

Further, we note that concurrent executions are possible in the program under WAC model. Since each action of the program in WAC model is guaranteed to be collision-free, even if multiple processes execute their write actions simultaneously, the resulting state change can be obtained by a computation in the read/write model. Thus, for a computation of the transformed program in the WAC model under power-set semantics, there is an equivalent computation of the given program in the read/write model. □

Observation 6.4 There exists a suffix of the computation of the transformed program where the number of processes enabled at any instant of time is maximal. □

6.3.2 Transformation for Arbitrary Topology

In this section, we extend the solution in Section 6.3.1 so that programs in read/write model on arbitrary communication graphs can also be transformed into programs in WAC model. The main idea behind this extension is graph coloring. For this extension, let the communication graph in the given program in read/write model be $G=(V,E)$. We transform this graph into $G'=(V,E')$ such that $E'=\{(x,y)|(x \neq y) \wedge ((x,y) \in E \vee (\exists z :: (x,z) \in E \wedge (z,y) \in E))\}$ (cf. Figure 6.7). In other words, two distinct vertices x and y are connected in G' if distance between x and y in G is at most 2. Let $f : V \rightarrow [0 \ldots K-1]$ be the color assignments such that $(\forall j,k : k$ is a neighbor of j in $G' : f(j) \neq f(k))$, where K is any number that is sufficient for coloring G'.

Let $f.j$ denote the color of process j. Now, process j can execute at $clock.j = f.j$. Moreover, process j can execute at time slots $\forall c : c \geq 0 : f.j + c * K$. When j executes, it writes its own state and the state of its neighbors in G. Based on the

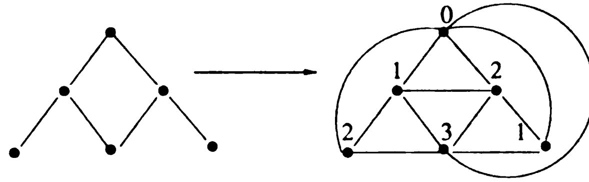


Figure 6.7: Transformation using graph coloring. The number associated with each process denotes the color of the process.

color assignment, it follows that two write actions do not collide. The actions of process j are as follows:

```
process j
const
        f.j;                                    // color of the process
        K;                                      // colors used to color the transformed graph
var
        v.j, l.j, clock.j;
initially
        set v.j according to the initial value of v.j in read/write model;
        set l.j according to the initial value of copy.j in read/write model;
        clock.j = 0;
begin
        (∃c : clock.j = f.j + c * K) ⟶ if(predicate(v.j, l.j)) update v.j;
                                       ∀k : k is a neighbor of j in the original graph :
                                                      l.k.j = v.j;
end
```

Figure 6.8: Transformation for arbitrary topology

Theorem 6.5 For every computation of the transformed program in WAC model under power-set semantics there is an equivalent computation of the given program in read/write model.

Proof. The proof of this transformation is similar to that of Theorem 6.3. □

Theorem 6.6 If the maximum degree of G is d, then the period between successive executions of a process is at most d^2+1.

Proof. If the maximum degree of G is d then the maximum degree in G' would be at most d^2. In [25], it has been shown that if each subgraph of a graph has a vertex of degree $K-1$ or less, then the graph can be colored with K colors. Hence, G' can be colored with at most $K=d^2+1$ colors. Thus, the period between successive executions of process j is at most d^2+1. □

Remark. The above theorem presents the upper bound on the performance of the transformation for arbitrary graphs. For sensor networks (e.g., [3, 6]), maximum degree d is usually small. Hence, the period between successive executions of a process is small. Moreover, d^2+1 is the upper bound for the period. For example, for a grid topology, maximum degree, $d=4$, and $period=K=5$.

6.3.3 Preserving Stabilization during Transformation

In this section, we discuss the modifications for the transformation algorithms in Section 6.3.2 to ensure that stabilization is preserved during transformation. (Similar modification can be achieved for transformation in Section 6.3.1.) Towards this end, we first discuss how we deal with the case where the clocks are not initialized or clocks of processes are corrupted. (Note that the rate of increase of the clocks is still the same for all processes.) Then, we show that with these modifications, if the given program in read/write model is stabilizing fault-tolerant then the transformed program is also stabilizing fault-tolerant in WAC model.

To recover from uninitialized clocks, we proceed as follows: Initially, we construct a spanning tree of processes. Let $p.j$ denote the parent of process j in this spanning tree. Process j is initialized with a constant $c.j$ which captures the difference between

the initial slot assignment of j and $p.j$ (i.e., the difference between the clocks of j and $p.j$ when j and $p.j$ execute for the first time).

If clocks are not synchronized, the action by which $p.j$ writes the state of j may collide with other write actions in the system. Process j uses the absence of this write to stop and wait for synchronizing its clock. When j later observes the write action performed by $p.j$, it can use $c.j$ to determine the next slot in which it should execute.

Since the root process continues to execute in the slots assigned to it, eventually, its children will synchronize with the root. Subsequently, the grandchildren of the root will synchronize, and so on. Continuing thus, the clocks of all processes will be synchronized so that the further computation will be collision-free.

In the absence of topology changes, the spanning tree constructed above and the values of $f.j$ (slot assigned to j based on graph coloring) and $c.j$ (difference between slots assigned to j and $p.j$) are constants. Hence, for a fixed topology, we assume that these values are not corrupted; this assumption is similar to the assumption that process IDs are not corrupted. Under these assumptions, if the program is perturbed then eventually, the clocks recover to states from where the subsequent computation is collision-free. Once the clocks are restored, from Theorem 6.5, for the subsequent computation of the transformed program, there is an equivalent computation of the given program in read/write model. Also, if the given program is stabilizing fault-tolerant then every computation of that program reaches legitimate states. Combining these two results, it follows that every computation of the transformed program eventually reaches legitimate states. Thus, we have

Theorem 6.7 If the given program in read/write model is stabilizing fault-tolerant then transformed program (with the modifications discussed above) in WAC model is also stabilizing fault-tolerant. □

6.4 WAC Model to Read/Write Model

In this section, we focus on the transformation of a program that is correct in the WAC model under power-set semantics into a program in read/write model. Recall that an action of the program in WAC model allows a process to write its own state and the state of its neighbors. By contrast, a program in read/write model can either read the state of one of its neighbors or write its own state. Hence, during this transformation, we need to split an action in WAC model into multiple actions in read/write model. Specifically, an action by which j writes its own state and the state of its neighbors is split so that j writes its own state and then allows each of its neighbors to read its state.

However, if multiple fragmented WAC actions are executed simultaneously in the read/write model, their execution may not correspond to the sequential (respectively, parallel) execution of those actions in the WAC model. For example, consider the execution of two actions, $a.j$ and $a.k$, at neighboring processes j and k in the WAC model. In a fragmented execution of these two actions, the following execution scenario is feasible: j writes its own state as prescribed by action $a.j$, k writes its own state as prescribed by $a.k$, j reads the state of k, and k reads the state of j. However, such a scenario is not possible in WAC model. Specifically, if $a.j$ and $a.k$ are executed simultaneously then, due to collision, j (respectively, k) will not be able to write the state of k (respectively, j). And, in a sequential execution where $a.k$ is executed after $a.j$, k will be aware of the new state of j and use that in the execution of $a.k$. By contrast, in the above fragmented execution, this property was not true. Thus, the fragmented execution of two actions in the WAC model may not correspond to their sequential or parallel execution. Hence, to transform the given program in WAC

model, we ensure that two neighboring processes do not simultaneously execute their fragmented WAC actions.

The problem of ensuring that neighboring processes do not execute simultaneously is a well-known problem in distributed computing. It has been studied in the context of local mutual exclusion (e.g., [19,20]), dining philosophers (e.g., [18,29]) and drinking philosophers (e.g., [18, 29]). Since either of these solutions suffices for our purpose, we simply describe the features of these solutions that are of importance here. Note that the local mutual exclusion algorithm used in this context must be correct in read/write model only. And, solutions in [18–20, 29] are indeed correct in read/write model.

Each of the solutions in [18–20, 29] has the following two actions: *enterCS* and *exitCS*. These solutions further guarantee that when a process is in its critical section (i.e., it has executed *enterCS* but not *exitCS*) none of its neighbors is in its critical section. Using these two actions, next, we demonstrate how one can transform a program in WAC model into a program in read/write model. (Note that the last action appears to allow j to read the state of all its neighbors; this action can be *slowly* executed so that j reads the counters of its neighbors one at a time.)

```
process j
var
    v.j, copy.j, counter.j;          // bounding of counter.j is discussed in Section 6.5
initially
    set v.j according to the initial value of v.j in WAC model;
    set copy.j according to the initial value of l.j in WAC model;
    ∀k :: counter.j.k = 0;
begin
    upon executing enterCS          ⟶    counter.j.j := counter.j.j + 1;
                                          execute 'write part' of the WAC
                                          atomicity action to update v.j;
    counter.j.k < counter.k.k       ⟶    counter.j.k, copy.j.k := counter.k.k, v.k;
    (∀k : counter.k.j ≥ counter.j.j) ⟶   execute exitCS;
                                          request for CS again;
end
```

Figure 6.9: WAC model to read/write model

Theorem 6.8 For every computation of the transformed program in read/write model there is an equivalent computation of the given program in WAC model under power-set semantics.

Proof. Consider a computation of the transformed program in read/write model. In the absence of state perturbations, we have, $(\forall j, k :: counter.k.j \leq counter.j.j)$. Now, when j executes $enterCS$, it increments $counter.j.j$. It follows that j cannot execute $exitCS$ until all neighbors of j copy the new value of $v.j$. Based on the guarantees of local mutual exclusion, neighbors of j do not execute until j exits critical section. Thus, the write action at j followed by read action by neighbors of j is equivalent to the action in WAC model that writes the state of j and all its neighbors.

Further, in the above transformation, it is guaranteed that two neighbors are not in the middle of executing a write action. Thus, for the computation of the transformed program in read/write model, there is an equivalent computation where the corresponding actions in WAC model are executed sequentially. In other words,

for every computation of the transformed program in read/write model there is an equivalent computation of the original program in WAC model under interleaving semantics. Based on the definition of power-set semantics, for every computation of the transformed program in read/write model there is an equivalent computation of the original program in WAC model under power-set semantics. □

6.4.1 Preserving Stabilization during Transformation

In this section, we show that if we use a local mutual exclusion algorithm that is stabilizing (e.g., [18–20]) then the above transformation is stabilization preserving. To show this, we first observe that any stabilizing solution for local mutual exclusion must ensure that eventually some process enters its critical section. Consider the case where process j is in critical section. If there exists a neighbor, say k, of j such that $counter.k.j < counter.j.j$ then k will copy $counter.j.j$. Thus, eventually, j will be able to exit critical section. Based on the above discussion, it follows that the stabilization property of local mutual exclusion is preserved. Hence, the program will recover to states from where no two neighboring processes are in their critical sections. Once every process enters CS sufficiently many times (equal to the maximum counter value in the initial state), the counter values will be restored, i.e., $\forall j, k : counter.k.j \leq counter.j.j$ will be true. Once counter values are restored, based on Theorem 6.8, for the subsequent computation of the transformed program in read/write model there is an equivalent computation of the given program in WAC model. Thus, we have

Theorem 6.9 If the given program in WAC model is stabilizing fault-tolerant then the transformed program in read/write model is also stabilizing fault-tolerant. □

6.5 Discussion

Our transformations from (respectively, to) WAC model to (respectively, from) read/write model raises several questions. We discuss some of these questions and their answers, next.

In this chapter, we considered transformations of programs in read/write model to WAC model, and vice versa. Can similar transformations be done for shared memory model or message passing model?

Yes. It is possible to transform a program in shared memory model or message passing model to WAC model, and vice versa. Towards this end, we need to combine the algorithms presented in this chapter with the transformations from these models to read/write model. For example, given a program in shared memory model, we can transform it to obtain a corresponding program in read/write model. Examples of such transformation algorithms include [18, 20]. Given a program that is correct in shared memory model, these algorithms transform it to obtain a corresponding program that is correct under the read/write model. Once we obtain a program that is correct in read/write model, we can use the algorithms in Section 6.2 or 6.3 to transform it into a program that is correct under the WAC model. Moreover, given a program in WAC model, we can use the algorithm in Section 6.4 to transform it into an algorithm that is correct in read/write model. By definition, it is correct under shared memory model.

Regarding transformation from WAC model to message passing model, we can first use our algorithm from Section 6.4. Then, we can use the approach in [18] to transform it to a program in message passing model. Likewise, given a program in

message passing model, we can first obtain a program in read/write model and then transform it into a program in WAC model.

How efficient are these transformations?

First, we compute the efficiency of the transformation algorithm from read/write model to WAC model. For untimed systems, according to Theorem 6.2, at most one process can execute at a time, if the system is untimed and processes are deterministic, cannot detect collisions, and cannot perform redundant writes. Hence, the one enabled process may take up to $O(N)$ time before it can execute next, where N is the number of processes in the system. Thus, in untimed systems, the transformation from read/write model to WAC model can slow down the algorithm in WAC model. However, as shown in Section 6.2.1, for this model this delay is inevitable. For timed systems, a process executes once in K slots where K is the number of colors used to color the extended communication graph. Thus, in timed systems, the transformation can slow down the given algorithm by a factor of K that is bounded by d^2, where d is the maximum degree of any node in the graph. Note that this slow down is reasonable in sensor networks where topology is typically geometric and value of K is small. For example, in commonly occurring grid topology $K=5$.

Likewise, in the transformation from WAC model to read/write model, the local mutual exclusion algorithm prevents two neighboring processes from executing concurrently. The slow down caused by this is $O(d)$ where d is the maximum degree in the given communication graph.

In the transformation shown in Section 6.4, the value of counter is unbounded. Can it be bounded?

Yes. Although it is preferred to bound the size of variables in stabilizing programs, the definition of self-stabilization does not preclude unbounded counters. Moreover,

a simple modification from [30] allows us to bound the counter while preserving stabilization. To bound the counter using the approach in [30], we maintain the bound B on counter to be a number greater than $N^2 + 1$ where N is the number of processes in the system. Now, the counter is incremented modulo B. While the details of this algorithm and the proof of boundedness is beyond the scope of this thesis, we describe the modifications needed to bound the counter value.

With this modification, j increments $counter.j.j$ if the counter values of its neighbors are in the (circular) range $[counter.j.j, counter.j.j + N]$. If j and k are neighbors such that the circular difference between $counter.j.j$ and $counter.j.k$ is larger than N and $counter.j.j > counter.j.k$ then j resets its counter to 0.

Are the transformations discussed in this chapter possible if the processes can detect collisions?

Yes. The transformations proposed in Section 6.2 for untimed systems and in Section 6.3 for timed systems are correct even if the processes can detect collisions. However, the optimality of the transformation for untimed systems (cf. Section 6.2.1) may not hold. Specifically, in Theorem 6.2, we assume that a process can go from being disabled to being enabled only if its neighbor writes its state. In an algorithm where collision detection is possible, a process can also go from being disabled to being enabled when it detects a collision.

How does the WAC model differ from 'point-to-point message passing' with collision model?

The model considered in this chapter is different from 'point-to-point message passing' with collision model. Notably, if we consider a network with four processes, $1, 2, 3, 4$, arranged in a line, then simultaneously, process 1 can send a message to 2 while 3 sends a message to 4. By contrast, in WAC model, if 1 and 3 transmit

simultaneously then there will be a collision at 2. We have chosen the WAC model as in sensor networks (e.g., [3, 6]), the only available communication primitive is broadcast to neighbors.

CHAPTER 7

Related Work

In this chapter, we focus on the previous work in sensor networks, medium access control (MAC) protocols for radio and sensor networks, energy-efficiency issues in the design of MAC protocols, and time synchronization. In Section 7.1, we introduce the challenges and opportunities that are inherent in sensor networks. Also, we discuss some of the recent developments in the field of sensor hardware, operating systems and programming language. Then, in Section 7.2, we discuss the previous work on communication protocols for radio/wireless networks. The protocols discussed here may not directly apply to sensor networks due to resource constraints. However, in Section 7.3, we discuss MAC protocols for sensor networks that enhances the protocols discussed in Section7.2 by dealing with the challenges in sensor networks. Further, most MAC protocols need the sensors to have synchronized time. Time synchronization is an important problem by itself. Previous work on time synchronization is discussed in Section 7.4 with emphasis on sensor networks. Finally, since our algorithms are self-stabilizing, we discuss related work on self-stabilizing protocols in Section 7.5.

7.1 Sensor Networks

In this section, we discuss the challenges and opportunities in sensor networks [1,2]. Further, we discuss the hardware limitations and operating system design for sensor networks [3, 6].

7.1.1 Challenges and Opportunities

Challenges, applications, and current technologies in sensor networks. In [1], Estrin et al introduce the challenges in the design of applications for sensor/pervasive networks, classify the expected types of systems, and discuss the current technological developments as well as future research directions. Specifically, the authors argue that sensor networks pose the following challenges: *scale*, *access*, and *environment*. By scale, the authors claim that applications usually require a large number of sensor devices. By access, the authors argue that the sensors are deployed in places where human access is limited. And, the authors visualize the environment where the sensors would be typically deployed. Next, the authors classify the applications based on three parameters: *scale*, *variability*, and *autonomy*. By scale, the authors mean that the systems can be classified by the amount of activity sensors perform, or extent to which the sensors are deployed. By variability, they mean that the systems can be classified based on the topology structure used, task performed, or mobility model used. And, by autonomy, the authors classify the system based on modalities or complexity. Finally, the authors provide a summary on recent technological developments, especially, in sensing and actuation, and localization services.

In this thesis, we show that our algorithms can handle the challenges in sensor networks. Specifically, our algorithms can be used to assign time slots to all the sensors in one collision-free diffusing computation. Further, time slots can be assigned without human intervention by sending a signal to the base station to start the diffusion. Finally, our algorithms can work in different environments and moreover, our algorithms can tolerate errors introduced in sensor location due to vehicle/intruder movement in the field. Thus, our algorithms deal with the challenges identified in [1].

Design principles for sensor networks. In [2], Estrin et al identify the design considerations for applications in sensor networks. Specifically, they argue that the networking model for sensor networks should be data-centric and application-specific, rather than traditional models like the Internet. Further, they motivate the necessity of localized algorithms for sensor networks, which rely on coordination among different sensors to execute a certain task. More specifically, since a sensor can communicate only with its neighbors up to a certain distance, localized algorithms are important to achieve a global property/goal. The authors illustrate their claim using a localized clustering algorithm for electing extremal sensors to report the location of an intruder/object. Further, the authors introduce directed diffusion, where abstractions for communication patterns that occur mostly in localized algorithms are provided. A more formal treatment of directed diffusion can be found in [31]. Finally, the authors provide the following design suggestions for designing localized algorithms. First, develop/model simple localized algorithms (e.g., adaptive fidelity algorithms where the quality is compromised for energy, bandwidth, etc). Then, characterize the performance of the localized algorithms.

In this thesis, we proposed TDMA algorithms customized for the application requirements. Further, the sensors coordinate among themselves to assign the initial slots and also, to determine the future TDMA slots. Thus, our algorithm follow some of the design considerations proposed in [2].

7.1.2 Sensor Hardware, Operating System and Programming Language

In this section, we introduce the hardware, operating system, and the programming language for sensor networks. In [3,6], the authors introduce the sensor hardware, *MICA motes*, developed by the University of California, Berkeley. Also, the

authors briefly outline the operating system, *TinyOS*, developed for such devices. In [13, 14], the authors discuss the design and implementation of the programming language, *nesC*, used to program these devices. We used MICA motes, TinyOS, and nesC in our implementation. Now, we briefly outline the features of MICA motes, TinyOS, and nesC.

MICA Motes. A typical MICA mote includes the following: a 4MHz Atmel ATMEGA103 processor, 128KB of flash memory to store application and operating system programs, 4KB of data memory, a single-channel low-power radio, 512KB of EEPROM secondary memory, and a sensor board with sensors such as photo sensor, thermostat, microphone, sounder, magnetometer, accelerometer, etc, attached to the main board on an expansion bus. The power consumption of these devices range from $5\mu A - 5mA$.

TinyOS. TinyOS is an event-driven tiny operating system for low-power, resource constrained sensor devices. It uses a component-based (or modular) architecture. Typically, each component consists of *commands*, *events*, *tasks*, and a *frame*. Commands are non-blocking requests to low-level component services. Events are asynchronous and deal with the hardware directly/indirectly. Tasks are light-weight threads that are atomic (i.e., run-to-completion). Tasks are usually scheduled by a FIFO scheduler. Frame is the memory/state of a component. Commands, events and tasks operate in the context of the frame. Frame has a fixed-size which is known at compile-time.

Components can be classified into three types: *hardware abstractions*, *synthetic hardware*, and *high-level software components*. Hardware abstraction components

map the hardware to the component model of TinyOS. Synthetic hardware components performs the role of device drivers. And, high-level software components provide the higher-level functionalities.

A typical TinyOS application is written by wiring the interface of different components. Note that this wiring need not be 1:1. At compile-time, all the wiring information is translated into assembly-level logic.

TinyOS has an additional feature called *crossing-layers without buffering.* This is similar to the acknowledgement based protocols, where the upper layer sends more data when it receives an acknowledgement from the lower layer. Further, in TinyOS, components such as CSMA, TDMA, time synchronization, power-management, localization, routing can be added as a middleware service to an application. Hence, the operating system remains small and is applicable in tiny embedded systems.

nesC. nesC [13], an extension of C, is the language used for programming MICA motes. Some of the ideas used in nesC are as follows:

- *Component-based design of programs.* Components are the basic building blocks of programs. As discussed earlier, components consists of commands, events, frame, and tasks.

- *Interface-based specification of component behaviors.* Component behaviors are expressed using interfaces. A component may use an interface (meaning, use the behaviors/functionalities exposed in the interface), or provide an interface (meaning, provide the behaviors/functionalities exposed in the interface).

- *Static/compile-time linking of components.* Components are statically wired to form the whole application. This allows the designer to statically analyze the program.

In addition to the features listed above, nesC provides the support for event-driven programming and concurrency. Further, since the components are statically linked, it allows whole-program analysis, race-detections and resource reduction optimizations. The language also provides features like parameterized interfaces, thereby allowing more instances of the same component in a program. Further, it reflects the design of TinyOS and hence, is more suited for MICA motes.

7.2 Communication Protocols for Radio/Wireless Networks

Related work that deals with communication issues in radio/wireless includes [32–34].

Fault-tolerant broadcasting. In [32], Kranakis et al provide a fault-tolerant broadcasting algorithm in radio networks. The model proposed in [32] assumes that the upper bound on the number of faulty nodes is known at start. They also assume that the faults are permanent. The authors do not consider the notion of interference range for nodes. The authors propose two versions of broadcasting algorithm for networks of line/grid topology: a non-adaptive version and an adaptive version. In the non-adaptive version, all transmissions are scheduled in advance while in the adaptive version, nodes can schedule future transmissions based on the communication history. The adaptive algorithm has two phases: a pre-processing phase and a broadcasting phase. In the pre-processing phase, the nodes determine fault status of other nodes and in the broadcasting phase, the actual transmission takes place. The work reported in [32] differs from our approach considerably. The assumption about knowledge of the number of faults is not made in our algorithms. Also, unlike [32], we allow sensors to fail/recover during computation. In order to reduce the collision-group size, we

use the diffusion messages and there are no separate slots assigned for determining faulty nodes.

Broadcasting in mobile/ad hoc networks. Work reported in [33, 34] provide algorithms for deterministic broadcasting. Specifically, in [33], Gasieniec et al provide algorithms for completely connected synchronous broadcast networks. The paper studies the difference between a globally synchronous (global clock) and a locally synchronous (local clock with same rate of increase) model with known/unknown network size. In [34], Chlebus et al provide algorithms for mobile/ad hoc networks. They provide broadcasting algorithms for a model without collision detection and a model with collision detection. These two papers assume that the network is fully connected. These algorithms are not stabilizing fault-tolerant.

MAC protocols for wireless networks. MAC layer protocols for wireless networks are discussed in [35]. It introduces the different collision-freedom and collision-avoidance protocols that are commonly used. Collision-free protocols like FDMA and TDMA are analyzed and compared. The authors argue that the static collision-free protocols do not utilize the medium effectively. Specifically, they argue that FDMA and TDMA waste bandwidth when the medium is lightly loaded. Further, the authors show that the two schemes have almost similar throughput delay characteristics. Another collision-free protocol, called code division multiple access (CDMA) is introduced in [36]. However, in CDMA, encoding and decoding of messages in the link-layer can be expensive.

Collision-avoidance protocols like CSMA and CSMA/CD are also used in wireless networks. Although such protocols are highly applicable in wired networks, CSMA and CSMA/CD are still used in wireless context. The main problem with CSMA and CSMA/CD is the hidden-terminal effect. With the use of request-to-send (*RTS*) and

clear-to-send (*CTS*) signals, CSMA becomes a valuable option in wireless scenario. But, collision-detection and backoff requires more power as the nodes need to listen to the medium regularly and determine the correct time to transmit a message. In [35], the authors give a detailed analysis of the collision-avoidance protocols applicable in radio/wireless networks.

7.3 MAC Protocols for Sensor Networks

Designing MAC layer protocols is one of the important research area in sensor networks. The protocol should be efficient in terms of the resources it uses due to the constraints inherent in sensor networks. Collision-freedom and collision-avoidance MAC protocols are proposed for sensor networks. Collision-avoidance medium access control (MAC) protocols like carrier sense multiple access (CSMA) [35,37] try to avoid collisions by sensing the medium before transmitting a message. Another example of collision-avoidance protocol is carrier sense multiple access and collision detection (CSMA/CD). CSMA/CD [35] is difficult to use in the context of sensor networks as the collisions are often detected at some receivers whereas other receivers and sender(s) may not detect the collision.

Collision-free medium access control protocols such as frequency division multiple access (FDMA), code division multiple access (CDMA), and time division multiple access (TDMA) ensure that collisions do not occur while the sensors communicate. FDMA [35] ensures collision-freedom by allotting different frequencies for the sensors. FDMA is not applicable in the context of sensor networks since the sensors (e.g., University of California, Berkeley's MICA motes [3, 6]) are often restricted to transmit only on one frequency. CDMA [36] requires that the codes used to encode

the message be orthogonal to each other so that the destination can separate different messages. Thus, CDMA requires expensive operations for encoding/decoding a message. Therefore, CDMA is not preferred for sensor networks that lack the special hardware required for CDMA and that have limited computing power.

Work reported in [37] presents a CSMA based protocol for sensor networks, especially, for motes [3, 6]. Majority of other work on medium access are based on collision-free MAC protocols, more specifically, TDMA based protocols. In Section 7.3.1, we discuss a CSMA based MAC protocol for sensor networks and compare it with our approach. And, in Section 7.3.2, we present collision-free MAC protocols for sensor networks and compare them with our approach. As noted earlier, energy efficiency in sensor networks is a major concern in designing any algorithm. In Section 7.3.3, we discuss some of the energy efficient MAC protocols from the literature. Also, we show how our protocol differs from others.

7.3.1 CSMA Based MAC Protocols

In [37], Woo and Culler introduce a new CSMA based MAC protocol for sensor networks. Specifically, the authors propose an algorithm that does not use *RTS/CTS* signals to solve the hidden-terminal problem. Their algorithm is however similar to the traditional CSMA/CD algorithms found in wired networks, where nodes sense the medium before transmitting a message. To reduce contention, the standard random backoff strategy is used.

The authors propose an adaptive transmission control scheme to control the amount of messages currently transmitted in the network. Further, this scheme helps in taking care of the hidden terminal problem in a multi-hop scenario (i.e., multi-hop hidden terminal problem). The transmission control scheme is similar to the linear

increase and multiplicative decrease approach to control the rate of transmission. Whenever the medium is not busy (i.e., successful transmission of a message), transmission rate is increased by a linear factor. And, whenever the node senses contention (i.e., failed transmission of a message), transmission rate is multiplicatively decreased. Multihop hidden terminal problem is solved using the transmission control scheme and phase changes. If a node sends a message at time t, it expects that the child would transmit the message at time $t{+}x{+}PACKET_TIME$, where x is the processing time at child, and $PACKET_TIME$ is the propagation time of a message. Thus, the adaptive transmission control scheme reduces the transmission rate such that a node does not contend for the medium with its children.

7.3.2 Collision-Free MAC Protocols

In this section, we discuss the related work on collision-free MAC protocols in sensor networks. Collision-free protocols for sensor networks mainly includes TDMA based schemes. TDMA protocols can be classified as randomized and deterministic protocols, based on the way time slots are alloted to different sensors or the startup algorithm works. Randomized TDMA protocols include [38–40]. And, deterministic TDMA protocols include [26, 41].

TDMA for distributed embedded systems. In [38], Claesson et al have proposed a randomized startup algorithm for TDMA. Whenever a collision occurs during startup, exponential backoff is used for determining the time to transmit next. In Chapter 4, we propose a deterministic startup algorithm that guarantees collision-freedom and stabilization in case of fail-stop failures. Further, the complexity of the algorithm proposed in [38] is $O(N)$ where N is the number of system nodes, whereas

the complexity of our diffusion algorithm is $O(D)$ where D is the diameter of the network. Moreover, the algorithm in [38] optimizes time and communication overhead with increased computation overhead, while our diffusion algorithm optimizes all the three overheads. The disadvantage of our algorithm is that it has a single point of failure (i.e., initiator of diffusion). In situations where the initiator fails, we can use the startup algorithm from [38] to assign TDMA slots.

TDMA and network organization. In [39], Sohrabi and Pottie propose a network self-organization protocol, where nodes identify the presence of other nodes and form a multi-hop network. In [40], Heinzelman et al propose a hierarchical clustering algorithm. In both these papers, initially, nodes are in random-access mode (i.e., nodes use CSMA scheme to communicate) and TDMA slots are assigned to the nodes during the process of network organization. During network organization, unlike [40], the protocol proposed in [39] allow nodes to communicate. Further, in [39], a node periodically spends time in random access mode to identify other nodes in the network. In [40], nodes use the alloted TDMA schedule only in the steady-state phase (i.e., after network organization is complete). By contrast, in our TDMA service, we use a collision-free deterministic startup algorithm to assign time slots to different sensors.

In [41], Arisha et al propose a TDMA based MAC scheme for sensor networks. Their approach uses clustering to allot time slots to different sensors. Each cluster has a gateway node. The gateway node informs each sensor in its cluster about the time slots in which the sensor should listen and also, the time slots in which the sensor can transmit messages. This model combines the MAC layer protocols with the routing protocols. The protocol consists of the following phases: *data-transfer phase*, *refresh phase*, and *event-triggered/refresh-based re-routing phase*. In

the data-transfer phase, sensors send messages in their alloted time slots. In the refresh phase, sensors inform the gateway node about its status in their pre-assigned time slot. And in the event-triggered/refresh-based re-routing phase, sensors update their routing/forwarding table. Event-triggered re-routing is done to conserve energy and the refresh-based re-routing is done whenever the routing information changes, due to some sensors becoming non-functional. Unlike our approach, this scheme combines the MAC layer protocol and routing protocol. Further, the sensors are organized into a multiple-cluster based networks with a leader/gateway node.

7.3.3 Energy Efficient MAC Protocols

As discussed earlier, energy efficient algorithms are desired for sensor networks. Work reporting energy-efficient MAC layer protocols include [41–43].

Energy-efficiency through periodic updates. The MAC layer protocol proposed in [41] (cf. Section 7.3.2), achieves energy-efficiency by allowing the sensors to update their forwarding table periodically. Further, the routing information updates routes based on the sensors energy-usage. Thus, if a particular sensor has limited energy, then the protocol can choose a different route where the sensors in the path have enough energy. Moreover, simulation results in the paper show that the number of packets lost in the network due to the presence of non-functional sensors approaches zero, when the buffer size at each sensor is increased. Also, the protocols allow sensors to go to idle state. However, the sensors should periodically inform the gateway node about their status. Thus, the protocol differs from the approach presented in Chapter 4, where the sensors need to inform only its neighbors about their state.

Power aware MAC protocol. Another interesting energy efficient MAC layer protocol is presented in [42] for ad hoc networks. Singh and Raghavendra propose

a power aware MAC protocol called PAMAS. PAMAS is based on the multiple access with collision avoidance (MACA) protocol [44]. This protocol uses a separate signaling channel that is different from the packet transmission channel. Signaling channel based protocols are studied in [45,46]. Note that the *RTS/CTS* signals associated with the MACA protocol is transmitted in the signaling channel. Further, this signaling channel can be used to determine the time for which nodes can power-off. The protocol has 6 states: *idle*, where the nodes cannot transmit or receive any packets, *awaitCTS*, where a node waits for *CTS* after transmitting *RTS* in the signaling channel, *binary exponential backoff (BEB)*, where the nodes backoff for some period if *CTS* is not received, *transmit packet*, where the nodes transmit the packet after receiving *CTS*, *await packet*, where the nodes wait for the packet after transmitting *CTS*, and *receive packet*, where a node receives packet. Further, when a node is in *receive packet* state, a busy-tone is signaled in the signaling channel. The authors also provide simulation results and mathematical bounds on power-conservation.

PAMAS is not directly applicable in sensor networks. PAMAS requires separate radio channel for signaling purposes. Also, it uses lot of resources, especially for detecting collisions, computing the sleep time, etc. Finally, *RTS/CTS* signaling mechanism is not applicable in sensor networks directly. To overcome the difficulties in applying PAMAS to sensor networks, in [43], a new MAC protocol for sensor networks is designed.

Sensor-MAC (S-MAC) protocol. In [43], Ye et al propose a energy efficient sensor-MAC (S-MAC) protocol. The paper identifies the sources of energy waste: *collisions*, where the sensors waste energy in transmitting the collided messages, *overhearing*, where the sensors listen messages that were not intended for them, *idle listening*, where the idle sensors need to listen to the medium in order to receive

the messages destined from them, and *control packet overhead*, where energy is spent in transmitting control messages. S-MAC proposes the following ideas in order to reduce the energy consumption:

1. periodic listen and sleep (reduces energy spent in idle listening).

2. based on PAMAS, using in-channel signaling to allow some neighboring sensors to sleep when the sensor is transmitting messages for other sensors.

3. using message-passing model in order to reduce the latency perceived by the applications and also, the control overhead involved.

The basic model of S-MAC is similar to CSMA protocol with *RTS/CTS* to avoid collisions. This approach differs from the protocol presented in Chapter 4. Specifically, our approach uses time-synchronization protocols to assign time slots to different sensors. Further, the time slots assigned can be validated in future (using the diffusing computation). Also, we show sensors can go to sleep as necessary except that they need to listen for the diffusion message periodically.

Table 7.1 lists the different approaches used in the design of energy-efficient MAC protocols. As we can observe from the table, a sensor must send status updates in order to save energy, if it uses a time-slot based MAC protocol. However, the advantage of using such an approach is that the sensor can inform its neighbors and go to sleep state at any time. In non-time slot based MAC protocols, the sensors are required to listen to the control commands like *RTS/CTS* in order to go to sleep state.

Table 7.1: Comparing different energy-efficient MAC protocols

	MAC protocol	Signaling	When sensors can go to sleep	Status updates
Arisha et al [41]	TDMA	In-channel	Any time	Periodically update the gateway node
Singh et al [42]	MACA	Separate signaling channel	Based on the busy tone in the signaling channel	None
Ye et al [43]	CSMA	In-Channel	When other sensors are using the channel	None
Kulkarni et al [26]	TDMA	In-Channel	Any time	Update the collision group size used in the network or listen in diffusion slots

7.4 Time Synchronization

Time synchronization is important in wireless/sensor networks that use TDMA based MAC protocols. Specifically, time synchronization is necessary to ensure that all sensors agree upon the time slots they use and that the time slots of neighboring sensors are sufficiently separated. Also, synchronization is necessary to deal with clock skew in the sensors. In [38], time synchronization protocol is proposed for assigning time slots to different nodes in a distributed embedded system. Work related to time synchronization in sensor networks are reported in [47–49].

TDMA synchronization for distributed embedded systems. As discussed in Section 7.3.2, in [38], Claesson et al present a startup algorithm for synchronizing the nodes in the network to use TDMA. The important assumption in this paper is that each node has unique message length, and hence the propagation time of a

message is unique among different senders. The synchronization/TDMA protocol uses this information in assigning time slots to different nodes. The synchronization protocol works in three modes: *normal*, *resynchronization*, and *recovery*. Initially, all the nodes are in recovery mode.

- *Recovery mode.* In recovery mode, a node (say, r) records the correct/incorrect reception of a message. If messages from majority of the nodes are received correctly then the protocol goes to normal mode. Further, node r records its time slot as $TC_r = ST_s + t_{ms}$, where ST_s is the local time when the message was sent by s, and t_{ms} is the propagation time. If node r sends a message successfully in its alloted time slot, then the protocol goes to resynchronization mode.

- *Normal mode.* In normal mode, a node (say, r) updates its time slot for every correctly received message from s as $TC_r = TC_r + t_{ms}$. If an incorrect/no message is received, the time slot TC_r is incremented by propagation time of the expected message. Further, similar to recovery mode operation, it records the correct/incorrect reception of messages from other nodes. If majority of the messages are incorrectly received, then the protocol goes into resynchronization mode.

- *Resynchronization mode.* In resynchronization mode, similar to the other modes of operation, nodes record the correct/incorrect reception of messages. Here, the protocol operations are similar to the normal mode operation. However, if the medium is completely silent for one full communication cycle (i.e., propagation time of the longest message), then it goes into recovery mode.

Moreover, the paper also presents proof of correctness of the above protocol and validates it experimentally. As discussed in Section 7.3.2, it has a number of shortcomings compared to the startup algorithm proposed in Chapter 3.

Design principles for time synchronization service in sensor networks. In [47], Elson and Romer argue that the synchronization protocols designed for traditional networks are not applicable in sensor networks. Moreover, the authors suggest the following design principles for time synchronization service in sensor networks: *energy-efficiency*, *scalability*, *robustness*, and *ad hoc deployment*. Further, the paper argues using global timescale is not feasible in sensor networks. Moreover, the paper recommends *post-facto* synchronization instead of synchronizing *a priori*. Another design principle suggested is to adapt the protocol for the application in hand. However, since the application requirements vary over time, tunable protocol/service is desired. Also, parameterizable or adaptive fidelity algorithm is proposed (as discussed in [2]). Thus, the system can choose from a set of synchronization algorithms based on the requirements of the application. Additionally, the paper also suggests to use the domain knowledge available. For example, in [50], time synchronization is achieved by leveraging the properties of the communication medium. Specifically, reference-broadcast based synchronization is achieved using the physical-layer broadcast channel.

Post-facto synchronization. In [48], Elson and Estrin propose the *post-facto* synchronization protocol, which rely on a third-party node. The paper identifies three main sources of error that affect the time synchronization service: receiver clock skew, variable delays (in detecting the synchronization signal due to nondeterminism in hardware and operating system) on the receivers, propagation delay of the synchronization signal/pulse. In their solution, nodes are normally considered to be unsynchronized. When a stimulus arrives, nodes record the stimulus time using their

local clock. Immediately, a third-party node, sends a synchronization pulse. Nodes receiving this pulse normalize their stimulus time using the time when they received the synchronization pulse as reference. By using a third-party node, the error factors that affect the protocol are reduced.

Tree and max algorithm for time synchronization. In [49], Herman proposes a time synchronization service for tiny sensor devices (e.g., *motes* [3,6]). This service maintains a tree structure of motes where the root sends a periodic beacon message about its time. Each non-root node gets the best-approximation of the root's time from the neighbor which is closest to the root.

Table 7.2: Comparing different time-synchronization protocols

	Protocol	Assumptions about messages	Assumptions about third-party node
Claesson et al [38]	Based on the unique propagation time of messages by different senders	Each sensor has a unique message length	None
Elson and Estrin [48]	Normalization of time based on the synchronization and stimulus pulses	None	Yes. Sends the synchronization pulse
Herman [49]	Tree-based algorithm: $time.j = time.(parent.j)$. Recent algorithm: $time.j = max(\forall k : time.k)$	None	None

Based on our observations with motes, collision-freedom is important in these system-wide computations. Hence, we designed collision-free diffusion and TDMA algorithms. Moreover, the TDMA algorithm discussed in Chapter 4 are orthogonal to the time synchronization approaches proposed in these papers. Specifically, the

TDMA algorithm can be used for collision-free transmission of the time synchronization messages, thereby enhancing the proposed time synchronization services. Further, time synchronization service can be used to validate the time slots assigned by by the TDMA algorithm. Table 7.2 compares the different approaches for time synchronization.

7.5 Self-Stabilization

As defined in Chapter 1, starting from an arbitrary state, a self-stabilizing system recovers to states from where the system specification is satisfied [4]. Self-stabilization was motivated by Lamport in [51]. According to Lamport, self-stabilization is one of important concept in fault-tolerance. He regards Dijkstra's seminal work on self-stabilization [4] as one of the greatest in the field of fault-tolerance.

In this section, we discuss self-stabilization in the context of sensor networks. Specifically, in Section 7.5.1, we discuss self-stabilizing algorithms and, in Section 7.5.2, we discuss stabilization preserving transformation algorithms.

7.5.1 Self-Stabilizing Algorithms

Self-stabilization was formally introduced by Dijkstra in [4]. In this paper, Dijkstra presents three self-stabilizing algorithms for ensuring that the system is in a legitimate state. The system recovers to legitimate states in spite of initial state or arbitrary state corruptions. Note that the legitimate states depend on the problem specification.

Many self-stabilizing algorithms for distributed systems are proposed in the literature, especially for clock synchronization and communication protocols.

Clock synchronization. Some of the self-stabilizing solutions for clock synchronization can be found in [52–55]. The main difference among the algorithms proposed in these papers and the algorithms discussed in Section 7.4 is that underlying model is

different. In [56], Dolev presents a self-stabilizing clock synchronizing protocol for a general communication graph. Further, the author discusses impossibility results for general graphs. Specifically, the paper argues that in the presence of even a single faulty Byzantine process, it is impossible to design a digital clock synchronization protocol.

Communication protocols. Self-stabilization is important in communication protocols as transient faults are more common in computer networks. Further, with the advent of wireless communication, self-stabilization plays an important role in communication protocols. Some of the previous work on self-stabilizing solutions for communication protocols are reported in [57–59]. Most of these papers consider the classical problems on computer networks, such as, sequencing, routing, etc.

Self-Stabilizing Algorithms for Sensor Networks

In [60], Arora motivated the design of self-stabilizing algorithms in sensor networks. In his talk, he discussed the model of sensor networks and its larger scale deployment requirements. Further, he motivated the need for self-stabilization, since the sensor networks pose new challenges by introducing new classes of faults, for example, noisy data, link asymmetry, and regional failures. Furthermore, he discussed the progress made in design of algorithms for sensor networks. Specifically, he discussed the applications, the pursuer-evader problem [61] and the Line in the Sand (LITeS) project [12], where sensor networks are extensively helpful.

Pursuer-evader problem. A pursuer-evader problem is the problem of tracking the evaders with help of sensor networks. Sensors (typically, motes) help the pursuers track the evaders by maintaining a tracking tree which is always rooted at the evader. In this paper, a self-stabilizing solution for tracking the evader is provided, where the evader knows all the moves of the pursuer (i.e., the evaders are omniscient) and the

pursuer relies on the sensor network to provide the tracking information. Further, the paper provides information on how fast the evaders are tracked.

Line in the sand (LITeS). LITeS is a technology demonstration project for the DARPA's Networked Embedded Software Technology (NEST) program. In this project, sensors (motes) help in identification and classification of intruders (typically, a person, a soldier carrying a pistol/gun, a car, or a heavy vehicle) along a thick line. Sensors are deployed around this line in a rectangular grid. Magnetometer and micro-powered impulse radar (MIR) are used in this application. Whenever an intruder comes closer to this grid, the sensors send tracking messages (sensors values and the class of the intruder) to the repeater (at the corners of the grid). The repeater forwards the messages to the base station which does high level processing of the messages. The base station provides a visualization facility which displays the sensor grid and the intruders activity.

In both these applications, collision-free communication is very important. As discussed earlier, collisions are very frequent in sensor networks. Further, applications cannot rely on layers above MAC layer to provide transmission control, due to the resource constraints inherent in the sensors. Thus, a collision-free MAC protocol is necessary. In [26] and in Chapter 4, a new self-stabilizing TDMA protocol is proposed. Furthermore, the delay in delivering a message using TDMA is within the limits of the application requirements (cf. Chapter 5).

7.5.2 Stabilization Preserving Model Conversions

The existence of model conversion algorithms allows one to write programs in one model and later, transform it to another, typically, restrictive model. Further, it is preferred that if the original program is self-stabilizing, the transformed program

be self-stabilizing. In other words, transformation algorithms should be stabilization preserving. In [5], Dolev discusses some of the traditional model conversion algorithms, where programs written in central daemon are converted to distributed daemon, programs in shared memory model are converted to message-passing model, etc.

Atomicity refinements. In [18], Nesterenko and Arora propose a stabilization preserving transformations of programs written in an abstract model (i.e., shared memory model) to a concrete model (i.e., read/write model). Their solution is based on the stabilizing dining philosophers problem [29]. The main idea is to split the high atomicity action (i.e., read action of the program in shared memory model) into a sequence of low atomicity actions (i.e., sequence of individual read actions of the program in the read/write model). However, since the actions of different processes can interleave in the low atomicity program, a local mutual exclusion is necessary to ensure neighboring processes are not in the critical section (i.e., reading the state of one of its neighbors) simultaneously. Note that, similar strategy is used in our transformation algorithm, where a program in WAC model is transformed to a program in read/write model (cf. Chapter 6). Moreover, in [18], the authors provide extensions to their refinement algorithm, where they further refine the program in read/write model to message-passing model. Also, they show how their refinement algorithms can be extended to solve stabilization preserving semantics refinement.

As mentioned in Chapter 6, model conversions algorithms are also reported in [16, 17, 19–21]. The solutions are based on local mutual exclusion. In [19, 20], self-stabilizing local mutual exclusion algorithms are proposed. In [16, 17], the authors present algorithms that transform programs in serial model to programs in distributed

computing environments. With the help of a linear alternator, the concurrent actions of the transformed program are synchronized.

CHAPTER 8

Conclusion and Future Work

In this thesis, we presented stabilizing algorithms for collision-free communication using collision-free diffusion and TDMA. We presented four versions of our collision-free diffusion algorithm based on the ability of sensors to communicate with each other and their ability to interfere with each other. While the solutions were designed for a grid network, we showed how we can modify them to deal with failed sensors as well as with arbitrary topologies. With these modifications, our solutions deal with commonly occurring difficulties, e.g., failed sensors, sleeping sensors, unidirectional links, and unreliable links, in sensor networks.

The algorithms permit sensors to save power by turning off the radio completely as long as the remaining sensors remain connected. These sleeping sensors can periodically wake up, wait for one diffusion message from one of its neighbors and return to sleeping state. This will allow the sensors to save power as well as keep the clock synchronized with their neighbors. Moreover, the algorithms are stabilizing fault-tolerant. Thus, even if all sensors are deactivated for a long time causing arbitrary clock drift, our algorithm ensures that starting from such an arbitrary state, eventually the diffusion will complete successfully and TDMA will be restored.

Mote-connectivity protocol. One of the important issues in our algorithms is to determine the communication and interference range of a sensor. There are several ways to achieve this: For example, we can begin with the third version of our algorithm (cf. Section 3.3) where the communication range is 1 and the interference

range is greater than 1. Initially, we overestimate the interference range by considering the manufacturer specification about the ability of sensors to communicate with each other. Subsequently, we can use the biconnectivity experiments by Choi et al [62] to determine appropriate communication range and appropriate interference range. Given any two sensors, j and k, these results allow these sensors to determine the probability that j can communicate with k and the probability that k can communicate with j. Using these results, we can update the communication range and the interference range: j is in the communication range of k iff min(probability with which j can communicate with k, probability with which k can communicate with j) exceeds a certain threshold. And, j is in the interference range of k iff max(probability with which j can communicate with k, probability with which k can communicate with j) is less than a certain threshold. Using the approach in Chapter 4 for dealing with failed/sleeping sensors , we can communicate the communication range and interference range of all sensors to the initiator of the diffusion. The initiator can then change the communication range and interference range appropriately.

The initiator of a diffusion can handoff its responsibility to other sensors as the diffusion can be initiated by any sensor as long as only one sensor initiates it. Thus, the current initiator can designate another sensor as subsequent initiator if the current initiator has low battery or if the initiating responsibility is to be shared by multiple sensors.

Communication patterns. We considered three types of communication patterns, namely, broadcast, convergecast and local gossip that occur frequently in sensor networks. We showed how the TDMA service is optimized for each of the communication patterns. Thus, based on the types of communication pattern encountered in a given application, it is possible to optimize the TDMA service.

As discussed in Chapter 5, we recommend that if the application requirements are unknown, then the TDMA service for the local gossip be used. Towards this end, we observe that the period used for local gossip is twice that for the case of broadcast/convergecast. Hence, it is possible that initiator(s) of broadcast/convergecast suffer extra delay when the local gossip solution is used. However, once the initiator sends its message, subsequent relaying occurs quickly. This is due to the fact that the solution for local gossip also ensures that the communication patterns such as broadcast and convergecast incur small delays at intermediate sensors. Thus, the solution for local gossip provides substantial benefit to broadcast and convergecast. In fact, he TDMA solution optimized for local gossip also enables a sensor to send data in any given direction in such a way that the delay incurred by the data at intermediate sensors is small.

Implicit acknowledgements. We can combine the TDMA algorithm with previous work on *implicit* acknowledgments [12]. We expect that for known communication patterns such as broadcast, convergecast and local gossip, combining TDMA with implicit acknowledgments will be especially useful. In these communication patterns, when sensor j transmits a message to k, k is expected to retransmit it to its successor (unless k is the last sensor to receive that communication). Since message sent by k is broadcast to all its neighbors, j can also hear that message. Thus, the retransmission by k acts as an implicit acknowledgment for j. With TDMA, j can wait until the TDMA slot assigned to k; if k does not transmit in that slot, j can conclude that k did not receive its message. Thus, j can reduce the power spent in waiting for the implicit acknowledgment by listening to the radio only in the TDMA slot for k.

Model conversions for sensor networks. Further, as an application to the collision-free diffusion and TDMA algorithms, we considered a novel model of computation, *write all with collision* (WAC), and presented stabilization preserving transformations to (respectively, from) other distributed computation models. Specifically, we compared the WAC model of computation with read/write model (as well as shared memory model and message passing model) considered in the literature. We showed that it is possible to transform a program in read/write atomicity into a program in WAC atomicity, and vice versa. However, while transforming a program in read/write model into a program in WAC model, if the transformed program is deterministic and cannot use time then the transformed program is considerably slow; at most one process can execute at a time. We showed that for a deterministic, untimed algorithm, where processes cannot detect collisions and cannot perform redundant writes, this transformation is optimal. We identified transformations where concurrent executions are possible if the processes are allowed to perform redundant writes. Also, we showed that, in a timed model, it is possible to allow processes to execute concurrently. Thus, we find that if the processes do not have the ability to read then the ability to determine time consistently is important.

Our transformation algorithms are designed for the case where collisions are assumed to be undetectable. These transformations can be easily used in contexts where collisions are detectable. However, the issue of optimality for untimed systems is open for the case where processes can detect collision.

For timed model, if the topology of processes is fixed, our transformation algorithm preserves the stabilization property of the given program in read/write model. Further, the transformation of programs in WAC model to read/write model is also stabilization preserving. This transformation does not assume a fixed topology.

Future work. There are several questions raised by this work: First, an interesting question is how to determine the initial sensor that is responsible for initiating the diffusion. In some heterogeneous networks where some sensors are more powerful and more reliable, these powerful/reliable sensors can be chosen to be the initiators. Alternatively, during deployment of sensors (e.g., by dropping them from a plane), we can keep several potential initiators that communicate with each other directly and use the approach in [33, 34] so that one of them is chosen to be the initiator. Another interesting extension of this work is to combine TDMA service with previous work on security [63, 64], dynamic composition [65] and wireless reprogramming in MICA motes [3,6,11] to provide a secure and reliable dynamic composition of TinyOS components over a radio network.

Work on model conversions for sensor networks also raised several questions. One of these problems is to develop fault-tolerance preserving transformations from read/write model to WAC model. Since our algorithms require some offline setup (e.g., graph coloring), they cannot deal with topology changes. Based on the results in Chapter 6, we expect that such transformations may not be possible in untimed, deterministic systems. However, we expect that such transformations could be obtained for timed systems. One of the interesting extension to this work is to design primitives that would allow us to identify transformations where concurrent executions are possible. In this thesis, we identified two such primitives, time and redundant writes. Another interesting extension of this work is to develop programs in WAC model that permit concurrent execution for untimed systems using the knowledge about the speed of processes. Furthermore, another extension is to weaken the assumption about rate of change for clocks in timed systems and allow a clock drift among processes.

BIBLIOGRAPHY

[1] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002.

[2] D. Estrin, R. Govindan, J. Heidmann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. *In Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking (MobiCOM)*, August 1999.

[3] D. E. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo. A network-centric approach to embedded software for tiny devices. In *EMSOFT*, volume 2211 of *Lecture Notes in Computer Science*, pages 97–113. Springer, 2001.

[4] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11), 1974.

[5] S. Dolev. *Self-Stabilization.* The MIT Press, 2000.

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. Pister. System architecture directions for network sensors. *In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, November 2000.

[7] J. D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, Rockville, MD, 1984.

[8] N. Deo. *Graph Theory with Application to Engineering and Computer Science.* Prentice-Hall, June 1974.

[9] A. Arora and M. Gouda. Distributed reset. *IEEE Transactions on Computers*, 43(9):1026–1038, 1994.

[10] S. S. Kulkarni and A. Arora. Multitolerance in distributed reset. *Chicago Journal of Theoretical Computer Science*, 1998.

[11] Crossbow Technology, Inc. *Mote In-Network Programming User Reference Version 20030315*, 2003. Available at: `http://www.xbow.com/Support/Support_pdf_files/Xnp.pdf`.

[12] The Ohio State University NEST Team. A Line in the Sand (LITeS), A DARPA-NEST Field Experiment. DARPA/US-SOCOM Project Demonstration, Tampa, Florida, August 2003. Experiment report available online at: `http://www.cis.ohio-state.edu/siefast/nest/nest_webpage/ALineInTheSand.html`.

[13] D. Gay, D. Culler, and P. Levis. *nesC Language Reference Manual*, September 2003. Available at: `http://nescc.sourceforge.net/papers/nesc-ref.pdf`.

[14] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. *In Proceedings of the Programming Language Design and Implementation (PLDI)*, June 2003.

[15] G. Simon, P. Volgyesi, M. Maroti, and A. Ledeczi. Simulation-based optimization of communication protocols for large-scale wireless sensors networks. *In Proceedings of the IEEE Aerospace Conference*, March 2003. Simulator (*Prowler*) available at: `http://www.isis.vanderbilt.edu/projects/nest/prowler`.

[16] M. Gouda and F. Haddix. The linear alternator. *In Proceedings of the Third Workshop on Self-stabilizing Systems*, pages 31–47, 1997.

[17] M. Gouda and F. Haddix. The alternator. *In Proceedings of the Fourth Workshop on Self-stabilizing Systems*, pages 48–53, 1999.

[18] M. Nesterenko and A. Arora. Stabilization-preserving atomicity refinement. *Journal of Parallel and Distributed Computing*, 62(5):766–791, 2002.

[19] H. Kakugawa and M. Yamashita. Self-stabilizing local mutual exclusion on networks in which process identifiers are not distinct. *In Proceedings of the 21st Symposium on Reliable Distributed Systems (SRDS)*, pages 202–211, 2002.

[20] G. Antonoiu and P. K. Srimani. Mutual exclusion between neighboring nodes in an arbitrary system graph tree that stabilizes using read/write atomicity. *In Euro-par'99 Parallel Processing, Springer-Verlag*, LNCS:1685:824–830, 1999.

[21] K. Ioannidou. Transformations of self-stabilizing algorithms. *In Proceedings of the 16th International Conference on Distributed Computing (DISC), Springer-Verlag*, LNCS:2508:103–117, October 2002.

[22] S. S. Kulkarni and U. Arumugam. Atomicity refinement for write-all-with-collision model. Technical Report MSU-CSE-03-13, Department of Computer Science, Michigan State University, June 2003.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, September 2001.

[24] G. Chartrand and O. R. Oellermann. *Applied and Algorithmic Graph Theory*. McGraw-Hill Inc., 1993.

[25] S. Ghosh and M. H. Karaata. A self-stabilizing algorithm for coloring planar graphs. *Distributed Computing*, 7(1):55–59, 1993.

[26] S. S. Kulkarni and U. Arumugam. Collision-free communication in sensor networks. *In Proceedings of the Sixth Symposium on Self-stabilizing Systems (SSS), Springer*, LNCS:2704:17–31, June 2003.

[27] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[28] N. Lynch and F. Vaandrager. Forward and backward simulations – Part 1: Untimed systems. *Information and Computation*, 121(2):214–233, September 1995. Also, Technical Memo MIT/LCS/TM-486.b, Laboratory for Computer Science, Massachusetts Institute of Technology.

[29] K. M. Chandy and J. Misra. The drinking philosophers problem. *ACM Transactions on Programming Languages and Systems*, 6(4):632–646, 1984.

[30] J. Couvreur, N. Francez, and M. Gouda. Asynchronous unison. *In Proceedings of the International Conference on Distributed Computing Systems*, pages 486–493, 1992.

[31] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A scalable and robust communication paradigm for sensor networks. *In Proceedings of the Sixth International Conference on Mobile Computing and Networks (MobiCOM)*, August 2000.

[32] E. Kranakis, D. Krizanc, and A. Pelc. Fault-tolerant broadcasting in radio networks. *Journal of Algorithms*, 39(1):47–67, April 2001.

[33] L. Gasieniec, A. Pelc, and D. Peleg. The wakeup problem in synchronous broadcast systems. *SIAM Journal of Discrete Mathematics*, 14(2):207–222, 2001.

[34] B. S. Chlebus, L. Gasieniec, A. Gibbons, A. Pelc, and W. Rytter. Deterministic broadcasting in ad hoc radio networks. *Distributed Computing*, 15(1):27–38, 2002.

[35] R. Rom and M. Sidi. *Multiple Access Protocols: Performance and Analysis.* Springer-Verlag, 1989. Also available at: `http://www.comnet.technion.ac.il/rom/PDF/MAP.pdf`.

[36] A. J. Viterbi. *CDMA: Principles of Spread Spectrum Communication.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, 1995.

[37] A. Woo and D. Culler. A transmission control scheme for media access in sensor networks. *In Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking*, pages 221–235, 2001.

[38] V. Claesson, H. Lonn, and N. Suri. Efficient TDMA synchronization for distributed embedded systems. *In Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 198–201, October 2001.

[39] K. Sohrabi and G. J. Pottie. Performance of a novel self-organization protocol for wireless ad-hoc sensor networks. *In Proceedings of the IEEE Vehicular Technology Conference*, pages 1222–1226, 1999.

[40] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, October 2002.

[41] K. Arisha, M. Youssef, and M. Younis. Energy-aware TDMA-based MAC for sensor networks. *In Proceedings of the IEEE Workshop on Integrated Management of Power Aware Communications, Computing and Networking (IMPACCT)*, May 2002.

[42] S. Singh and C. S. Raghavendra. PAMAS: Power aware multi-access protocol with signaling for ad hoc networks. *ACM Computer Communications Review*, July 1998.

[43] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. *In Proceedings of the IEEE Infocom*, 2002.

[44] P. Karn. MACA - A new channel access protocol for packet radio. *In Proceedings of the ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pages 134–140, 1990.

[45] C-S. Wu and V. O. K. Li. Receiver-initiated busy-tone multiple access in packet radio networks. *In Proceedings of the ACM SIGCOMM workshop*, 17(5):336–342, May 1997.

[46] F. A. Tobagi and L. Kleinrock. Packet switching in radio channels: Part II - the hidden terminal problem in carrier sensor multiple-access models and the busy-tone solution. *In Proceedings of the IEEE Transactions on Communications*, COM-23(12):1417–1433, 1975.

[47] J. Elson and R. Romer. Wireless sensor networks: A new regime for time synchronization. *In Proceedings of the First Workshop on Hot Topics in Networks (HotNets-1)*, October 2002.

[48] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. *In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Parallel and Distributed Computing Issues in Wireless and Mobile Computing*, April 2001.

[49] T. Herman. NestArch: Prototype time synchronization service. NEST Challenge Architecture. Available at: `http://www.ai.mit.edu/people/sombrero/nestwiki/index/ComponentTimeSync`, January 2003.

[50] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[51] L. Lamport. Solved problems, unsolved problems and non-problems in concurrency. *In Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–11, 1984, invited address.

[52] A. Arora, S. Dolev, and M. G. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1:11–18, 1991.

[53] A. Ciuffoletti. Using simple diffusion to synchronize clocks in a distributed system. *In Proceedings of the 14th International Conference on Distributed Computing Systems (ICDCS)*, pages 484–491, 1994.

[54] S. Chandrasekar and P. K. Srimani. A self-stabilizing algorithm to synchronize digital clocks in a distributed system. *Computers and Electrical Engineering*, 20(6):439–444, 1994.

[55] A. Ciuffoletti. Self-stabilizing clock synchronization in a hierarchical network. *In Proceedings of the Fourth Workshop on Self-Stabilizing Systems*, pages 86–93, 1999.

[56] S. Dolev. Possible and impossible self-stabilizing digital clock synchronization in general graphs. *Journal of Real-Time Systems*, 12(1):95–107, 1997.

[57] J. M. Spinelli. Self-stabilizing network communication protocols. *Communication, Control, and Signal Processing*, pages 141–147, 1990.

[58] J. M. Spinelli and R. G. Gallager. Event driven topology broadcast without sequence numbers. *IEEE Transactions on Communications*, 37:468–474, 1989.

[59] S. Dolev, D. K. Pradhan, and J. L. Welch. Modified tree structure for location management in mobile environments. *Computer Communications*, 19:335–345, 1996.

[60] A. Arora. Taking stabilization to the masses: Problems, Opportunities, and Progress. *Sixth Symposium on Self-Stabilizing Systems (SSS)*, June 2003, invited talk.

[61] M. Demirbas, A. Arora, and M. G. Gouda. A pursuer-evader game for sensor networks. *In Proceedings of the Sixth Symposium on Self-Stabilizing Systems (SSS), Springer*, LNCS:2704:1–16, June 2003.

[62] Y. Choi, M. Gouda, M. C. Kim, and A. Arora. The mote connectivity protocol. Technical Report TR03-08, Department of Computer Sciences, The University of Texas at Austin, 2003.

[63] C. Karlof, N. Sastry, and D. Wagner. *TinySec 0.91: User Manual*, February 2003. Available at: `http://www.cs.berkeley.edu/~nks/tinysec/tinysec.pdf`.

[64] S. S. Kulkarni, M. G. Gouda, and A. Arora. Instantiating security in mobile networks. Technical Report MSU-CSE-03-6, Department of Computer Science, Michigan State University, March 2003.

[65] S. S. Kulkarni, K. N. Biyani, and U. Arumugam. Composing distributed fault-tolerance components. *In Proceedings of the International Conference on Dependable Systems and Networks (DSN), Supplemental Volume, Workshop on Principles of Dependable Systems*, pages W127–W136, June 2003.