A DECENTRALIZED SINGLE-HOP TIME SYNCHRONIZATION PROTOCOL BASED ON ASYNCHRONOUS CONSENSUS CONTROL IN WIRELESS SENSOR NETWORKS

BY

RAMADAN ABDUL-RASHID

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

TELECOMMUNICATION ENGINEERING

DECEMBER 2017

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **RAMADAN ABDUL-RASHID** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN TELECOMMUNICATION ENGINEERING**.

Thesis Committee

Dr. Ali Al-Shaikhi (Adviser)

Dr. Ahmad Masoud (Member)

Dr. Samir Alghadhban (Member)

Dr. Ali Al-Shaikhi
Department Chairman

Dr. Salam A. Zummo
Dean of Graduate Studies

7/1/18
Date

©Ramadan Abdul-Rashid
2017

To my Beloved Zeena

ACKNOWLEDGMENTS

All thanks and praises are due to Allah, the Creator and Sustainer of the universe. May He be glorified for the innumerable blessings and mercies bestowed upon me. I would like to express my utmost gratitude to my thesis advisor, Dr. Ali Ahmad Al-Shaikhi for his continuous directions, motivations and support. I couldnt hope for a more suited advisor to make my Masters study such an exciting and inspirational journey. Special thanks goes to Dr. Ahmad Abdallah Masoud for his invaluably astute guidance, critiques and contributions to my work which has greatly enrich my research experience. I would also like to show my appreciation to Dr. Samir Nasser Alghadhban for his inspirations, advices and warm hospitality during my thesis work and stay at KFUPM. I ask Allah to shower his blessings on you all in this world and the next. My thanks also go to Dr. Bo Liu of the Center for Energy and Geo Processing (CeGP) at the Electrical Engineering Department for our fruitful discussions in relation to my thesis research. I would also like to thank my family for their encouragement and support, especially to my wife, Zeena Abdul-Nasir Mukhtar, my mother, Sakinatu Yakubu and my father, Abdul-Rashid Abubakar. Finally, my heartfelt gratitude goes to the Electrical Engineering Department of King Fahd University of Petroleum and Minerals for granting me the opportunity to study in such a serene environment.

TABLE OF CONTENTS

ACKNOWLEDGEMENT **iii**

LIST OF TABLES **ix**

LIST OF FIGURES **xi**

ABSTRACT (ENGLISH) **xvi**

ABSTRACT (ARABIC) **xix**

CHAPTER 1 WIRELESS SENSOR NETWORKS OVERVIEW **1**

1.1 Architecture 2

1.1.1 Node Architecture 3

1.2 Topologies 6

1.3 Wireless Standards 6

1.3.1 ZigBee / ZigBee PRO / ZigBee IP 8

1.4 Practical WSN Schemes 10

1.4.1 TinyOS and NesC Overview 10

1.4.2 Memsic Mote Overview 11

1.5 Applications of Wireless Sensor Networks 14

1.6 Network Establishment and Management Services for WSNs . . . 16

1.7 Summary 17

CHAPTER 2 TIME SYNCHRONIZATION IN WIRELESS SENSOR NETWORKS **19**

2.1 Background 20
    2.1.1 The Time Synchronization Problem 20
    2.1.2 The Need for WSN Time Synchronization 22
2.2 Node Clock Modeling and Synchronization 23
    2.2.1 Hardware Clock Models 23
    2.2.2 Logical Clock Models 28
2.3 Performance Metrics for Time Synchronization Protocols 29
    2.3.1 Accuracy 30
    2.3.2 Precision 33
    2.3.3 Cost 34
    2.3.4 Fault Tolerance (Robustness) 34
    2.3.5 Scalability 35
2.4 Review of Time Synchronization Protocols 35
    2.4.1 Review of Centralized Protocols 37
    2.4.2 Review of Distributed Protocols 44
    2.4.3 Comparison Between Distributed and Centralized Protocols 51
2.5 Summary 53

CHAPTER 3 PROBLEM DEFINITION, STATEMENT AND FORMULATION 55

3.1 Problem Definition and Statement 55
3.2 Suggested Synchronization Procedure 57
    3.2.1 Node Local Time Update 58
    3.2.2 Dip Phenomenon in Suggested Synchronizer 62
    3.2.3 Criterion for Stopping in Dip Region 63
    3.2.4 Local Clock Reset and Resynchronization 64
    3.2.5 Generalized Algorithm for Suggested Synchronization Procedure 66
3.3 Formulation and Methodology 68

3.3.1 Accuracy-Enhanced Method for Time Synchronization in WSN using Synchronous Average Consensus Control . . . 69
3.3.2 Drawbacks in the Synchronous Local Node Time Update . 70
3.3.3 Suggested Asynchronous Local Node Time Update 70
3.4 Thesis Contributions 72

CHAPTER 4 ACCURACY-ENHANCED METHOD FOR TIME SYNCHRONIZATION IN WSN USING SYNCHRONOUS AVERAGE CONSENSUS CONTROL **75**

4.1 Introduction 75
4.2 Accuracy-Enhanced Method for Time Synchronization: Concept . 76
4.2.1 System Stability 79
4.2.2 Hypothesis for Accuracy-Enhancement 80
4.3 Evaluation Methodology for Simulations and Practical Experimentation 83
4.3.1 Description and Specifications of Simulations 83
4.3.2 Framework for Practical Experimentation 84
4.4 Test Networks 85
4.4.1 4-Node Network 86
4.4.2 9-Node Network 90
4.4.3 16-Node Network 93
4.4.4 Synchronous Time Update (STU): Implementation Algorithm 98
4.5 Accuracy-Enhanced Method for Time Synchronization: Results . 100
4.5.1 Simulation Results: Error Profiles for Varying Added Parameter 101
4.5.2 Practical Results: Error Profiles for Varying Added Parameter 105
4.5.3 Relationship Between Parameter, e and Minimum Error in Dip Region 109
4.6 Summary 110

CHAPTER 5 LOCAL NODE TIME UPDATE USING ASYNCHRONOUS CONSENSUS: METHODS AND RESULTS 113
5.1 Introduction 113
5.2 Proposed Asynchronous Methods for Local Node Time Update . . 114
5.2.1 Timed Sequential Asynchronous Update (TSAU) 115
5.2.2 Unidirectional Asynchronous Flooding (UAF) 116
5.2.3 Bidirectional Asynchronous Flooding (BAF) 120
5.2.4 Performance Metrics for Evaluation 123
5.3 Simulation Evaluation of Proposed Methods of Local Time Update 125
5.3.1 Simulation Results for Timed Sequential Asynchronous Update (TSAU) 125
5.3.2 Simulation Results for Unidirectional Asynchronous Flooding (UAF) 130
5.3.3 Simulation Results for Bidirectional Asynchronous Flooding (BAF) 136
5.3.4 Simulation Comparison between Proposed Methods of Local Time Update 142
5.4 Practical Evaluation of Proposed Methods of Local Time Update 149
5.4.1 Practical Results for Timed Sequential Asynchronous Update (TSAU) 150
5.4.2 Practical Results for Unidirectional Asynchronous Flooding (UAF) 152
5.4.3 Practical Results for Bidirectional Asynchronous Flooding (BAF) 154
5.4.4 Practical Comparison between Proposed Methods of Local Time Update 158
5.5 Simulation on Network with Stochastic Link Connectivity 163
5.6 Summary 167
CHAPTER 6 EXPERIMENTAL PERFORMANCE EVALUA-

TION: COMPARISON WITH OTHER PROTOCOLS 170

6.1 Introduction 170

6.1.1 Test Networks 172

6.1.2 Experiment Parameters 172

6.1.3 Energy Consumption and Memory Requirements 173

6.1.4 Local and Global Synchronization Errors 179

6.2 Summary 192

CHAPTER 7 CONCLUSION AND FUTURE WORK 195

7.1 Conclusion 195

7.2 Future Work 198

REFERENCES 199

VITAE 209

LIST OF TABLES

1.1 Comparison Between Bluetooth, Wifi and Zigbee 8
1.2 Categorical Applications of Wireless Sensor Networks 15

2.1 Comparison Between Distributed and Centralized Protocols . . . 52

4.1 Parameter Specifications for MATALB Simulation 84
4.2 Simulation Results Summary of Number Iterations and Error Values 102
4.3 Simulation Results Summary of Number Iterations and Error Values 103
4.4 Simulation Results Summary of Number Iterations and Error Values 105
4.5 Practical Results Summary of Number Iterations and Error Values 106
4.6 Practical Results Summary of Number Iterations and Error Values 107
4.7 Practical Results Summary of Number Iterations and Error Values 109

5.1 4 Node Network Dip Region Results Comparison for Timed Sequential Asynchronous Update 127
5.2 9 Node Network Dip Region Results Comparison for Timed Sequential Asynchronous Update 130
5.3 16 Node Network Dip Region Results Comparison for Timed Sequential Asynchronous Update 132
5.4 4 Node Network Dip Region Results Comparison for Unidirectional Asynchronous Flooding 134
5.5 9 Node Network Dip Region Results Comparison for Unidirectional Asynchronous Flooding 137

5.6 16 Node Network Dip Region Results Comparison for Unidirectional Asynchronous Flooding 139
5.7 4 Node Network Dip Region Results Comparison for Bidirectional Asynchronous Flooding 141
5.8 9 Node Network Dip Region Results Comparison for Bidirectional Asynchronous Flooding 144
5.9 16 Node Network Dip Region Results Comparison for Bidirectional Asynchronous Flooding 146
5.10 4 Node Grid Network Dip Region Results Comparison 151
5.11 9 Node Grid Network Dip Region Results Comparison 152
5.12 16 Node Grid Network Dip Region Results Comparison 154
5.13 4 Node Grid Network Dip Region Results Comparison 155
5.14 9 Node Grid Network Dip Region Results Comparison 156
5.15 16 Node Grid Network Dip Region Results Comparison 156
5.16 4 Node Grid Network Dip Region Results Comparison 158
5.17 9 Node Grid Network Dip Region Results Comparison 159
5.18 16 Node Grid Network Dip Region Results Comparison 161

6.1 Memory Requirements, CPU Overhead, and Synchronization Message Length of FTSP, FloodPISync, EGTSP, AvgPISync, TSAU, UAF and BAF 176
6.2 Current Consumption by MicaZ Nodes [1] 178
6.3 Global and Local Error Comparison for Grid Topology 182
6.4 Global and Local Error Comparison for Line Topology 188

LIST OF FIGURES

1.1 Typical Node Architecture 4
1.2 Typical Sensor Network Arrangement 4
1.3 Typical WSN Topologies 7
1.4 MicaZ Mote attached to MIB520 Board[2] 12
1.5 MDA100 Sensor Board[3] 13
1.6 MIB520 USB Interface Board [2] 14

2.1 Estimating Angle of Sound Impingement on an Array 21
2.2 An illustration of Aging Effects of Node Clocks over Time (days) . 26
2.3 The Relationship between Hardware Clock and Logical Clock . . 28
2.4 Nodes Clock of Varying Drift Rates 29

3.1 Block Diagram of Suggested Synchronization Procedure 59
3.2 Network of N Nodes and L Links 59
3.3 Transient (Dip) and Steady State Behavior of Node Clocks with respect to Gateway Node Clock 63
3.4 Suggested Clock Setup for Node Synchronization 67

4.1 4-Node Grid Topology 86
4.2 4-Node Random Topology 88
4.3 4-Node Hexagonal Topology 89
4.4 9-Node Grid Topology 91
4.5 9-Node Random Topology 92
4.6 9-Node Hexagonal Topology 93

4.7 16-Node Grid Topology 94
4.8 16-Node Random Topology 96
4.9 16-Node Grid Topology 97
4.10 Simulation Error Profiles for Varying Added Parameter for 4 Node Grid Topology 101
4.11 Simulation Error Profiles for Varying Added Parameter for 9 Node Grid Topology 103
4.12 Simulation Error Profiles for Varying Added Parameter for 16 Node Grid Topology 104
4.13 Practical Error Profiles for Varying Added Parameter for 4 Node Grid Topology 106
4.14 Practical Error Profiles for Varying Added Parameter for 9 Node Grid Topology 107
4.15 Practical Error Profiles for Varying Added Parameter for 16 Node Grid Topology 108
4.16 Relationship Between Added Parameter,e and Minimum Error in Dip Region for Simulations of 4, 9 and 16 Node Networks 109
4.17 Relationship Between Added Parameter,e and Minimum Error in Dip Region for Experiments on 4, 9 and 16 Node Networks 110

5.1 Asynchronous Wake-up Cycle and Operation Mechanism for TSAU 116
5.2 Asynchronous Wake-up Cycle and Operation Mechanism for UAF 118
5.3 Asynchronous Wake-up Cycle and Operation Mechanism for BAF 122
5.4 Node Time and Error Profiles for Grid Topology 126
5.5 Node Time and Error Profiles for Random Topology 126
5.6 Node Time and Error Profiles for Hexagonal Topology 127
5.7 Node Time and Error Profiles for Grid Topology 128
5.8 Node Time and Error Profiles for Random Topology 128
5.9 Node Time and Error Profiles for Hexagonal Topology 129
5.10 Node Time and Error Profiles for Grid Topology 129

5.11 Node Time and Error Profiles for Random Topology 131
5.12 Node Time and Error Profiles for Hexagonal Topology 131
5.13 Node Time and Error Profiles for Grid Topology 133
5.14 Node Time and Error Profiles for Random Topology 133
5.15 Node Time and Error Profiles for Hexagonal Topology 134
5.16 Node Time and Error Profiles for Grid Topology 135
5.17 Node Time and Error Profiles for Random Topology 135
5.18 Node Time and Error Profiles for Hexagonal Topology 136
5.19 Node Time and Error Profiles for Grid Topology 136
5.20 Node Time and Error Profiles for Random Topology 138
5.21 Node Time and Error Profiles for Hexagonal Topology 138
5.22 Node Time and Error Profiles for Grid Topology 140
5.23 Node Time and Error Profiles for Random Topology 140
5.24 Node Time and Error Profiles for Hexagonal Topology 141
5.25 Node Time and Error Profiles for Grid Topology 142
5.26 Node Time and Error Profiles for Random Topology 142
5.27 Node Time and Error Profiles for Hexagonal Topology 143
5.28 Node Time and Error Profiles for Grid Topology 143
5.29 Node Time and Error Profiles for Random Topology 145
5.30 Node Time and Error Profiles for Hexagonal Topology 145
5.31 Simulation Comparison of Minimum Errors, $\mathbb{E}_{min}^{dip}$ for TSAU, UAF and BAF For Varying Network Sizes 147
5.32 Simulation Comparison of Communication Cycles, k_{min}^{dip} for TSAU, UAF and BAF For Varying Network Sizes 148
5.33 Simulation Comparison of Variances in Communication Cycles, $\mathbb{V}_{k_{min}^{dip}}$ for TSAU, UAF and BAF For Varying Network Sizes 149
5.34 Practical Node Time and Error Profiles for 4 Grid Topology . . . 150
5.35 Practical Node Time and Error Profiles for 9 Grid Topology . . . 151
5.36 Practical Node Time and Error Profiles for 16 Grid Topology . . . 153
5.37 Practical Node Time and Error Profiles for 4 Grid Topology . . . 153

5.38 Practical Node Time and Error Profiles for 9 Grid Topology . . . 155
5.39 Practical Node Time and Error Profiles for 16 Grid Topology . . . 157
5.40 Practical Node Time and Error Profiles for 4 Grid Topology . . . 157
5.41 Practical Node Time and Error Profiles for 9 Grid Topology . . . 158
5.42 Practical Node Time and Error Profiles for 16 Grid Topology . . . 160
5.43 Practical Comparison of Minimum Errors, $\mathbb{E}_{min}^{dip}$ for TSAU, UAF and BAF For Varying Network Sizes 162
5.44 Practical Comparison of Communication Cycles, k_{min}^{dip} for TSAU, UAF and BAF For Varying Network Sizes 163
5.45 Practical Comparison of Variances in Communication Cycles, $\mathbb{V}_{k_{min}^{dip}}$ for TSAU, UAF and BAF For Varying Network Sizes 164
5.46 Network for Channel Availability Test 165
5.47 Node Error profile for the Test Network for Different Probabilities of Channel Availability for TSAU 166
5.48 Node Error profile for the Test Network for Different Probabilities of Channel Availability for UAF 166
5.49 Node Error profile for the Test Network for Different Probabilities of Channel Availability for BAF 167

6.1 Distribution of Sensor Nodes for Practical Experiments 173
6.2 TimeSync Packet Payload Field Descriptions for FTSP, FloodPISync, EGTSP, TSAU, UAF and BAF 175
6.3 UAF Neighborhood and Global Synchronization Error for Grid Topology 183
6.4 TSAU Neighborhood and Global Synchronization Error for Grid Topology 184
6.5 FloodPISync Neighborhood and Global Synchronization Error for Grid Topology 184
6.6 AvgPISync Neighborhood and Global Synchronization Error for Grid Topology 185

6.7 BAF Neighborhood and Global Synchronization Error for Grid Topology 185
6.8 FTSP Neighborhood and Global Synchronization Error for Grid Topology 186
6.9 EGTSP Neighborhood and Global Synchronization Error for Grid Topology 186
6.10 TSAU Neighborhood and Global Synchronization Error for Line Topology 189
6.11 UAF Neighborhood and Global Synchronization Error for Line Topology 189
6.12 FloodPISync Neighborhood and Global Synchronization Error for Line Topology 190
6.13 AvgPISync Neighborhood and Global Synchronization Error for Line Topology 190
6.14 EGTSP Neighborhood and Global Synchronization Error for Line Topology 191
6.15 BAF Neighborhood and Global Synchronization Error for Line Topology 191
6.16 FTSP Neighborhood and Global Synchronization Error for Line Topology 192

THESIS ABSTRACT

NAME:	Ramadan Abdul-Rashid
TITLE OF STUDY:	A Decentralized Single-Hop Time Synchronization Protocol based on Asynchronous Consensus Control in Wireless Sensor Networks
MAJOR FIELD:	Telecommunication Engineering
DATE OF DEGREE:	December 2017

Time synchronization is a crucial requirement in Wireless Sensor Networks (WSNs) in almost all applications. It is the synchronization of the virtual clock of each network node to the clock of the gateway node of the WSNs. This thesis concerns with the design of simple time synchronization protocol based on consensus control concept suited for operation in harsh environments using single hop communication in distributed WSNs. The thesis uses the averaging algorithm used for synchronous synchronization as the main algorithm. First, the thesis presents a modified averaging algorithm for synchronous synchronization that improves the accuracy of the averaging algorithm. The modified algorithm is termed Accuracy-Enhanced Method for Time Synchronization in WSN using syn-

chronous average censuses control. It provides a trade-off between the convergence time and synchronization accuracy. This method is inspired by the realization that, the state equation representing the synchronization error with respect to the gateway time can be reduced further by reducing the forced response term by introducing a new parameter, a fraction of the time communication cycle, in the local time update of each node. As this fraction approaches one, it is observed through simulations and practical experiments that it will lead to orders of magnitude less error in the dip region but with a corresponding increase in the number of communications instances needed to reach the dip synchronization error. Despite its excellent performance and simplicity, the averaging protocol is impractical since it assumes synchronous node time update. To overcome this problem, the thesis proposes three practical asynchronous schemes named Timed Sequential Asynchronous Update (TSAU), Unidirectional Asynchronous Update (UAF) and Bidirectional Asynchronous Flooding (BAF). All these proposed methods are decentralized and light weight by design. In TSAU, time updates are carried out in atomic fashion sequentially, whereas in UAF nodes within the same proximity to the gateway update simultaneously. UAF requires regulation of asynchronous activations but BAF removes the need for regulation and is adaptive to topology dynamics. These proposed methods were evaluated using simulations and practical experiments on the MicaZ sensor node platform and present a comprehensive evaluation of the energy consumption, memory requirements, convergence time, local and global synchronization errors of our proposed methods against FTSP, *EGTSP, FloodPISync and AvgPISync. The proposed schemes consume about 80% less power and approximately 35% less memory allocation than FTSP and EGTSP but have similar energy and memory requirements as FloodPISync and AvgPISync. Further, the proposed methods outperform all these protocols in terms of convergence time, global and local synchronization errors.*

ملخص الرسالة

الاسم الكامل: رمضان عبد الرشيد
عنوان الرسالة: بروتوكولٌ لامركزيٌّ ذو قفزةٍ واحدةٍ متزامنةٍ وقتياً مرتكزٌ على التحكم التوافقيّ غير المتزامن في شبكات الاستشعار اللاسلكية
التخصص: هندسة الاتصالات
تاريخ الدرجة العلمية: ديسمبر ٢٠١٧

يعتبر التزامن الوقتي مطلباً حاسماً في شبكات الاستشعار اللاسلكية (WSNs) في جميع التطبيقات تقريباً. و هو تزامن الساعة الافتراضية لكل عقدة شبكة مع ساعة عقدة البوابة لشبكات الاستشعار اللاسلكية. تهتم هذه الأطروحة بالتصميم البسيط لبروتوكول تزامنٍ وقتيٍّ مرتكزٍ على مفهوم التحكم التوافقي، مناسبٍ للعمل في بيئاتٍ قاسيةٍ باستخدام اتصالات القفزة الواحدة في شبكات الاستشعار اللاسلكية الموزعة. تستخدم الأطروحة خوارزمية المتوسط المستخدمة في المزامنة التوافقية كخوارزميةٍ رئيسيةٍ. أولاً، تقدم الأطروحة خوارزمية المتوسط المعدّلة للمزامنة التوافقية و التي تحسن دقة خوارزمية المتوسط. يُطلق على الخوارزمية المعدلة "طريقة تحسين الدقة للمزامنة الوقتية في شبكات الاستشعار اللاسلكية باستخدام تحكم التعدادات التوافقية المتزامنة". و توفر مقايضةً بين وقت التقارب و دقة التزامن. هذه الطريقة مستوحاة من إدراك أن معادلة الحالة التي تمثل خطأ التزامن بالنسبة لوقت البوابة يمكن تقليلها بتقليل مدة الاستجابة القسرية من خلال إدخال معاملٍ جديدٍ، جزءٍ من دورة وقت الاتصال، في تحديث الوقت المحليّ لكل عقدة. مع اقتراب هذا الجزء من واحد، يُلاحظ من خلال عمليات المحاكاة و التجارب العملية أنه سيؤدي إلى خطأٍ ذي قيمةٍ أسيةٍ أقل في منطقة الانخفاض، و لكن مع زيادةٍ مقابلةٍ في عدد حالات الاتصالات اللازمة للوصول إلى خطأ تزامن الانخفاض. بالرغم من أدائه الممتاز و بساطته، فإن بروتوكول المتوسط ليس عملياً لأنه يفترض تحديث الوقت التزامني للعقدة. للتغلب على هذه المشكلة، تقترح الأطروحة ثلاثة مخططاتٍ عمليةٍ غير متزامنةٍ تدعى: التحديث غير المتزامن للتوقيت التسلسلي (TSAU)، و التحديث غير المتزامن أحادي الاتجاه (UAF)، و الفيض غير المتزامن ثنائي الاتجاه (BAF). جميع هذه الطرق المقترحة هي لامركزيةٌ و خفيفة الوزن من حيث التصميم. في TSAU، تحديثات الوقت تُجرى تتابعياً بطريقةٍ ذريةٍ، في حين أن عقد UAF الواقعة بقرب البوابة يتم تحديثها في نفس الوقت. تتطلب طريقة UAF تنظيماً للتفعيلات غير المتزامنة، و لكن طريقة BAF تزيل الحاجة للتنظيم و هي متكيفةٌ لديناميكية البُنية. تم تقييم هذه الطرق المقترحة باستخدام المحاكاة و التجارب العملية على برنامج مستشعر ميكاز MicaZ، و تم تقديم تقييمٍ شاملٍ لاستهلاك الطاقة، و متطلبات الذاكرة، و وقت التقارب، و أخطاء التزامن المحلية و العالمية لطرقنا المقترحة مقارنةً بطرق FTSP، و EGTSP، و FloodPISync، و AvgPISync. المخططات المقترحة تستهلك حوالي 80% طاقةً أقل و تقريباً 35% تخصيص ذاكرةٍ أقل من FTSP و EGTSP، و لكن لديها متطلباتٍ مماثلةٍ من الطاقة و الذاكرة لكلٍ من FloodPISync و AvgPISync. علاوةً على ذلك، فإن أداء الطرق المقترحة يفوق أداء جميع هذه البروتوكولات من حيث وقت التقارب، و أخطاء التزامن المحلية و العالمية.

CHAPTER 1

WIRELESS SENSOR NETWORKS OVERVIEW

Wireless Sensor Networks (WSNs) are infrastructure-less networks composed of small sensors that are deployed in remote locations to sense particular conditions and send information pertaining to these conditions to a Central Control Room (CCR). WSN are ad hoc in nature and are often deployed with a large number of nodes. They work with limited resources and are mostly composed of cheap nodes that are rarely replaced. WSNs have an endless number of applications. They are often used in military defense to monitor national borders or munition deployment of adversaries. They are also normally used by environmentalists to monitor environmental changes such as temperature and humidity in certain regions. In industries, WSNs can sense temperature and/or pressure of certain devices and can be used to instrument an industrial plant. WSNs have certain limitations such as low communication range, small node battery sizes, and network structures which require very resource efficient algorithms. To successfully deploy and operate a WSN, a myriad of materials and methods are required. This chapter delves into a general review of wireless sensor networks and explains typical architectures of sensor networks. This includes an explanation of the architecture of sensor nodes and outlining node arrangements and topologies of typical WSNs. The most common wireless protocols adopted for setting up WSNs are then presented. The operating system, programming language, and the layout of the sensor node platform adopted in this thesis are presented. Typical applications of wireless sensor networks are then listed followed by some typical services needed for setting up a WSN.

1.1 Architecture

A wireless sensor network (WSN) is a network comprising of specially fabricated nodes that has the ability to instrument, observe and react to events and phenomena in a certain environment [4]. Each node in the network is composed of a sensing unit, a computing (processing) unit, a memory and wireless communication element. The environment of WSN operation can be the physical world, a biological system or an information technology framework, etc. WSNs in general are made up of four basic components. These basic components are [4].

1. An assembly of distributed or localized sensors
2. An interconnecting wireless protocol

3. A base station (central point) of information clustering
4. A group of computing resources at the central point or beyond to handle data collection, event trending, status querying and data mining.

Sensors are deployed in specialized domains called sensor fields. These venues of deployment can vary from large fields like battlefields or a river bed to small fields like a human organ.

1.1.1 Node Architecture

A sensor node has a processing module with embedded processing capabilities and an internal memory for temporal storage of sensed or relay data. Also embedded on a node are sensors collecting data on radio, acoustic, chemical, magnetic, optical or infrared phenomena. A node has a wireless communication unit for inter-node or node-sink communication. A node also has a power source coupled with a power processing unit to supply the appropriate power to node components. Generally, nodes might have position or location knowledge through GPS or some local positioning algorithm. A typical node architecture is as depicted in Figure 1.1.

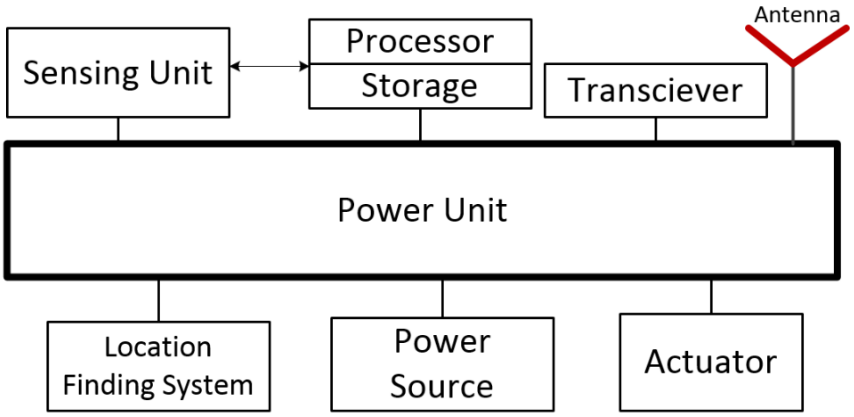


Figure 1.1: Typical Node Architecture

Each sensor node deployed in the sensor field has the capability to collect, analyze and route field data to a designated sink point. Some nodes in the field may be built to perform specialized functions in addition to the above mentioned ones. A typical WSN node arrangement is as shown in Figure 1.2.

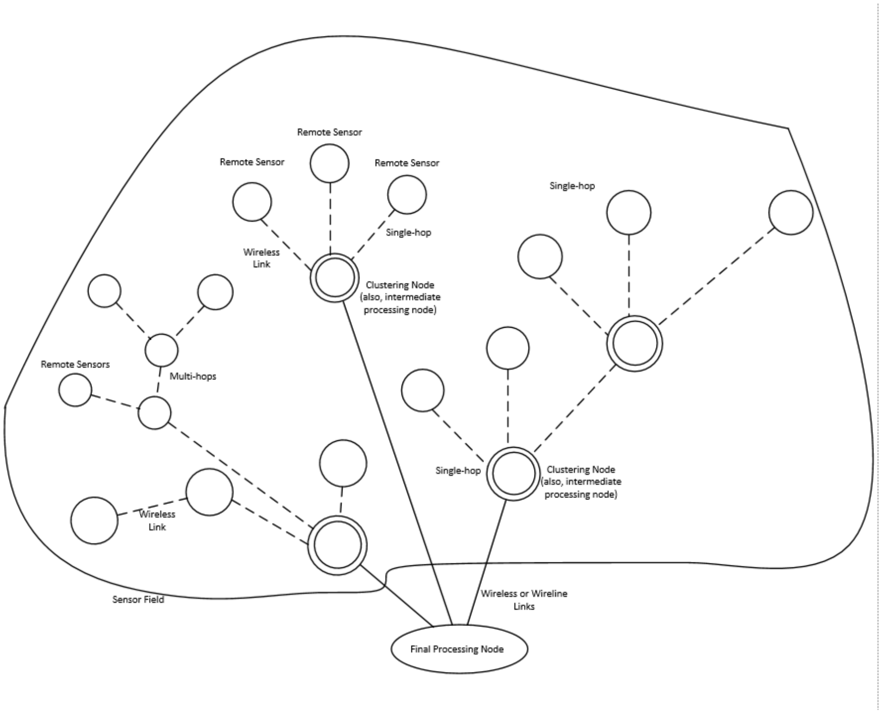


Figure 1.2: Typical Sensor Network Arrangement

The processor subsystem is the focal component in a WSN and the decision of a processor indicates the trade-off between adaptability and productivity which is identified with vitality and execution. The processors have numerous parts which include: microcontrollers, computerized signal processors, application-particular coordinated circuits, and field programmable gate arrays (FPGA). The sensing component is composed of larger than one analogue sensors. These sensing equipments are produces analogue or digital system for reading the sensor values. Most of such sensors contain their locally fashioned analog-to-digital converter (ADC) which is able to directly interface sensors with the processing unit using a standard chip-to-chip protocol. Several microcontrollers/processors are composed of one or more internal ADCs to connect to analogue devices. Recent microcontrollers integrate flash storage, RAM, ADC, and digital I/O onto a single integrated circuit. In choosing a microcontroller family, several factors has to be taken into consideration such as energy consumption, support for peripherals, voltage requirements, cost, and number of external components required.

The communication component of a node is connected to the processing unit using the serial port interface (SPI) bus. The communication subsystem is the most energy intensive component and the its operation has to managed so as to conserve power. Several of the market available transceivers give a controlling functionality to alternate the transceiver between different operation levels such as active, idle and sleep state.

The power subsystem gives a supply of direct current (DC) power to all node units and their components. This unit is composed of the energy storage component, voltage regulator, and sometimes the energy scavenging component. Node power is usually obtained from a battery-pack. Moreover, other components could be used in generating power for the sensor nodes so as to extend the lifetime of the network such as those components solar energy storage qualities.

1.2 Topologies

Due to the instrumentation and monitoring nature of WSNs, the positions of the nodes with respect to each other and a gateway in the sensor field is a crucial criteria that determines performance metrics like network power consumption which determines network coverage and lifetime [5]. For example the network topology is a key factor in time synchronization since the relative position of a node to its neighbors affect the paths chosen in synchronization algorithms for time update. Several topologies exist for WSNs. Common amongst these topologies are the star, ring, grid, random and linear topologies. Figure 1.3 presents a general picture of these topologies for a four node network.

1.3 Wireless Standards

In general, sensor nodes store and/or process the data that are collected from the sensor field. The sensed data is then transmitted to a base station or sink node in a centralized network, or can go through processing rather than transmitting

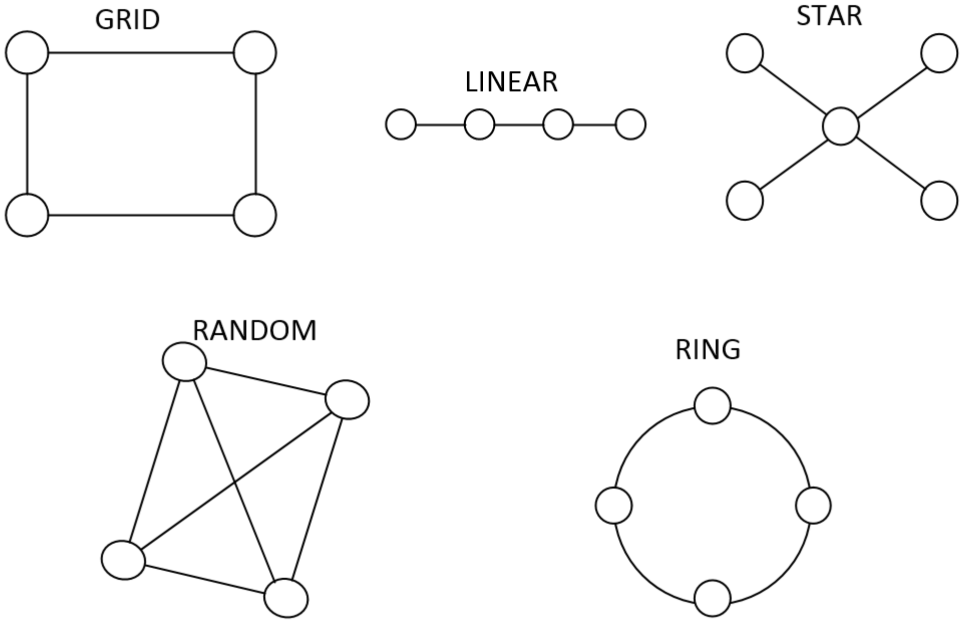


Figure 1.3: Typical WSN Topologies

directly to the base station as in distributed network. Several types of communication channels such as microwave, radio links and satellite links can be used for transmitting and extracting the information obtained from the wireless sensor networks.

There are various wireless communication standards employed in WSNs like: IEEE 802.15.4 and ZigBee. IEEE 802.15.4 standard is normally employed in low data rate networks where the sensor field is relatively small and is a power efficient protocol. ZigBee also normally communicates using low data rate and low power consumption protocols and are normally employed in a myriad WSN applications. For upper layers (application and network), ZigBee is considered as the main protocol, while for the lower layers (MAC and physical) IEEE 802.15.4 is considered as the main protocol. Some other protocols considered for wireless

Table 1.1: Comparison Between Bluetooth, Wifi and Zigbee

Category	Bluetooth	Wifi	Zigbee
Specification Authority	Bluetooth SIG	IEEE, WECA	IEEE
Year of Development	1994	1991	2003
Bandwidth	Low(800 Kbps)	High(11 Mbps)	Low(250 Kbps)
Range	10 meters	100 meters	10-100 meters
Power Consumption	Low	High	Low
Cost	Low	High	Low
Frequency of Operation	2.4 GHz	2.4 GHz	2.4 GHz

communication in wireless sensor networks are IEEE 802.11 also known as Wifi and IEEE 802.15.1 mostly referred to as Bluetooth. Table 1.1 compares between Wifi, Bluetooth and Zigbee.

1.3.1 ZigBee / ZigBee PRO / ZigBee IP

The ZigBee specification [5], was first introduced in 2004 and was further upgraded in 2006 and 2007. This wireless communication protocol has a low data rate and energy consumption and was designed by the ZigBee Alliance [6]. In the specifications of Zigbee, the application and network layers are defined on top of the physical and Mediun Access Control (MAC) layers of IEEE 802.15.4-2003, and it's main target applications are smart grid, home automation and consumer electronics applications. Since the ZigBee specification uses the physical and MAC layers of the IEEE 802.15.4, they have the same modulation techniques, bandwidth and channel configurations [7].

A ZigBee network communicates using the same, user defined channel throughout in its operation lifetime. This gives it a vulnerability in terms of interference from nearby networks communicating on a similar frequency and also susceptible to noise from different signal sources in its sensor field. Hence Zigbee is not usually employed for applications requiring robust communication protocols like in harsh industrial sensor fields [8]. To deal with this issue, the ZigBee Alliance upgraded the first variant to the ZigBee PRO specification [7] in 2007. ZigBee PRO is designed such that, it will perform effectively in industrial settings with added features like,improved security and agility towards channel noise and network frequency interference. In this protocol, the phenomena of frequency agility is added where the entire network has the capability of altering its operation channel frequency when it encounters large amounts of noise and/or interference. Despite these additions and upgrades, ZigBee has not seen a wide industrial adoption. The ZigBee Alliance pronounced in April 2009 to incorporate standards from the Internet Engineering Task Force (IETF) into future ZigBee releases, thereby opening up for IP-based communication in ZigBee networks. Of special interest for the ZigBee Alliance is the 6loWPAN working group which has created a Request for Comments (RFC4944) investigating the transmission of IPv6 packets over IEEE 802.15.4 networks. This work resulted in the ratification of the ZigBee IP specification in February 2013 [9]. Although Zigbee is not popular in industrial applications, it is widely used in small electronic equipments, in small scale commercial applications and in scientific experimental inquiry. Most wireless sensor nodes use Zigbee protocol or it's variants for communication in the sensor field.

Zigbee is also widely used in network architectures and simulators to test the performance of newly designed or proposed schemes of WSN network establishment and management services such as routing, time synchronization, localization and congestion control protocols.

1.4 Practical WSN Schemes

1.4.1 TinyOS and NesC Overview

TinyOS is an open-source working framework intended for wireless embedded sensor networks. It highlights a segment based architecture, which empowers quick development and usage while minimizing code size as required by the serious memory limitations intrinsic in sensor networks. TinyOS's segment library incorporates system protocols, disseminated services, sensor drivers, and information obtaining instruments all of which can be utilized as is or be further refined for a custom application. TinyOS's occasion driven execution model empowers fine-grained power administration yet permits the planning adaptability made important by the flighty way of wireless communication and physical world interfaces. TinyOS is not a working framework (”OS”) in the customary sense; it is a programming structure for embedded frameworks and set of parts that empower building an application particular OS into every application. The explanation behind this is to guarantee that the application code has an amazingly little memory impression. Furthermore TinyOS is intended to have no record framework, underpins just static memory designation, execute a straightforward assignment display, and give negligible gadget and systems administration abstractions [1]. TinyOS has a part based programming model arranged by the nesC programming language. Like other working frameworks, TinyOS sorts out its software segments into layers. The lower the layer the nearer it is to the equipment; the higher the segment, the nearer it is to the application. A complete TinyOS application is a diagram of segments, each of which is a independent computational entity. The nesC (network embedded systems C) is an extension to C designed to embody the structuring concepts and execution model of TinyOS.

1.4.2 Memsic Mote Overview

There are different sensor nodes platforms that are normally employed in the research fields, such as Mica, Mica2, MicaZ,TelosA, TelosB and IRIS. These sensor node platforms share the same operating system (TinyOS) and all use the nesC language for implementing WSN applications. Most experimental research works in wireless sensor networks use memsic motes. The Memsic node consists of three main components namely; MPR2400 (MicaZ mote), MIB520 gateway, and sensing boards which come in MDA's or MTS's and have varying sensing capabilities. The following sections describe the hardware and software parts of the Memsic MicaZ mote.

---

[1] http://www.tinyos.net/

1.4.2.1 Motes

Figure 1.4 shows the MicaZ mote and the block diagram MicaZ is the latest generation of motes from Memsic. The MicaZ mote is composed of different hardware components such as processor, radio transceiver, and external flash (logger). The MPR2400 (2400 MHz to 2483.5 MHz band) uses the Chipcon CC2420, IEEE 802.15.4 compliant and ZigBee ready radio frequency transceiver integrated with an Atmega128L micro-controller. It has 51 pin I/O connectors, and a serial flash memory is used. All MICA applications, softwares and sensor boards are compatible with the MPR2400.

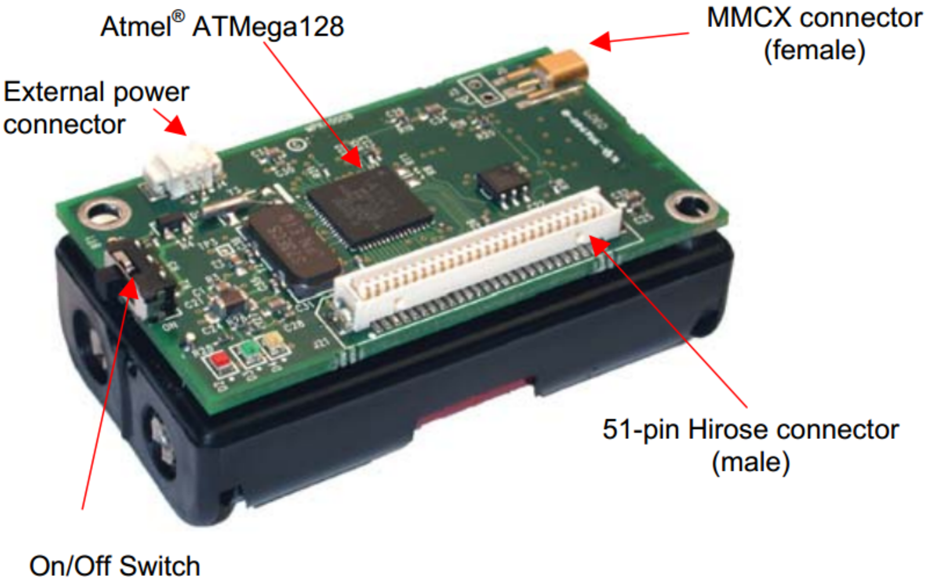


Figure 1.4: MicaZ Mote attached to MIB520 Board[2]

1.4.2.2 Sensor Board

The MDA100CB sensor and information obtaining board shown in Figure 1.5 has a precision thermistor, a light sensor/photocell and a general prototyping region. Intended for use with the IRIS, MICAz and MICA2 Motes, the prototyping territory underpins connection with every one of the 51 pins on the expansion connector, and gives an extra 42 unconnected solder points for breadboarding.

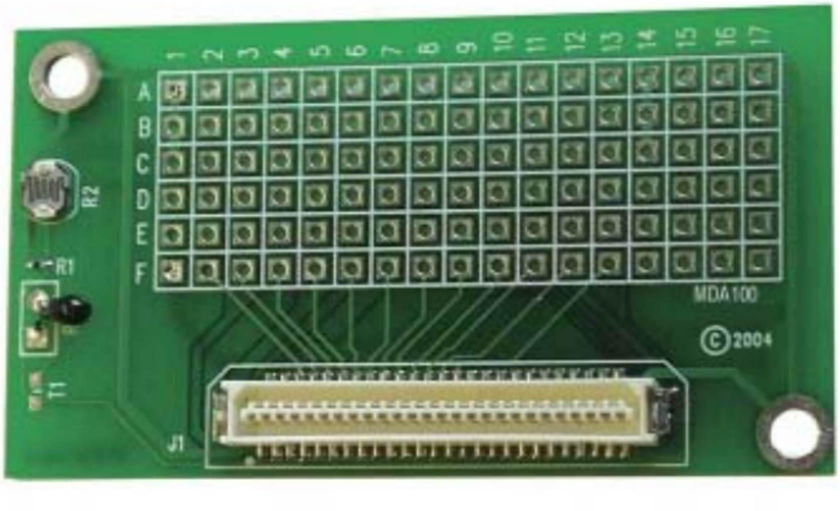


Figure 1.5: MDA100 Sensor Board[3]

1.4.2.3 MIB520 USB Interface Board

MIB520 provides USB connectivity to the Micaz motes for communication and insystem programming. It supplies power to the devices through USB bus. MIB520CB has a male connector as shown in Figure 1.6. Usually, this board connects to Micaz mote to construct the base station node that is connected to the PC for recording the received data.

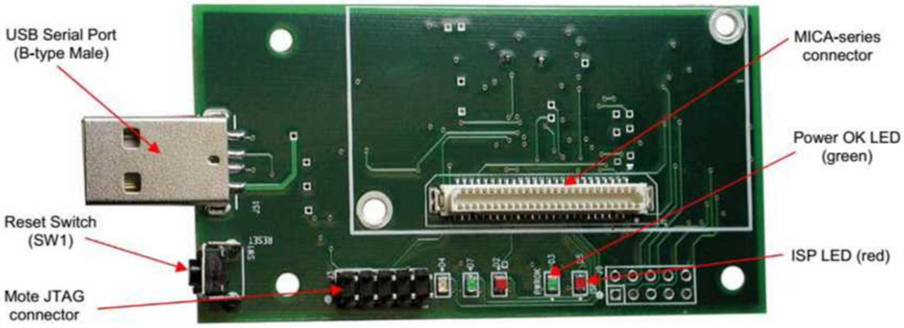


Figure 1.6: MIB520 USB Interface Board[2]

1.5 Applications of Wireless Sensor Networks

WSNs find application in several areas in the real world. These networks are employed in commercial, civil, research, industrial, military, ecological and medical applications just to mention a few, and serve a myriad of purposes like data collection, monitoring, field management and so on. Table 1.2 summarizes some applications into categories [4] and gives examples for each category.

Table 1.2: Categorical Applications of Wireless Sensor Networks

Residential	Environment	Medical	Industrial	Metropolitan	Military
Building Automation (HVAC, Security, etc.)	Weather Sensing and monitoring	Heartbeat sensors	Assembly line and workflow	Highway and Bridge Monitoring	Battle field management, surveillance and Reconnaissance
Hotel energy management	Tracking soil contamination	Medical disaster response	Inventory control and management	City Lighting control	Radiation and nuclear threat detection
Home control	Earthquake detection	Telemedicine	Distributed robotics	Public assembly locations monitoring	Missile detection systems
Residential access control	Wildfire instrumentation	Patient rehabilitation	Industrial Instrumentation	Highway monitoring	Intrusion detection
Appliance control (lighting and HVAC)	Precision agriculture	Body-worn medical sensors	Mine air-conditioning	Mass-casualties management	Monitoring on-truck and on-ship tamper of asset
Wi-Fi tags to track children	Farm sensor and actuator networks	Collection of long-term databases of clinical data	Industrial automation	Highway monitoring	Military tactical surveillance
Gas, water, and electric meters	Monitoring animal populations	Heartbeat sensors	Building automation	Bridge and highway monitoring	Monitoring for explosives

1.6 Network Establishment and Management Services for WSNs

It is clear from these applications that, several aspects of industrial, military and commercial works require WSNs and the need for such networks is on an exponential increase in all aspects of distributed systems [10]. Information generation, transmissions and processing in WSNs require several features. Key amongst them are;

1. Wireless transmission protocols like Zigbee, ISA100.11a, IEEE 802.11 and WirelessHart
2. Infrastructure establishment services like time synchronization, topology control, clustering and localization
3. Network management services such as naming and operation mappings
4. Effective operating systems like TinyOS, Mate, MagnetOS, MANTIS, PicOS, and SenOS.
5. Sensor network databases
6. Traffic management services like MAC Protocols, transport protocols and routing protocols

Upon initial activation, WSNs need to perform a myriad of tasks to establish the necessary infrastructure to ensure effective collaborative work among sensor nodes. This makes infrastructure establishment very crucial for network operation.

Infrastructure establishment services like time synchronization, topology control, clustering and localization allow for nodes to organize themselves for effective clustering and mapping, establish communication for efficient resource management, and establish themselves for higher level collaborative network functions like routing. Among these services, time synchronization is the most important. This is because, synchronizing network nodes in time is mostly needed before almost all other tasks are carried out for establishing network infrastructure.

Also, time synchronization is needed for assigning global time stamps for sensed data and events, for coordinated network tasks such as duty cycling of nodes for energy efficient techniques and low power Time Division Multiple Access (TDMA) based Medium Access Control (MAC) layer and so on. Further, in some of applications such as data fusion, human and animal tracking, speed estimation, the network needs to know the time of all nodes in order to determine the time occurrence of the events. Synchronizing nodes in time can help in saving the energy by reducing the guard times that are attached to the transmitted packets among nodes. This is mainly true for the networks that use duty-cycling techniques and switch off the radio to reduce the energy consumption.

1.7 Summary

In this chapter, we presented a general overview of wireless sensor networks. We reviewed the general architectures of wireless sensor networks and sensor nodes. We further described the common topologies and wireless protocols used for setting up wireless sensor networks. We presented an overview of the TinyOS platform and the NesC programming language employed for experimentation in most wireless sensor networks. We then gave a summary about some network establishment and managements services that are crucial for the operation of wireless sensor networks. Key services among them are routing and time synchronization. We briefly gave an exposition on the significance of time synchronization in wireless sensor networks and the problem of synchronization.

To address the issue of time synchronization in WSNs, lots of research has gone into developing the most efficient protocols to address this issue. The next chapter delves into time synchronization in WSNs and addresses the problems and features of synchronization, and the required tasks needed to carry it out. The chapter ends with a review of the more relevant recent reported protocols and algorithms for time synchronization in WSNs.

CHAPTER 2

TIME SYNCHRONIZATION IN WIRELESS SENSOR NETWORKS

Wireless Sensor Networks (WSNs) as distributed systems used for several precision and sensitive sensing and instrumentation applications require network nodes to be as closely synchronized in time as possible. Several factors [11], which include but not limited to; the tight link between sensors and the physical world, the scarcity of energy for deployed nodes, the need for large scale deployment, decentralized topologies and unpredictable and intermittent connectivity between network nodes necessitate an accurate, flexible and robust time synchronization for wireless sensor networks. Although most traditional networks depend on physical time for time synchronization, WSN applications such as object tracking, consistent state updates, and distributed beamforming make it impractical to depend on logical physical time for WSN time synchronization.

In this chapter, we present a general overview on the topic of time synchronization in wireless sensor networks. The chapter gives an exposition on the problem of time synchronization, the need for effective and robust time synchronization algorithms in wireless sensor networks and the models of hardware and logical clocks of wireless sensor network nodes. An extensive literature survey is carried out for the recently reported time synchronization protocols presented in the literature where we classify protocols into the centralized and distributed types. The main merits and demerits of these classes are then discussed.

2.1 Background

2.1.1 The Time Synchronization Problem

Generally, the concept of logical time eliminates the need for physical time synchronization in applications where only the causal relationships between different events are of interest. Despite this advantage, logical time captures the relationships between in-system events which is determined by communication among application processes generating those events. This does not allow for the interpretation of system events in relation to the physical world and therefore makes physical time insufficient for the operations of wireless sensor networks. Consider the problem [12] in Figure 2.1, where an acoustic wavefront generated by a distance sound source impinges onto an array of acoustic sensor nodes. It is required in this problem to estimate the angle at which the wavefront from the sound source impinges the array denoted as β. Each node in the array records a time,t at which it is impinged by a sound event.

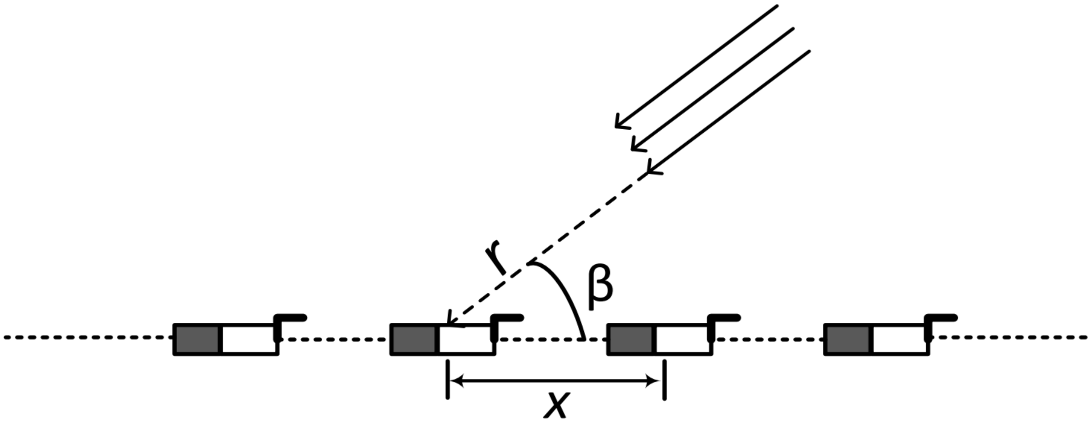


Figure 2.1: Estimating Angle of Sound Impingement on an Array

From Figure 2.1, $x = rsin\beta$ and $\beta = arcsin\frac{x}{r}$. Since sensors have knowledge of their respective positions, x is known. Using knowledge of time difference between node sensors, Δt and the speed of sound, ν , the value of r can be calculated as $r = \nu\Delta t$. Now, assuming $x = 1m$ and $\Delta t = 0.001s$ then the calculated angle, $\beta = 0.336rad$. If the clocks of the node sensors are only within $500\mu s$ accurate, the true difference in time can occupy a range of $[500\mu s, 1500\mu s]$ hence the estimated angle, β can be in a range of $[0.166rad,\ 0.518rad]$. This presents a big problem, since a very small error in time synchronization of sensor nodes can lead to a significant biased estimates obtained from sensed quantities.

2.1.2 The Need for WSN Time Synchronization

There are at least two ways [12] to obtain a more accurate estimate. One method is to obtain the combined measurements of several sensors and average them to find the estimate. This method however is guaranteed accuracy if the variance in readings is very small. If the variance is very large, the mean value of the combined reading might not be a true representation of the event(s). The second method which has received more research attention is to employ rigorous time synchronization algorithms to keep the inter-sensor time difference as small as possible by synchronizing all array sensors and hence all WSN nodes in time. These amongst several applications require for time synchronization of all nodes clocks with respect to physical time. Some of these applications are [11]:

1. Distributed Beamforming: To perform spatial filtering using beamforming arrays, i.e. receiving signals from a specified direction, depends on the relative offsets of the array sensors.

2. Duplicate Detection: The time of reading an event can help nodes or administrator application determine whether it is a single event detected from two vantage points or two distinct real time events.

3. Object Tracking: Combining proximity detection from several sensors at different locations in a sensor field can be used to determine the size, shape, velocity, location, and /or the direction of an object.

4. Consistent State Update: To accurately estimate the current state of an

event, it might require knowing the most recent measurements of the event.

5. Temporal Order Delivery: Several data fusion algorithms like Kalman filters require events in their order of occurrence in order to process them.

2.2 Node Clock Modeling and Synchronization

2.2.1 Hardware Clock Models

As per the definition in [13], a hardware clock is an electronic circuit that counts oscillations at a certain operating frequency. Henceforth, a hardware clock for the most part comprises of an oscillator and a counter. The oscillator is utilized to produce periodic events whereas the counter aggregates these events so as to obtain the measured time. For example, the oscillator output can be expressed by a sinusoidal waveform,

$$S(t) = A(t) sin \Phi(t) \tag{2.1}$$

where $\Phi(t)$ is the phase and;

$$A(t) = A + \Delta A(t) \tag{2.2}$$

is the amplitude, $\Delta A(t)$ defines the variations in amplitude, and t denotes the reference or global time. Note that the specific amplitude values of the oscillator are insignificant for this model. The instantaneous radian frequency function $\hat{\Phi}$

can be modeled in the form [14]:

$$\hat{\Phi}(t) = \omega_o + \sum_{k=0}^{M-1} \frac{L(k)}{k!} t^k + \hat{\xi}(t) \tag{2.3}$$

where ω_o is a constant representing the nominal value of the free running radian frequency of the oscillator. $L(0)$ represents the initial radian frequency error (departure). This error arising from the uncertainties in the initial oscillator settings. The $L(k)$s $k = 1, \cdots, M$ specify a set of time-independent values which characterizes the kth order radian frequency drifts, and $\hat{\xi}(t)$ is a stationary zero-mean random process characterizing the short-term oscillator instabilities. The oscillator phase process can be obtained by integrating (2.3) from 0 to t resulting into:

$$\Phi(t) = \Phi(0) + \omega_o t + \sum_{k=1}^{M} \frac{L(k-1)}{k!} t^k + [\xi(t) - \xi(0)] \tag{2.4}$$

for $M \geqslant 1$. The value of the hardware clock is obtained by dividing the oscillator phase by the nominal free-running radian frequency of the oscillator ω_0. Correspondingly, the hardware clock value $T(t)$ can be defined as

$$T(t) \triangleq \frac{\Phi(t)}{\omega_o} = T(0) + t + q(1)t + \frac{q(2)}{2} t^2 + \sum_{k=3}^{M} \frac{q(k)}{k!} t^k + \Upsilon(t) \tag{2.5}$$

where $T(0) = \frac{\Phi(0)}{\omega_o}$, $q(k) = \frac{L(k-1)}{\omega_o}$, $k = 1, \cdots, M$ are a set of values modeling the $(k-1)$ th order time drifts and $\Upsilon(t) = \frac{\xi(t)+\xi(0)}{\omega_o}$ is, in general, a non-stationary stochastic process characterizing the short-term clock instabilities. Even though

(2.5) is a complete hardware clock model in the sense that it will be accurate for smooth phase functions as M is large and t is small, it is not necessary in practice. Firstly, the values of $q(k)$, $k = 3, \cdots, M$ are so small (compared to the clock resolution) [15] that it can be totally ignored. Moreover, in this work it is assumed that $\Upsilon(t) = 0$. This is a reasonable assumption since we are not dealing with the precise measurement of time, but rather the relative time synchronization of clocks in the network [15]. In this way, the important terms that characterize the performance of a hardware clock can be defined based on (2.5). These terms can also be called hardware clock parameters and they include:

1. Offset: $\theta \triangleq T(0)$
2. Frequency: $f \triangleq 1 + q(1)$
3. Skew: $\rho \triangleq q(1)$
4. Drift: $\frac{q(2)}{2}$

Note that offset, frequency, and drift are all determined by the hardware clock and cannot be measured or adjusted. For a crystal oscillator commonly used in telecommunication radios, the reasonable values for the skew are [1, 100] part per million (ppm) relative to f[10]. Additionally, the drift is mainly caused by a gradual change in frequency over days or months, which is known as aging. An illustration of the aging effect is shown in Figure 2.2, which is inspired by [15]. Since the values of skew and drift usually have different scales, there should be different adopted hardware clock models in different applications.

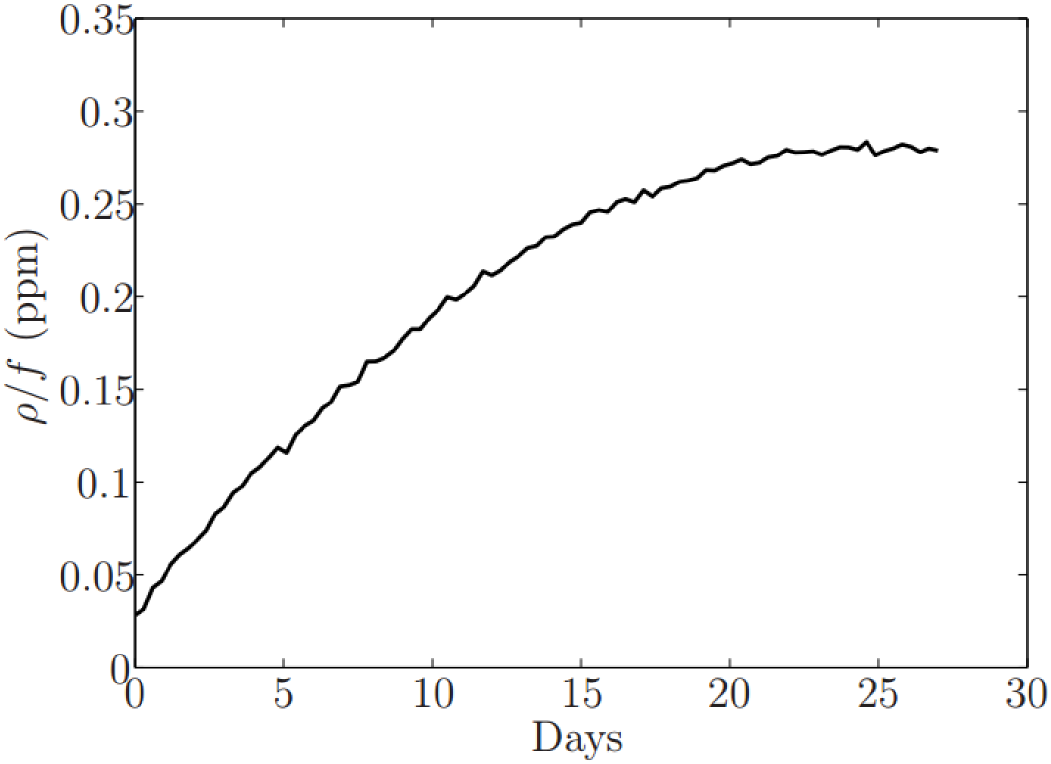


Figure 2.2: An illustration of Aging Effects of Node Clocks over Time (days)[20]

For long-term applications, i.e., the synchronized clocks need to be used for more than several hours, the effects of drift should be considered [16], [17], [18], [19]. In this case, clock **model I** is presented as follows.

$$T(t) = Dt^2 + ft + \theta = Dt^2 + (1 + \rho)t + \theta \tag{2.6}$$

For short-term applications, the effects of drift can be neglected. Then, there are two possible hardware clock models being utilized in the existing literature. Some works take both offset and skew into account [18], which gives clock **model II** as

$$T(t) = ft + \theta = (1 + \rho)t + \theta \tag{2.7}$$

Other works only consider offset [16], [19], , which introduces clock **model III**

$$T(t) = t + \theta \tag{2.8}$$

In [20], the authors presented the ticks of node hardware clocks of sensor nodes as a cumulative function that gives a description of the time evolution of the tick counter that can be used for the implementation of the clock. This is given as

$$s(t) = \sum_{k=0}^{\infty} \mathbf{1}(t - t(k)) \tag{2.9}$$

where $\mathbf{1} = \begin{cases} 1, & t > 1 \\ 0, & \text{otherwise} \end{cases}$

$t(k), k \in \mathbf{N}$ is the time at which the hardware oscillator produces an event. $s(t)$ could be further approximated by [20] as,

$$s(t) = \int_{-\infty}^{t} f(\zeta) d\zeta \tag{2.10}$$

and the frequency function is interpreted as $f(t) \triangleq \frac{1}{t(k+1)-t(k)}, \forall t \in [t(k), t(k+1)]$

Typical nominal values of $f(t)$, $\hat{f}$ are specified with upper and lower bounds given respectively as $fmax$ and f_{min} such that $f(t) \in [fmax, f_{min}] = [\hat{f} - \Delta f_{max}, \hat{f} + \Delta f_{max}]$

2.2.2 Logical Clock Models

Since the clock parameters of a hardware clock cannot be measured or adjusted manually, each node also maintains a logical clock whose value which is a function of the current hardware clock value. In this work, the focus is on the affine function, and calculates the logical clock value $C(t)$ as

$$C(t) = \alpha T(t) + \beta \tag{2.11}$$

where $\alpha(\alpha > 0)$ and β are control parameters updated by the synchronization algorithm. In this way, the logical clock value $C(t)$ represents the synchronized time for each node.

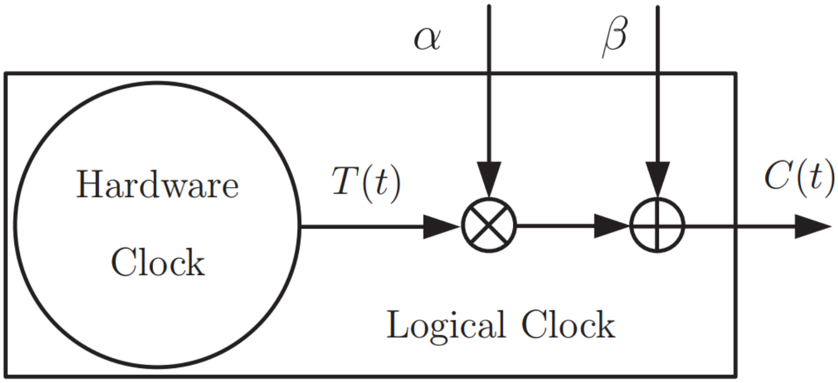


Figure 2.3: The Relationship between Hardware Clock and Logical Clock

The logical clock $C(t)$ is used in the update because resetting the hardware clock for an update during node operation is inefficient for node operation and may interfere in the operation of node application. This relationship is also shown in 2.3. Where, α is the local clock skew rate which determines the clock speed and β is the local clock offset. As shown in Figure 2.4, the clock skew of a clock determines how fast the clock is and different sensor nodes equipped with clocks with different oscillators will tick at relatively different speeds.

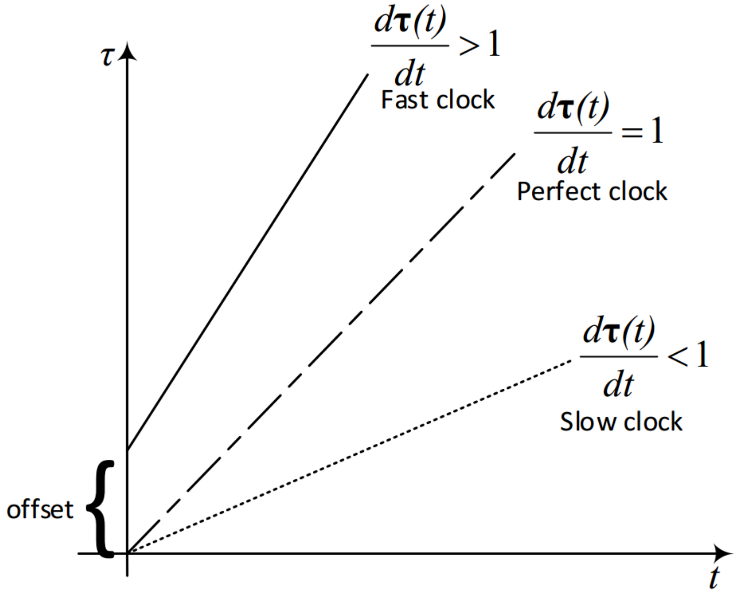

$\frac{d\tau(t)}{dt} > 1$
Fast clock
$\frac{d\tau(t)}{dt} = 1$
Perfect clock
$\frac{d\tau(t)}{dt} < 1$
Slow clock
τ
offset
t

Figure 2.4: Nodes Clock of Varying Drift Rates

2.3 Performance Metrics for Time Synchronization Protocols

Even though a wide range of applications require time synchronization, each application places different performance demands. For instance, some applications need high accuracy, e.g., object tracking, while some applications have less requirement on the accuracy but more on the energy efficiency, e.g., instrumentation or power management applications. To the best of our knowledge, there is no unique criterion to evaluate various time synchronization schemes. Some of the criteria might even be unique to the nature of the synchronization algorithm. In this section, we provide some possible metrics for the performance evaluation of time synchronization protocols.

2.3.1 Accuracy

In pairwise clock synchronization, the objective is usually to synchronize one node to the other one by adjusting local clock parameters. In this way, the clock synchronization problem is inherently a parameter estimation problem. Even in network-wide synchronization, each node needs to update its clock parameters based on the timing messages received from other nodes. This is also related to parameter estimation if we consider that the received messages are inaccurate due to the delays and distortions. In parameter estimation, a fairly common performance metric is the root mean square error (RMSE). The RMSE of an estimator $\hat{T}$ is defined as

$$RMSE \triangleq \sqrt{(\hat{T} - T)^2} \tag{2.12}$$

where T is the true value, and the expectation is taken over different realizations.

2.3.1.1 Problem of Accuracy

Each node in a sensor network is composed of an oscillator operating at a specific frequency and a counter register, which is incremented in the node hardware after a certain number of oscillator pulses. The nodes software (firmware) can only access the register values and the time resolution is determined by the time between two increments. Where events occurring between two increments are not distinguishable from their timestamps.

The value of the hardware clock of a node in real time t denoted as $T(t)$ which can be seen as an abstraction of the counter register with a non-decreasing time value. With this representation, the software equivalent of the clock value, $C(t)$ given by Equation 2.11, i.e.,

$$C(t) = \alpha T(t) + \beta \tag{2.13}$$

With the phase shift (offset) and the skew rate of a node represented as α and β respectively. Since it is neither possible nor desirable to alter the oscillator or the shift register, the coefficients and could be manipulated to obtain a desired value. Based on this definition, two cases of the precision sensor node clock values are defined and hence synchronization classified into two folds. The external and internal synchronization precision.

2.3.1.2 External Synchronization

Given a network of n nodes, all nodes are said to be accurate at time, t within a bound δ if

$$|C(t) - t| < \delta, \forall i, j \in \{1, 2, \cdots, n\} \tag{2.14}$$

If the above condition is satisfied in a wireless sensor network then all the nodes are said to be externally synchronized [21].

2.3.1.3 Internal Synchronization

Similarly, nodes 1,2, $\cdots$,n are said to agree on the time within a bound of δ or are said to be synchronized internally if for two nodes i and j

$$|C_i(t) - C_j(t)| < \delta, \forall i, j \in \{1, 2, \cdots, n\} \tag{2.15}$$

A reliable source of real time like UTC must be available so as to achieve external synchronization. It is obvious from (2.14) and (2.15) that, if all nodes are externally synchronized with a bound δ, then the nodes are also internally synchronized within a bound 2δ [21].

From these definitions, three problems can be discussed.

1. Nodes are switched on at different and essentially random times, and hence, without correction, their initial phases are also stochastic in nature.

2. Oscillators often have a priori a slight random deviation from their nominal frequency, called drift or clock skew due to impure crystals but oscillators also depend on several environmental conditions like pressure, temperature, and so on, which in a deployed sensor network might well differ from laboratory specifications. The clock skew is normally expressed in parts per million (ppm) and gives the number of additional or missing oscillations a clock makes in the amount of time needed for one million oscillations at the nominal rate. Normally, cheaper sensor nodes having cheaper oscillators have a relatively larger clock skew as compared to expensive nodes with well

fabricated oscillator.

3. The oscillator frequency is time dependent. The frequency variations could be within a short span which is normally caused by changes in temperature, electrical supply variations, and variations in air pressure. Frequency variations could be long term which is caused by aging. This factor makes the oscillator frequency stable within a certain time value and hence time resynchronization must be done within this time interval to keep track of changing oscillator frequencies.

From the above problems, it could be inferred that even if two nodes have the same type of oscillator and are started at the same time with identical logical clocks, the difference of $|C_i(t) - C_j(t)|$ can be arbitrary large as t increases. Hence a time synchronization protocol is needed to keep the difference within the bounds of (3.12). Also, since the skew rate β is time varying, one time synchronization of the nodes within a wireless sensor network is not useful. It is, however, often possible to bound the maximum drift rate, that is there is a $\rho > 0$ such that[3],

$$\frac{1}{\rho_i + 1} \leq \frac{dT_i(t)}{dt} \leq \rho_i + 1 \tag{2.16}$$

This bound can be used to find the conservative resynchronization frequency.

2.3.2 Precision

For deterministic algorithms, precision is defined as the maximum synchronization error between a node and real time or between two nodes. For stochastic algorithms, the mean error, the error variance, and certain quantiles are of interest.

2.3.3 Cost

Another evaluation criterion, which is mainly a practical issue, is the cost of implementing an algorithm. In general, cost is commonly studied using the following metrics.

1. Convergence time, which is the time needed by the algorithm to converge to a desired synchronization accuracy.
2. Power consumption, which is a combination of the power required to perform the local implementation of the synchronization algorithm, and the power consumed to send and receive timing messages.
3. Communication overhead, which is the amount of transmitted information needed for the synchronization algorithm.

2.3.4 Fault Tolerance (Robustness)

This metric determines how well an algorithm can cope with unreliable and time variant communication links and channels, network partitions, random topologies, node failures, node mobility, and so on. There are different sources for the randomness in WSNs therefore, robustness is another important issue for clock synchronization algorithms. The considerations of robustness include

1. Robustness against random transmission delays with different levels and distributions;

2. Robustness against dynamic topologies with node failures and changing connections, which is considered for network-wide synchronization algorithms;
3. Robustness against packet losses regarding the timing messages.

2.3.5 Scalability

The network may consist of a few or many nodes. Hence, one way to assess the network-wide synchronization algorithms is to evaluate if they can perform well in both small networks and large networks, which is often referred as the scalability.

2.4 Review of Time Synchronization Protocols

Due to the several discussed difficulties in the problem of WSN time synchronization and the importance of keeping sensor nodes in a WSN synchronized as closed as possible, an extensive study has been carried out in this discipline. The development of novel time synchronization algorithms began for the most part, with the development of centralized protocols [22], [23] and [24]. These synchronization schemes normally achieve global network time-synchronization by synchronizing all network nodes to a root/leader/reference node in a hierarchical tree fashion or using network clusters. Popular among these protocols include the Reference Broadcast Synchronization (RBS) [25], Timing-synch Protocol for Sensor Networks (TPSN) [26], Lightweight Time Synchronization (LTS) [27], Hierarchical Time Synchronization Time Synchronization (HRTS) [28]. Realizing the problems of node failure, sensitivity to topology dynamics and scalability in the usage of centralized protocols, research focus switched towards the development of distributed schemes where the issue of network robustness, node failure and scalability were of prime focus. Most distributed algorithms however are based on consensus, where an agreement is sort between sensor nodes so as to attain global time synchronization in the entire network. With several of the consensus protocols like the Gradient Time Synchronization Protocol (GTSP) [29] and the Average Time Synch (ATS) protocol [22]. The adoption of a consensus approach to the problem of time synchronization employed by most researchers find motivation in three main advantages. First, consensus based algorithms are distributed and hence do not require a root(leader) node or a spanning tree for time synchronization. Secondly, more accurate synchronized clocks may be obtained for the entire network especially for neighbors based on a consensus synchronization algorithm. Finally, consensus algorithms compensate for the skew and offset differences among network nodes. This change in the research direction has led to the development of several distributed protocols. Examples of these protocols being, the Time Synchronization Protocol using the Maximum And Average Values (TSMA) [30], Time Diffusion Protocol (TDP) [31], External Gradient Time Synchronization Protocol (EGSync) [32], Reach-Back Firefly Algorithm (RFA) [33], Gradient Time Synchronization Protocol (GTSP) [29], Average Time-Sync Protocol(ATS) [22], Weighted Maximum Time Synchronization Protocol(WMTS) [34], and Maximum Time Synchronization Protocol [34].

2.4.1 Review of Centralized Protocols

TPSN [26] was designed by adapting the design of NTP to give some flexibility which would allow for time synchronization in wireless sensor networks. The algorithm is made up of the level discovery and the synchronization stages. In the level discovery part of the algorithm, the entire network is organized into a tree topology with a leader node acting as the root node. A pair-wise synchronization is then carried across the whole tree between nodes sharing a common branch using the conventional sender receiver synchronization handshake exchange approach. TPSN is a centralized scheme and hence uses medium access layer time-stamping to minimize message delivery non-determinism and to enhance the efficacy of synchronization. The convergence speed of TPSN is reported to increase linearly with the maximum number of network hops. TPSN has two main disadvantages. Firstly, suppose the root node goes off, then a root discovery process has to commence for the election of a new leader. This adds a significant overhead on the protocol source-code and results in prolonged long periods of network desynchronization. Furthermore, geographically close nodes might not be close in terms of the tree distance, which leads to an increase local synchronization errors.

The Lightweight tree-based synchronization (LTS) protocol presented in [27]; achieves a reasonable level of accuracy while using reasonable amount of computational resources like memory space and CPU time. LTS is classed into two categories; centralized and decentralized. In the centralized approach, each round starts by only one node with a certain frequency whereas in the decentralized, each node can start the synchronization. The LTS algorithm uses a searching process to construct the tree-based structure for the whole network. Tree nodes then share the synchronization data with each other. The drawback of this algorithm is that the accuracy of the synchronization decreases with increasing depth of tree and this in turn increases the error value for each node.

Jiming in [35] proposed a time synchronization algorithm called feedback-based synchronization that considers the synchronization problem is a closed-loop control problem and using proportional-integral (PI) controller to compensate the drift of clock that results from the internal and external factors. The accuracy of this algorithm depends on the response and overshoot time. This algorithm needs a reference node and it is a tree- based synchronization and hence suffers from link and node failures.

Tiny-sync and mini-sync presented in [36]; it depends on a set of data points, where each point is collected by a two-way message exchange and consists of two constraints which are bounded by the offset and the skew parameters. Increasing the number of the data points increases the precision of the estimation bounds of the two parameters. The computational complexity of the tiny-sync algorithm is low because it is dependent on the specification of only four points with few operations. The mini-sync algorithm has improved accuracy than the tiny-sync, which is achieved at a small computational cost. This algorithm has an accurate offset and drift information together with tight, deterministic bound, accuracy, low computation and storage complexity, insensitivity to communication errors and each clock can be approximated by an oscillator with fixed frequency.

RBS [25] is designed to ensure the clock synchronization between a class of network nodes located within a single-hop broadcast range of a beacon node. In comparison to conventional protocols operating on LANs, this algorithm contributes to synchronization by directly removing two of the main sources of nondeterminism found in packet transmissions, period of message exchange and access time. This was done by making use of the property of time critical path, defined as the path of a message that adds to stochastic errors in clock synchronization. The algorithm makes use of least-square linear regression to compensate for the clock skew, its convergence speed is reported to increase linearly with the maximum number of network hops. The use of the leader(beacon) node within a cluster makes RBS vulnerable to node and link failures and changing topologies.

FTSP [37] is an ad-hoc, multi-hop time synchronization algorithm for WSNs. In this protocol, the node with a low value of ID is selected as a root node whose clock value is used as the reference time for other nodes. The root node periodically floods a synchronization packet with its local time to all network nodes. In this protocol, clock synchronization between sensor nodes is achieved by employing a single message that is transmitted by a sender node to multiple receiver nodes. This message is time-stamped by both the transmitting node and receiving ones at their MAC layers. This time stamping done at the MAC layer helps improve the synchronization accuracy of the protocol. To compensate for drifts between nodes and the reference nodes, linear regression is employed in FTSP.

Typical WSN operate in areas larger than the broadcast range of a single node. Hence sensor networks employed for running this protocol tend to be multi hop networks. The leader or reference node of the network is a single node which is dynamically selected using a specially designed algorithm through election. This reference node maintains the global clock values and synchronizes the slave nodes to its clock. In FTSP the ad-hoc type network structure is employed as compared to the spanning tree structure employed in other centralized protocols like the TPSN. The method employed in FTSP saves the initial phase of tree maintenance and hence improving convergence time of the protocol. The FTSP methodology proved more robust under different failures scenarios in the network due to the flooding technique used. In RBS time-stamps are not included in the message which is transmitted whereas in FTSP time-stamp of the transmitting node is incorporated in the currently sent message. Hence, the time-stamping on the sender side must be performed before the message is transmitted to the receiver. In a real practical case if clocks that operates on similar frequencies are employed, setting the clock offset once suffices for synchronization. But this is not the case and we need to send synchronization message again and again. Since this doesnt occur practically, in FTSP, network resynchronization is carried out with a period that is less than one second. If resynchronization period is chosen to be less than a second, then a microseconds synchronization accuracy is achieved if not node clocks tend to loss accuracy due to clock skews. The main setback of this protocol is that resynchronization period is very shortly and hence a large overhead along with a substantial bandwidth is used in every resynchronization period, hence the significant expense of energy.

Lenzen *et al* [38] studied the effects of clock drifts and communication delays when scaling the diameter of a network. The authors then do a rigorous analysis to prove the bound of synchronization accuracy which they report as approximately the square root of the network diameter and design a novel synchronization technique named PulseSync to achieve this bound of accuracy. In this technique, flooding is done in a rapid manner as opposed to the slow flooding adopted in FTSP.

Yildirim and Kantarci [39] show that the slow propagation speed of flooding adopted in FTSP significantly reduces the synchronization accuracy and scalability of wireless sensor networks. They also show that, the rapid flooding adopted in PulseSync [38], has several drawbacks. The authors design a protocol which removes the undesirable drawbacks of the slow flooding in FTSP without altering the propagation speed of the flood. This method is dubbed Flooding with Clock Speed Agreement (FCSA) protocol. The show with experiments, that the synchronization accuracy and scalability of slow-flooding can drastically be improved by employing a clock speed agreement algorithm among the sensor nodes.

DMTS [40] collects different concepts at the same time such as master-slave synchronization, sender-receiver synchronization and clock correction approach. This protocol was created to avoid the round trip time estimation in the previous protocols. DMTS synchronizes the transmitter with multiple receivers at the same time with less number of packets when compared to RBS. In this protocol, the leader node is selected as time master and broadcasts its time. All receivers estimate the delay value and set their time the same as the master time. All nodes that receive this packet can synchronize with this leader. DMTS has some advantages like; computational complexity is low and energy efficiency is high. On the other hand, one of the drawbacks of DMTS protocol is that it uses only low frequency external clocks.

The authors in [41] presented a novel technique for time synchronization that uses a dual-clock delayed-message method, for wireless sensor networks (WSNs). In order to ensure energy efficiency for sensor nodes, the time synchronization scheme adopts the flooding time synchronization method using one-way timing messages. The clock variables like the clock drift and clock offset are then estimated using the maximum-likelihood (ML) estimation of time parameters, so as to ensure an efficient time synchronization. Further, this synchronization method transforms the clock drift and offset estimation problem into a model that has no dependency of random delay and propagation delay. This synchronization scheme has the merits of minimal energy consumption, low complexity and effective for clock synchronization of sensor networks prone to jitter and delays. The method is however not so efficient for very large networks since the flooding method would require the transmission of several packets in other to attain synchronization.

In [20], a time synchronization algorithm named FloodPISync and PulsePISync are presented. In these algorithms, the clock of nodes are modeled and adjusted as a proportional integral control system and show with computations, simulation and experimental work that this model effectively synchronizes Wireless sensor nodes. The FloodPISync is done by synchronizing all nodes to a reference node by adjusting the node drifts and offsets using the PI analogy. A similar strategy is adopted in the PulsePISync but here, flooding is done in a rapid manner similar to that of PulseSync.

The authors in [42] presented a novel framework for addressing the problem of wireless sensor node clock synchronization. This framework was analyzed and found to conserve some node resources and hence presented a more energy efficient method for achieving synchronization in time as compared to traditional synchronization techniques like the Reference Broadcast Time Synchronization (RBS) and Time-Sync Protocol for Sensor Network (TPSN). The synchronization was designed to be not require a Global Positioning System or some other form of external time like UTC in order to effectively achieve synchronization coordinate with time, like conventional time synchronization protocols used in wireless sensor networks. This method was classified as a peer to peer, clock-correcting sender- receiver network-wide synchronization protocol which depends on the features of the sensor network. The analysis of the clock offset was done using the maximum probability theory. The synchronization accuracy of this scheme was reported to depend on the resynchronization interval. The protocol performance was then evaluated by conducting a simulation on NS2. The protocol was found to demonstrate energy efficiency in the synchronization process.

2.4.2 Review of Distributed Protocols

Weilian et al. [31] proposed the Time Diffusion Protocol (TDP) that pushes all nodes to have time slot with a small difference. This algorithm is applied periodically to compensate for clock drifts. It is divided into two parts; active and inactive parts. In the active part, there are multiple of cycles with T for each cycle. During each cycle, a set of nodes are selected as master nodes by election. Each master node starts the diffusion of timing messages; it builds tree-based scenario in the network. Additionally, the network has non-leaf nodes which are considered as diffused leaders and elected by the election procedure. This will make some propagation on the timing messages. This algorithm tolerates packet losses, can achieve equilibrium for all nodes during all synchronization times. Since it is independent on the static structure, it provides the network with flexibility, mobility, and many of the master nodes are distributed in the network with the hierarchal structure. The last advantage is that the synchronization can be done without using an external time. However, there are several drawbacks for this algorithm including; high complexity, convergence time is high when there is no external time, and clocks can run backward. This can occur when the value of clock is changed to a lower value.

In GCS [43], nodes take turns to broadcast a synchronization request to their neighbors. Each neighbor then responds with a packet having their local time. The receive node(s) centered in this exchange averages the received timestamps and broadcasts this value back to its neighbors. The neighbors use this value to update their clocks. This process repeats until a global network synchronization is attained at a certain level of accuracy. The protocol is fully distributed, but has no clock skew compensation.

Yi *et al*, in [10] proposed a clustering firefly synchronization algorithm called reach-back firefly algorithm (RFA) that depends on the initial phases of all nodes. Due to the difference between the initial phases, the number of clusters will be evaluated. Each cluster starts the synchronization process independently and each node receives firing packets from its cluster, until all clusters reach the synchronous state. These synchronous clusters are considered as new integrated nodes when the clusters enter the synchronization phase. This technique deals with nodes that are randomly distributed, all- to-all communication, has homogeneous oscillators and bi-directional links. The simple RFA technique mainly suffers from a worse precision in averaging the packet delays and is not robust. Leidenfrost *et al* in [44] proposed another technique that overcomes this drawback by using the two techniques together which called Fault-Tolerant Averaging (FTA) and robust RFA. This technique is suitable for network that suffers from delays and provides a high level of synchrony in multi-hop networks.

Sommer in [29] presented the gradient time synchronization protocol (GTSP).This protocol is a fully distributed time synchronization scheme. In its operations every node periodically sends a broadcast packet with the time information. This packet will be received by all neighbors and used to estimate their clocks. In this network neither tree nor any reference point is required that makes

GTSP robust to link/node failures; GTSP depends only on the local information of the nodes.

Apicharttrisorn *et al.* in [45] proposed an energy-efficient gradient time synchronization protocol (EGTSP) that is distributed, gradient-based and energy-efficient. This protocol is completely localized, achieves time consensus and gradient using drift estimation and incremental average estimation. In GTSP, every node estimates its clock by using the received time from all neighbors. According to this estimation, the global clock is adjusted. This adjustment can be large, this may cause some errors. In GTSP the broadcasting period is constant and therefore it has small trends that significantly consume sensor networks energy. Each node in EGTSP estimates the incremental average of time immediately after receiving the broadcasting packet from its neighbors. Whenever the incremental averaging is less, the global time is improved.

Qun et al. in [43] discussed a distributed time synchronization protocol (DTSC), it is consensus-based algorithm that uses to maintain only the clock offsets and neglecting the clock drifts. On the other hand, Cremaschi et al. in [46] discussed distributed frequency compensation i.e. clock drift compensation for phase locked loops (PLLs) using consensus techniques. Additionally, Carli in [] proposed a proportional-integral (PI) consensus-based controller that compensates both clock offset and clock drift. But, still these algorithms consume more energy to reach the synchronous state since the internal components are complex. This will reduce the lifetime of all nodes when they are deployed.

Schenato et al. in [22] proposed another consensus algorithm called average time sync (ATS) algorithm. It is an asynchronous consensus protocol and it is used to average the local time of the nodes to agree on the global synchronization in the network. Correspondingly, it is used to cascade the two consensus methods to estimate the clock parameters where the clock converges to a specific value. This algorithm has three main properties. First it is fully distributed and it is robust to node failure and it is easy to add a new node. Secondly, it maintains the clock skew differences among all nodes. Thirdly, it involves only simple computations like sum/product operations. ATS algorithm is adaptive to slowly time-varying clock drifts and need minimal memory and computational resources. Since ATS is a fully distributed communication topology, there are no specific nodes such as roots and all nodes run with the same algorithm; the nodes broadcast their local time to calculate the skew rates relative to each other. Thereafter, the nodes broadcast their current estimate of the skew rate. Finally, the receiving nodes measure the relative skew estimates depending on the skew rate of other nodes to justify their own virtual clock estimate.

CCS [47] uses average consensus algorithm to estimate and update the offset of each nodes clock. From the accumulation clock offset errors which are removed in each run of offset compensation, each node is able to observe the drift of their respective clocks from the global consensus time, then each node uses this data to compensate the clock drift. This method is observed to be an enhancement of ATS algorithm. This scheme is also fully distributed and just like the ATS but has a relatively of slow convergence speed.

Qun and Rus in [43] discussed a new time synchronization consensus protocol using maximum and average values called TSMA. The main idea is that this technique is based on the maximum and averaging time values to estimate the offset and skew values. This algorithm is fully distributed like ATS, does the skew compensation, contributes MAC- layer to increase the accuracy, does not need a root node, it is asynchronous, robust to node failure and replacement and high convergence speed compared to ATS. This algorithm uses average consensus to estimate the clock offset. It aims to obtain an internal agreement of the network on the time and how fast it travels. For each synchronization round, this algorithm updates the skew and offset for each node until the clocks converge to a specific value. Mainly, this process is divided into two parts; offset and skew estimation. In the offset estimation part, nodes exchange their local clocks to synchronize nodes to the same time. While in the skew estimation, nodes compare their current and previous values in each round to improve the accuracy of these parameters.

Jianping et al. in [48] presented the maximum time synchronization (MTS) protocol that depends on the maximum values and the objective is to maximize the local time to get global synchronization within the network. The benefits of this algorithm compared to other algorithms is that it has higher convergence speed with a finite value, compensate the skew/offset values at the same time, it is fully distributed, asynchronous, robustness to node failure and replacement or adding new nodes is easier. This algorithm pushes the nodes to get the maximum value of time for all nodes and each of these nodes broadcasts a packet with its local hardware clock and relative logical clock skew and offset, without any feedback data from the neighboring nodes. The same author proposed another algorithm in [48] called weighted maximum time synchronization (WMTS) protocol by taking care of the delay problem in the reception and transmission packets. In this algorithm there are two decision variables; source reference node and the number of hops where the logical clock information will be sent according to these variables from a source node to the receiver node. MTS and WMTS have many advantages over ATS and GTSP such as; GTSP and ATS have asymptotic convergence while MTS converges to the global synchronization with finite time. The convergence time of ATS and GTSP depends on the error value but MTS does not, and the compensation of skew and offset can be done simultaneously using MTS but in GTSP and ATS, offset will be started after skew has been completed. So, the MTS/WMTS has higher speed convergence compared to the other techniques. Moreover, these two algorithms are asynchronous, distributed, and robust to packet losses and node failure, replacement or relocation is possible or easier. On the other hand, WMTS needs a reference node in its operation.

In [49] a new scheme comprising two distributed time synchronization algorithms are presented for deployment in wireless sensor networks. These proposed scheme was based on method of multi-agent consensus and therefore do not need any master or leader node for time synchronization. Furthermore, these schemes use an event-based method for the enhancement of the efficiency in information transmission between network nodes. The author then analyzes the convergence characteristics. The method is shown to attain network synchronization at a certain precision with a minimal exchange packets between nodes. Which indicated that in clock synchronization is easily achieved up to some thresholds defined in the protocol design. Numerical computations and simulations carried out on some chosen networks showed the minimization of the rates of communication using this scheme.

Wu et al [50] proposed a novel scheme called clustered consensus time synchronization (CCTS) for the synchronization of nodes clocks in wireless sensor networks. The scheme is distributed in operation and is based on consensus. A clustering technique is then imbibed into this scheme so as to improve a higher convergence time and hence reduce the consumption of node battery power. This CCTS algorithm was classified into the intracluster and intercluster time synchronization. In the intracluster time synchronization, the enhanced DCTS is used. The cluster head is used for the exchange of packets in a particular domain or cluster. The mean value of drift compensation variables of intracluster virtual clock and the mean value of intracluster virtual clocks are employed for the update of the drift compensation parameter and offset compensation parameter, respectively. In the intercluster time synchronization, cluster heads exchange packets via gateway nodes. To update the clock compensation parameters of the network virtual clocks, clock compensation parameters of intracluster virtual clocks of every cluster head are assigned with corresponding weights based on the size of each cluster. Results obtained by simulating the algorithm using some selected networks. The simulation results showed a reduction in communication traffic as compared to traditional distributed consensus time synchronization algorithms and improves the rate of convergence due to the amalgamation of the clustering topologies. The algorithm however might be sensitive to node failures and dynamic topologies due to the leader nodes in each network cluster.

Yildirim *et al* [20] proposed a new control theoretic distributed time synchronization algorithm, named PISync, which is based on a Proportional-Integral (PI) controller. They presented a flooding-based and fully distributed PISync protocols which are based on PISync algorithm and observed their performances through real-world experiments and simulations. Their research revealed that PISync protocols have several superiorities over existing protocols in that, they do not store any distinct time information and have very little memory allocation overhead, have a very little CPU overhead, requires very little amount of information to be exchanged, have a very small code footprint, and are quite scalable in terms of steady state global synchronization error performance.

2.4.3 Comparison Between Distributed and Centralized Protocols

Generally, Distributed protocols are robust and flexible to the variations in the network topology and have a steady state value. Additionally, similar to other protocols they are affected by network propagation delays and noise. These protocols are characterized by low complexity iterative processes since the neighboring nodes can communicate with each other to achieve the agreement point depending only on the initial evaluations without the need of transmitting data to a reference point. Centralized protocols on the other hand, are generally easy to implement, and mostly comprised of pairwise synchronization followed by global network synchronization. However, these protocols have several drawbacks such as the high overhead behind constructing the whole tree structure, not suitable for operation in networks with changing topologies, and usually require more time and overhead when there is a new node added to the network or leader election required. Table 2.1 summarizes the merits and demerits of centralized and distributed time synchronization protocols.

Table 2.1: Comparison Between Distributed and Centralized Protocols

Centralized Protocols	**Distributed Protocols**
High overhead in network construction	High processing capabilities
High overhead in leader election	High memory in calculations
Sensitive to node and link failures	Robust to Link and Node failures
Lower convergence speed	Relatively high convergence speed
Relatively high power consumption	Requires less power to converge
Relatively high convergence time	Mostly takes less time to converge

2.5 Summary

In this chapter the topic of time synchronization was thoroughly discussed. First, the problem of time synchronization in WSNs was presented and models of hardware and logical clocks for sensor nodes was then reviewed. We then carried out an extensive literature survey on time synchronization protocols. We observed from our review that, most of the reported protocols require a high number of communication cycles to synchronize, since synchronization has to be carried out throughout the operation of the network. This makes the reported protocols wasteful in terms of energy consumption and hence will not be well suited for operation in networks deployed in harsh environments.

A lot of WSNs are, however, usually deployed in harsh environments under unstable conditions where regular and/or frequent communications might not be practical. This is challenging for the applications that consider time as an important factor in their operations. Harsh environments are unpredictable and uncontrolled sensor fields where factors such as vast fluctuation in temperatures, rain storms, vibration, humidity, chemicals, electrical shocks, pressure, mechanical stress may affect the normal operation of the nodes. Deployed WSNs must achieve synchronization while facing these obstacles coupled with the goal to achieve other primary requirements including being computationally light, scalable, and robust to node and link failures. Most of the reviewed protocols do not take the above stated challenges into consideration in their design. Chapter 3 focuses on a novel method of synchronization that addresses the problem of operation in harsh environments. The presented method addresses the synchronization problem by focusing on consensus control, nearest neighbor communication and synchronization to network gateway node.

CHAPTER 3

PROBLEM DEFINITION, STATEMENT AND FORMULATION

3.1 Problem Definition and Statement

For a wireless sensor network to operate properly, all nodes in the network have to achieve a consensus on a common time. Although several methods have been suggested to accomplish this task, these methods are designed based on some assumptions that make them unsuitable for operation in harsh environments. Most methods adopted for time synchronization in wireless sensor networks normally assume specific stationary network connectivity, a multi-hop communication and/or a node labeling scheme, etc [22, 34]. Moreover, sensor nodes employed for industrial applications are normally cheap and small with limited resources [32], have a wide range of drifts: usually between 30 to 100 ppm [29] and are mostly deployed in unstructured dynamic networks [26, 37]. Therefore, time synchronization protocols operating under such assumptions will not effectively synchronize nodes under these stated conditions and would have to constantly resynchronize to ensure global network synchronization which sharply depletes node energy and hence, network lifetime.

In this work, we adopt a method suitable for operation in harsh environments that uses a single hop communication scheme to synchronize state parameters representing the virtual clock of each node to the clock of the gateway node of the wireless sensor network. This decentralized approach to time synchronization was first introduced in [51].

A set of nodes in a distributed network can communicate through the exchange of information where each node initially holds a measured or computed parameter and wants to learn in a distributive way, the average of all the measurements of the other nodes in the network. Distributed average consensus algorithms are designed to solve this problem. Node units in the network do not necessarily have a thorough global knowledge about the network. For example nodes might not be aware of the number of nodes, the network topology, or the type of quantities collected at other nodes, etc. Moreover, in some applications or network frameworks, the network topology can vary with time due to link instability or node mobility. The goal of the average consensus algorithm is to reach consensus in a reliable and robust manner. Average consensus algorithms operate iteratively and the instantaneous value at each node is an estimate of the measurements average. These algorithms are designed such that all the estimates in a particular network converge to the sought average up to any desired level of precision. The iterative process is classed into three parts.

1. First one or several nodes wake up.
2. Then the woken nodes send their estimates to one or several neighbors in the network.
3. Finally each receiving node updates its estimate to a value which depends on its current estimate and on the estimates it has received from the woken nodes.

In a synchronous algorithm, all the nodes in the network wake up at each instance of iteration and broadcast their estimates. All nodes in the network then update their estimates before the next iteration instance can begin. On the other hand, in an asynchronous algorithm, only one node or a subset of nodes wake up at each iteration. These node(s) call some chosen neighbors. Only a subset of nodes in the network update their estimates at the end of each iteration.

3.2 Suggested Synchronization Procedure

In this thesis, we present a protocol that accomplishes time synchronization by adopting the average consensus approach through the exchange of the global estimated time values of each node with its neighboring nodes until all network nodes reach an agreement in global time within acceptable error margins. This method is inspired by the synchronization approach presented in [51, 52]. In this approach, a decentralized dynamical system is used to drive the virtual clocks of each network node to the time of the gateway node. Through interaction with one hop neighbors, the synchronization procedure can drive the local time of each node to the time of a gateway node until all the nodes converge to almost the same time. Once a network node achieves a minimum synchronization error with respect to the gateway node, the node halts its communication and the local physical(logical) clock of each node is reset to the time estimate at this minimum error. The task of detecting the instant of minimum synchronization error is carried out using a stopping criterion derived from the dynamic nature of the local time series estimate. At this instant of minimum error, the nodes stop updating so as to conserve energy and memory [52]. The nodes are made to resume update once the nodes' clocks drift beyond a certain threshold or after a predetermined period. The updates are then halted after a consensus is reached. This process is carried out continuously to keep the network synchronized as shown in the block diagram of Figure 3.1. Detailed discussion of each stage or component of the synchronization procedure is presented.

3.2.1 Node Local Time Update

In [51, 52], a synchronous average consensus approach is adopted for local time update at each network node. This approach assumes a simultaneous update for

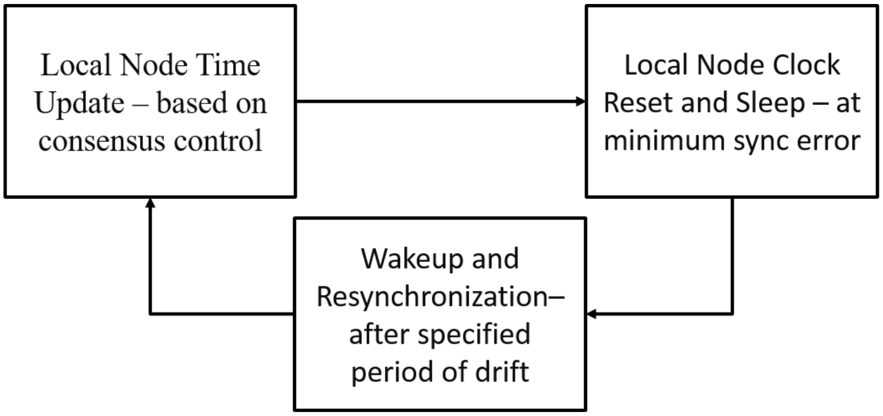

Local Node Time Update – based on consensus control
Local Node Clock Reset and Sleep – at minimum sync error
Wakeup and Resynchronization– after specified period of drift

Figure 3.1: Block Diagram of Suggested Synchronization Procedure

all network nodes at each communication round as described below.

Consider a network having N nodes comprising of 1 gateway node, g and $N-1$ ordinary nodes as shown in Figure 3.2. If there exist a spanning path from a certain node i in the network to the gateway node [51], the update of its time estimate can be expressed in terms of the neighborhood time values, $t_j(k)$ and the gateway time, $t_g(k)$.

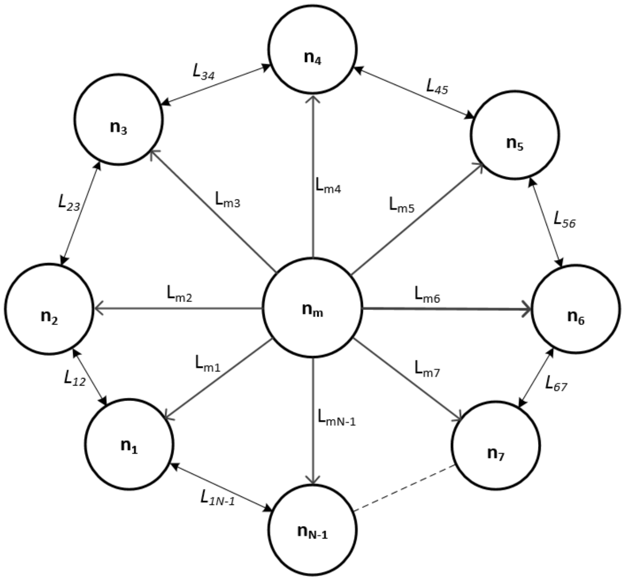


Figure 3.2: Network of N Nodes and L Links

If node i has n nearest neighbors [1], where $n < N - 1$, then at the k^{th} time instant when the node receives $t_j(k-1)$, a new time average at node i denoted by $t_i(k)$ is calculated using the relation:

$$t_i(k) = \frac{1}{n} \sum_{j=1, j \neq i}^{N-1} t_j(k-1) \tag{3.1}$$

In this synchronization procedure, it is assume the gateway node ticks as a perfect clock given by $t_g(k) = \Delta k$, where Δ is the ticking rate of the gateway clock. If i is also connected to the gateway node, then the update is expressed in terms of both neighbors' time estimate and gateway clock as;

$$t_i(k) = \frac{1}{n} \sum_{j=1, j \neq i}^{N-2} t_j(k-1) + \frac{1}{n} \Delta k \tag{3.2}$$

At the k^{th} iteration, each node yields time estimates $\mathbf{T}_k$. Hence a general linear state equation relating all values in the iterative process can be given by:

$$\mathbf{T}_k = \mathbf{A}\ \mathbf{T}_{k-1} + \Delta k \mathbf{B} \tag{3.3}$$

Hence the network acts like a discrete linear dynamic system with a discrete-time ramp input signal with an increment of Δ, where the output of the first component of the synchronizer is a time series of individual node time estimates, representing the virtual clocks of all network nodes, i.e. $\mathbf{T}_k$ which

---

[1] It is assumed that, all node that can communicate with node i in a single hop are nearest neighbors

is a vector of the entire network time estimates also expressed as: $\mathbf{T}_k = [t_1(k)\ t_2(k)\ t_3(k)\ \cdots\ t_{N-1}(k)]^{\mathbf{T}}$

$\mathbf{B}$ is an $(N-1)$x1 matrix representing the connectivity matrix between nodes and the gateway node and $\mathbf{A}$ is an $(N-1)$x$(N-1)$ matrix representing the connectivity matrix between nodes and their neighbors excluding the gateway node. The entries of matrices $\mathbf{A}$ and $\mathbf{B}$ are respectively given by;

$$a_{ij} = \begin{cases} \frac{1}{n}, & \text{if nodes } i \text{ and } j \text{ are neighbors} \\ 0, & \text{otherwise} \end{cases}$$

$$b_{i1} = \begin{cases} \frac{1}{n}, & \text{if node } i \text{ is connected to the gateway node} \\ 0, & \text{otherwise} \end{cases}$$

The error at time k with respect to the gateway clock is given by;

$$\mathbf{E}_k = \mathbf{T}_k - k\Delta\mathbf{1} \tag{3.4}$$

This error can also be represented by the difference equation given by;

$$\mathbf{E}_{k+1} = \mathbf{A}\mathbf{E}_k + \Delta\mathbf{B} - \Delta\mathbf{1} \tag{3.5}$$

It is shown in [51] that if the gateway node is connected to at least one of the network nodes, i.e. at least $b_{i1} = 1$ for any node i in the network, then if there exist a spanning path to the gateway node, the system is shown to be marginally stable with a steady state error vector given by;

$$\mathbf{E}_{ss} = [\mathbf{I} - \mathbf{A}]^{-1} \times [\Delta\mathbf{B} - \Delta\mathbf{1}] \tag{3.6}$$

Where, $\mathbf{E}_{ss} = \lim_{k\to\infty} \mathbf{E}(k)$ and $\mathbf{1}$ is an all-one vector.

3.2.2 Dip Phenomenon in Suggested Synchronizer

It is shown in [51] that, the synchronization algorithm represented by the state equation given in (3.3) can drive all network nodes' times to the clock of the gateway node with a very small error margin if the rate of the gateway clock is significantly reduced. This solution is not practical since an increase in the gateway clock ticking rate would mean a higher number of communication cycles, which would lead to high consumption of node power [51]. It is observed, however, from the evolution of node local clock value with respect to the gateway clock that, there exist a certain time estimate, $t_i(k)$, where the node's local time curve intersect with that of the gateway clock curve as illustrated by Figure 3.3. This phenomenon is observed in node error profile curves with respect to the gateway time as a dip before steady state as shown in Figure 3.3. Ideally, at this point of intersection labeled as *Transient Error* in Figure 3.3, a node i in the network has to be perfectly synchronized to the gateway node. However, for a certain number of communication cycles, an absolute error well below Δ can be achieved between the clock of node i and the gateway node [51]. To harness this advantage, a scheme has to be devised to make nodes recognize and stop at this point of intersection. If this task is achieved, synchronization can be reached at maximum accuracies using minimal number of communication cycles. This is expected to make our proposed method of synchronization outperform several of the reported synchronization strategies in the literature in terms of memory and energy consumption.

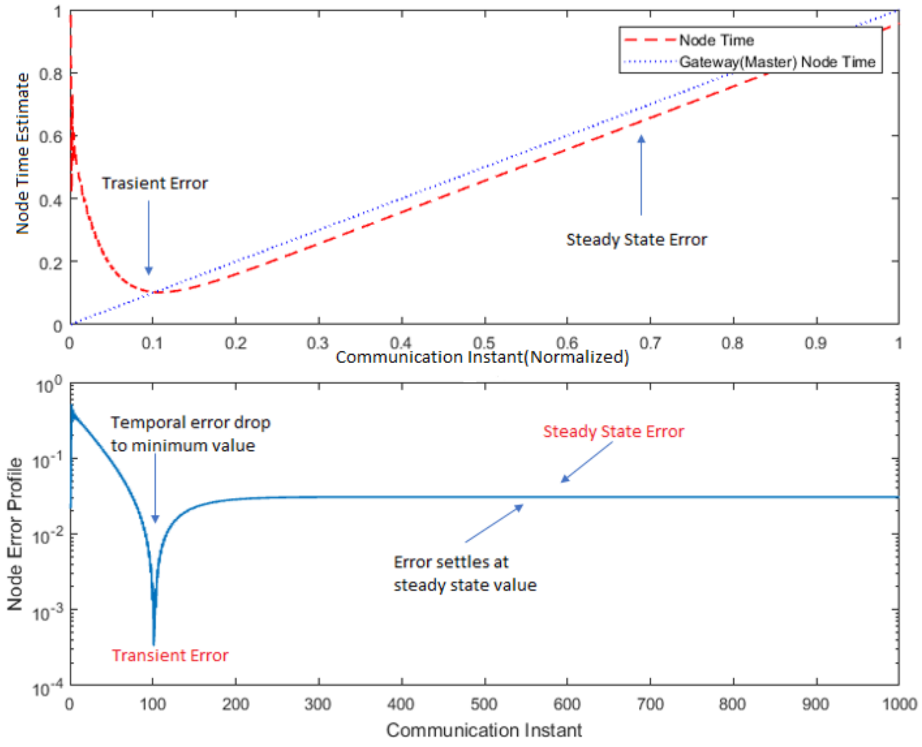


Figure 3.3: Transient (Dip) and Steady State Behavior of Node Clocks with respect to Gateway Node Clock

3.2.3 Criterion for Stopping in Dip Region

To exploit the dip nature of the evolving node time estimates, a stopping criterion is needed to halt the iterative process of each node before steady state is reached. This solution might come in the form of a detection filter which operates on each node's time series to detect the minimum error at the dip region. This filter will produce a signal to indicate where in the time series a node's time estimate is closest to the gateway time. To design an optimum filter that achieves the cessation of updates at the dip requires rigorous analysis of the dynamic linear equations presented by Equation 3.3 which is beyond the scope of this work. However, in [51], a heuristic finite impulse response filter which exploits the turning point of the time curve is employed to provide the indicator signal for detecting the dip region. Through experimentation, it was reported that the zero-crossings of the filter output corresponds with high probability, to the instant of error dip. An example of such a filter was reported in [52, 51] to have an impulse response, $h(k)$ given by,

$$h(k) = 0.2\delta(k+3)+0.5\delta(k+2)+0.2\delta(k+1)-0.2\delta(k-3)-0.5\delta(k-2)-0.2\delta(k-1) \tag{3.7}$$

Where δ is the Khronecker delta function. The above filter is a typical example of a difference filter of length 7.

3.2.4 Local Clock Reset and Resynchronization

When the filter indicates the dip region, an algorithm is used to locate the iterative instant corresponding to this minimum time estimate. To effectively carry out this task, the consensus algorithm used for local node update has to achieve the dip at almost the same time instant. If the variance in the instances of reaching the dip region for all network nodes is nearly null, then the scheduling, wakeup and sleep of the network can be carried out in unison which would make synchronization more energy efficient. In [51], it is reported that this algorithm used for the stopping criterion is based on polarity change of the filter output provided a certain number of communication cycles have passed. It was further indicated that, 11 communication cycles, representing the very rough transient period at the beginnings of synchronization, were optimal for all the cases tested. Once the time estimate at the transient error is obtained, it is used to reset the physical local clock of the node. After which it seizes to update and goes into sleep mode. The node then awakes after a certain predetermined period when the clocks of the network nodes drift beyond a certain threshold to start. In [37], experiments were conducted using mica and mica2 mote platforms to determined the resynchronization period. The authors reported a resynchronization period of $30s$ gave the best results but showed that, when skew compensation is taken into consideration in protocol design, the resynchronization period can go up to several minutes depending on the accuracy requirements of the specific application of the wireless sensor network. In [21], it is reported that if a node only does offset compensation during synchronization, given a node with a constant drift rate, x ppm, if an application desires an accuracy of δ *seconds*, a protocol has to resynchronize every $\frac{\delta}{x \times 10^{-6}} seconds$. Where x is the accuracy of the crystal oscillator of the node clock given in *parts-per-million (ppm)*. Hence an oscillator with $100ppm$ running at 1 MHz drifts apart 100μs in one second.

3.2.5 Generalized Algorithm for Suggested Synchronization Procedure

To summarize the suggested scheme for time synchronization in wireless sensor networks, the pseudo-code presented by **Algorithm 1** and illustrated in Figure 3.4 for a node i having n nearest neighbors and deployed in a network of size N is presented in this subsection. Node i maintains two variables related to its clock: a time estimate, t_i which is an estimation of the gateway clock representing the global estimate and a logical clock variable, C_i which is a representation of the physical clock of node i and related to the hardware clock, T_i by Equations 2.11 and 2.13 [2]. Due to memory constraints, initially when each node is turned on or comes online, the stored data for the previous synchronization cycle except for the core heuristics in the node repository are cleared. Node i then initializes the variable t_i to the current hardware clock value, $T_i(0)$ and set its beaconing rate, R to a predetermined resynchronization period [3]. If a synchronization packet from a neighbor, j is received, node i calculates its time estimate, $\hat{t}$ using a local time update algorithm and sets t_i to $\hat{t}$. In [51, 52], the synchronous average consensus algorithm is used to compute, $\hat{t}$ [4] as discussed in Section 3.2.1. It is assume that with single hop communication, node i receives packets from all neighbors at approximately the same time and hence $\hat{t}$ is computed using the t_j's from all n nodes.

---

[2] Refer sections 2.2.2 and 2.3.1.1

[3] Refer section 3.2.4

[4] Refer section 3.2.1

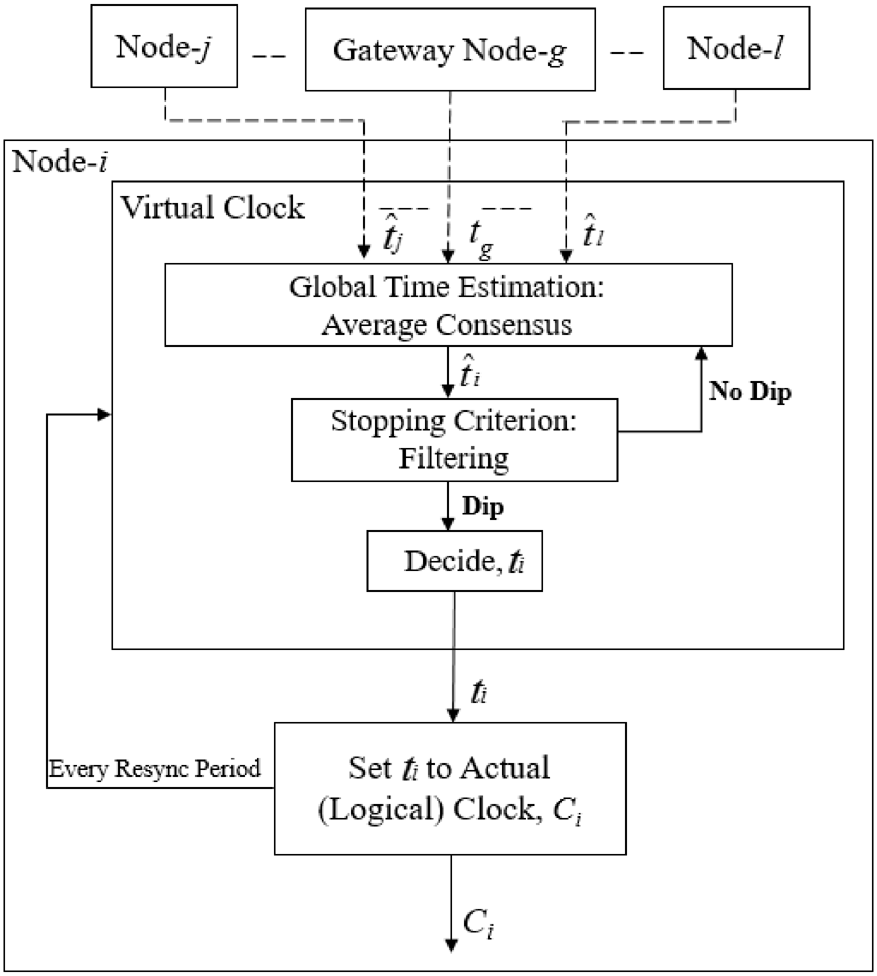


Figure 3.4: Suggested Clock Setup for Node Synchronization

For each time estimate, t_i, node i uses a stopping criterion to check if it is in the transient or dip region of minimum error [5]. If t_i is at the transient stage, then the logical clock C_i set to t_i and the node seizes communication. On the other hand if t_i is not in the transient stage, the node broadcasts it's synchronization packet and reinitialize the synchronization algorithm. If the former is fulfilled, the node starts a timer to fire after R seconds ticks of the hardware clock and then goes to sleep. Upon timer timeout, the node wakes up and reinitializes the

---

[5] Refer to section 3.2.3

Algorithm 1: Suggested Synchronization Procedure Pseudo-Code for Node i

```
☐ Initialization
  Wakeup to resynchronize to network
  Clear repository
  $t_i \longleftarrow T_i(0)$; $R \longleftarrow$ Predetermined resynchronization period;
☐ Upon receiving $< t_j >$ from n nodes
  Compute neighborhood time estimate, $\hat{t}$ using Local Time Update Algorithm
  set $t_i \longleftarrow \hat{t}$
☐ For each estimate $< t_i >$
  If $< t_i >$ is in the transient phase then
    set $C_i \longleftarrow t_i$ and seize communication
  elseif $< t_i >$ is not in the transient phase then
    transmit $< t_i >$ and reinitialize algorithm
  endif

☐ Upon $C_i$ set and communication seized
  Set timer to fire after $R$ seconds
  Node goes to sleep mode

☐ Upon timer timeout
  Node wakeup and reinitialize algorithm
```

synchronization algorithm.

3.3 Formulation and Methodology

In this section, the problem of synchronization with respect to the suggested method for synchronizing network nodes is analyzed and formulated. The proposed methods adopted to improve the suggested synchronizer given in [52, 51] are presented. These improvements are carried out on the first component of the proposed synchronizer, i.e. the local node time update [6].

---

[6] Refer to Figure 3.1, section 3.2

3.3.1 Accuracy-Enhanced Method for Time Synchronization in WSN using Synchronous Average Consensus Control

Equation 3.5 gives a representation of each nodes time error with respect to the gateway time as a function of discrete time. It can be observed for (3.5) that this error depends on the connectivity matrices **A** and **B** and the ticking rate of the gateway node, Δ. It can be observed that, the last term of Equation 3.5 is a constant error vector, $\Delta\mathbf{1}$. Since Δ is alway known, the error of each node can be minimized by reducing the effect of this constant vector. Hence the local node time estimate given in Equation 3.2 can be modified to by including a parameter, e which is a scalar factor of Δ. This modified average node time estimate is given by;

$$t_i(k) = \frac{1}{n} \sum_{j=1, j \neq i}^{N-2} t_j(k-1) + \frac{1}{n}\Delta k + e \tag{3.8}$$

This parameter, e, is observed to improve synchronization of each node at the expense of increased cycles of communication and hence presents a designer of a WSN with flexibility since some applications need a higher synchronization accuracy whereas others are energy efficiency centric.

3.3.2 Drawbacks in the Synchronous Local Node Time Update

The synchronous local update approach adopted in [51, 52] is able to drive all nodes' time estimates to that of the gateway node but the assumption of simultaneous updates of all nodes at each instant of communication is not practical. This stems from the fact that all nodes will have to be switched on almost at the same time and all nodes must also have some initial conception of network time during operation. In other words, synchronization is needed to carryout a synchronous local node time update. This makes the use of synchronous consensus algorithms for time synchronization paradoxical. Therefore we adopt practical approaches based on asynchronous local time updates which does not have these setbacks for reaching consensus among network nodes.

3.3.3 Suggested Asynchronous Local Node Time Update

For the asynchronous averaging process, since only a subset of nodes carry out the averaging at each instant, the state linear equation given in 3.3 includes an activation matrix, J_k which is a representation of the subset of network nodes that update at each iteration. The dynamic state space representation of the asynchronous local time update is therefore given by;

$$\begin{aligned} \mathbf{T}_k &= \mathbf{J}_k(\mathbf{A}\ \mathbf{T}_{k-1} + \Delta k \mathbf{B}) \\ \mathbf{J}_k \mathbf{1} &= \mathcal{F}(\mathbf{J}_{k-1}\mathbf{1}) \end{aligned} \tag{3.9}$$

Where,

$$\mathbf{J}_k = \begin{bmatrix} J_{1k} & 0 & \dots & 0 \\ 0 & J_{2k} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & J_{Nk} \end{bmatrix}$$

The non-zero components of $\mathbf{J}_k$ at discrete time, k are expressed as a of function their state values at the previous time, $k-1$ and the activation function $\mathcal{F}(.)$ is determined by the graph connectivity of the WSN. The entries of J_{ik} are expressed as $J_{ik} = \begin{cases} 1, & \text{if node } i \text{ updates at time } k \\ 0, & \text{otherwise} \end{cases}$

The local node time estimate for node i at discrete time k can therefore be represented as;

$$t_i(k) = \frac{J_{ik}}{n} \left[\sum_{j=1, j\neq i}^{N-2} t_j(k-1) + \Delta k \right] \tag{3.10}$$

Where node i is activated for update when J_{ik} is $'1'$ and inhibited from update when J_{ik} is $'0'$. The activation variables are used to control the local time update of each node and is determined by the graph of the wireless sensor network. The subset of nodes that update at any time instant also depend on their respective virtual clock values in the previous instant.

Based on this framework, we develop three procedures for synchronization where the node local time update is done in an asynchronous manner. In the first method, synchronization is carried out by making one node conduct the averaging process at each instant in a sequential manner. The updates are done in a chronological order based on proximity to the gateway node. This method is dubbed *Timed Sequential Asynchronous Update.* Since some network nodes can be at an approximately equal distance from the gateway node, the second method named *Unidirectional Asynchronous Flooding*, which improves upon the first method is designed to make such nodes update at the same communication instant, hence reducing the convergence time and saving node energy. However, this method requires the gateway node to trigger a wake-up protocol for the asynchronous waking of the nodes. The third method called the *Bidirectional Asynchronous Flooding* eliminates the need for wake-up triggers by the gateway node using a two-way triggering protocol conducted by the closest and furthest node(s) to the gateway node. This method is hence adaptive to changes in network topology and is self regulating in terms of wake-up and sleep thus is best suited for networks deployed in harsh environments where node failures and topology changes occur frequently. It is worth noting that, Equation 3.9 becomes a synchronous system represented by Equation 3.3 if $J_{ik} = 1, \ \forall i \in N$ or $\mathbf{J}_k = \mathbf{I}$, hence (3.9) represents a somewhat general form of all the methods of node local time update.

3.4 Thesis Contributions

In this chapter, we presented our proposed synchronizer and gave a detailed description of its components. The proposed synchronization procedure is based on consensus control, single hop communications and synchronization to the clock of a gateway node which is based on the synchronizer presented in [51]. The main components of the proposed synchronizer are first, the local node update where average consensus is adopted for establishing a common time estimate for all nodes. The mathematical representation of the local node time update and dynamic state space linear equations are presented for the synchronous average consensus presented in [51]. This is followed by a filtering process to detect the instance of minimum error in the dip region, representing the second component of the synchronizer. Finally, in the third component, each node is made to initialize its physical clock to the time at the detected iterative instance. We then present and address the drawbacks of the synchronous average consensus adopted for the local time update in the previous work. This was followed by a description of the main contributions of this work which are to:

1. Propose an accuracy-enhanced method for synchronizing sensor nodes using synchronous average consensus.
2. Remove the need of synchronous local time update by extending to an asynchronous framework for local time update which is more practical.
3. Present three methods to practically achieve synchronization using the asynchronous local node time update.

The rest of this thesis is organized as follows. Chapter 4 gives a detailed exposition on the design flexibility for the protocol and presents the general methodology for simulations and practical experiments adopted in this work. In chapter 5, we formulate and present the methods adopted for the suggested asynchronous framework for local time update. The simulation and practical experimental results for each method are also presented in this chapter. A comprehensive comparative practical evaluation of our proposed methods with some reported protocols in the literature are presented in Chapter 6. Chapter 7 ends the report by concluding the thesis work and gives some recommendations for future research.

CHAPTER 4

ACCURACY-ENHANCED METHOD FOR TIME SYNCHRONIZATION IN WSN USING SYNCHRONOUS AVERAGE CONSENSUS CONTROL

4.1 Introduction

The original averaging consensus control method is able to drive nodes in a wireless sensor network by letting each node trace the time of a perfect clock (gateway node clock) and simultaneously averaging its clock values with that of its neighbors. Hence for that proposed method to run in an optimized mode, each nodes clock at a certain discrete time, k is supposed to be as close to the perfect(gateway) clock as possible. To understand the behavior of the algorithm, the evolution of the error existing between each node clock, $t_i(k)$ and the perfect gateway clock $t_g(k) = k\Delta$ has to be carefully analyzed. Here, a somewhat detailed analysis of this error is presented. From the analyses, we devise a method of improving the accuracy in synchronization of the suggested method. To examine this concept, we perform simulations and practical experiments on nodes of different networks. From these simulations and experiments, we show that the local synchronization accuracy of each node can be improved at the expense of increased number of communication cycles. Simulations and practical experiments and the evaluation results of the modified method are also presented and discussed in this chapter.

4.2 Accuracy-Enhanced Method for Time Synchronization: Concept

Given a wireless sensor network having N nodes comprising of 1 gateway node, g perfectly ticking at a rate, Δ, and $N-1$ ordinary nodes subject to drifts, the error at time k with respect to the gateway clock is given by;

$$\mathbf{E}_k = \mathbf{T}_k - t_g(k)\mathbf{1} = \mathbf{T}_k - k\Delta\mathbf{1} \tag{4.1}$$

$\Rightarrow \mathbf{T}_k = \mathbf{E}_k + k\Delta\mathbf{1}$ and similarly $\Rightarrow \mathbf{T}_{k+1} = \mathbf{E}_{k+1} + (k+1)\Delta\mathbf{1}$

Inserting $\mathbf{T}_{k+1}$ and $\mathbf{T}_k$ in Equation 3.3, we have

$$\mathbf{E}_{k+1} + (k+1)\Delta\mathbf{1} = \mathbf{A}[\mathbf{E}_k + k\Delta\mathbf{1}] + (k+1)\Delta\mathbf{B}$$

$$\mathbf{E}_{k+1} = \mathbf{A}[\mathbf{E}_k + k\Delta\mathbf{1}] + k\Delta\mathbf{B} - (k+1)\Delta\mathbf{1} + \Delta\mathbf{B}$$

$$\mathbf{E}_{k+1} = \mathbf{A}\mathbf{E}_k + \Delta k\mathbf{A}\mathbf{1} + \Delta k\mathbf{B} - k\Delta\mathbf{1} - \Delta\mathbf{1} + \Delta\mathbf{B}$$

$$\mathbf{E}_{k+1} = \mathbf{A}\mathbf{E}_k + \Delta k[\mathbf{A}\mathbf{1} + \mathbf{B} - \mathbf{1}] + \Delta\mathbf{B} - \Delta\mathbf{1}$$

But the network connectivity matrices are such that, $\mathbf{A}\mathbf{1} + \mathbf{B} = \mathbf{1}$

Hence,

$$\mathbf{E}_{k+1} = \mathbf{A}\mathbf{E}_k + \Delta k[\mathbf{1} - \mathbf{1}] + \Delta\mathbf{B} - \Delta\mathbf{1}$$

Which Implies that,

$$\mathbf{E}_{k+1} = \mathbf{A}\mathbf{E}_k + \Delta\mathbf{B} - \Delta\mathbf{1} \tag{4.2}$$

This can be rewritten as;

$$\mathbf{E}_{k+1} = \mathbf{A}\mathbf{E}_k + \mathbf{R}, \text{ where } \mathbf{R} = \Delta\mathbf{B} - \Delta\mathbf{1}. \tag{4.3}$$

Equation 4.2 shows that, at some discrete time, $k+1$, the error vector, $\mathbf{E}_{k+1}$ depends on the network connectivity matrices $\mathbf{A}$ and $\mathbf{B}$, the previous error vector, $\mathbf{E}_k$ and the constant rate, Δ of the gateway clock.

To further probe the nature of the error vector, we solve Equation 4.3 iteratively. This goes as follows:

$$\mathbf{E}_1 = \mathbf{A}\mathbf{E}_0 + \mathbf{R}$$

Similarly;

$$\mathbf{E}_2 = \mathbf{A}\mathbf{E}_1 + \mathbf{R} = \mathbf{A}[\mathbf{A}\mathbf{E}_0 + \mathbf{R}] + \mathbf{R}$$

$$\mathbf{E}_2 = \mathbf{A}^2\mathbf{E}_0 + \mathbf{R}\mathbf{A}\mathbf{1} + \mathbf{R}$$

$$\mathbf{E}_3 = \mathbf{A}[\mathbf{A}^2\mathbf{E}_0 + \mathbf{R}\mathbf{A}\mathbf{1} + \mathbf{R}] + \mathbf{R}$$

$$\mathbf{E}_3 = \mathbf{A}^3\mathbf{E}_0 + \mathbf{R}\mathbf{A}^2\mathbf{1} + \mathbf{R}\mathbf{A}\mathbf{1} + \mathbf{R}$$

$$\mathbf{E}_4 = \mathbf{A}[\mathbf{A}^3\mathbf{E}_0 + \mathbf{R}\mathbf{A}^2\mathbf{1} + \mathbf{R}\mathbf{A}\mathbf{1} + \mathbf{R}\mathbf{1}] + \mathbf{R}$$

$$\mathbf{E}_4 = \mathbf{A}^4\mathbf{E}_0 + \mathbf{R}\mathbf{A}^3\mathbf{1} + \mathbf{R}\mathbf{A}^2\mathbf{1} + \mathbf{R}\mathbf{A}\mathbf{1} + \mathbf{R}$$

$$\vdots$$

$$\mathbf{E}_k = \mathbf{A}^k\mathbf{E}_0 + \mathbf{R}\mathbf{A}^{k-1}\mathbf{1} + \mathbf{R}\mathbf{A}^{k-2}\mathbf{1} + \cdots + \mathbf{R}\mathbf{A}\mathbf{1} + \mathbf{R}$$

$$\mathbf{E}_k = \mathbf{A}^k\mathbf{E}_0 + \mathbf{R}[\mathbf{A}^{k-1}\mathbf{1} + \mathbf{A}^{k-2}\mathbf{1} + \cdots + \mathbf{A}\mathbf{1} + \mathbf{1}]$$

which can be reconfigured as;

$$\mathbf{E}_k = \mathbf{A}^k\mathbf{E}_0 + \mathbf{R}[\mathbf{1} + \mathbf{A}\mathbf{1} + \cdots + \mathbf{A}^{k-2}\mathbf{1} + \mathbf{A}^{k-1}\mathbf{1}]$$

Hence we have,

$$\mathbf{E}_k = \mathbf{A}^k\mathbf{E}_0 + \sum_{j=0}^{k-1} \mathbf{A}^{k-j-1}\mathbf{R},\ k \geq 1 \tag{4.4}$$

From Equation 4.4, the error vector, $\mathbf{E}_k$ could be described as the solution of the state equation of a discrete linear time varying system with input given by $u(k) = 1$ for $k \geq 0$ [53] and simplified as;

$$\mathbf{E}_k = \mathbf{A}^k\mathbf{E}_0 + \sum_{j=0}^{k-1} \mathbf{A}^{k-j-1}(\Delta\mathbf{B} - \Delta\mathbf{1}),\ k \geq 1 \tag{4.5}$$

$$\mathbf{E}_k = \mathbf{A}^k\mathbf{E}_0 - \Delta\sum_{j=0}^{k-1} \mathbf{A}^{k-j-1}\mathbf{1} + \Delta\sum_{j=0}^{k-1} \mathbf{A}^{k-j-1}\mathbf{B},\ k \geq 1$$

$$\mathbf{E}_k = \mathbf{U}_k + \mathbf{V}_k$$

We note from Equation 4.5 that this solution has two distinct parts; a homogeneous part, $\mathbf{U}_k = \mathbf{A}^k\mathbf{E}_0$ depends only on the initial state error $\mathbf{E}_0$ and the other, $\mathbf{V}_k = \sum_{j=0}^{k-1} \mathbf{A}^{k-j-1}\mathbf{R}$ depends on the gateway connectivity matrix, $\mathbf{B}$ and the rate of the gateway node, Δ. These terms could be respectively called the natural or unforced or zero-input error response, and the zero-state error response [53, 54, 55, 56]. With a constant input, $u(k) = 1,$ for $k \geq 0$, the system stability could be said to mainly depend on the natural error response [53]. It is worth noting here that, the zero-state response, $\mathbf{V}_k$ has a form that is reminiscent of a convolution sum.

4.2.1 System Stability

From stability theory [53, 48], the nature of $\mathbf{E}_k$ is such that, the system is asymptotically stable if $\mathbf{U}_k$ converges to zero, which occurs if and only if the magnitude of all the eigenvalues, λ_i of the connectivity matrix $\mathbf{A}$ are less than 1 [53, 54].

Further analysis of this system is needed to explain the reason and nature of the dip feature of the protocol. i.e.,

$$\lim_{k\to\infty} \mathbf{E}_k = \lim_{k\to\infty} \mathbf{A}^k \; \mathbf{E}_0 = \mathbf{0} \text{ iff } |\lambda_i| < 1$$

Three other cases of stability could be considered [57];

1. If one or more eigenvalues, or pair of conjugate eigenvalues, has a magnitude larger than one, there is at least one corresponding modal component that increases exponentially without bound from any initial condition, violating the definition of stability.
2. Any pair of conjugate eigenvalues that have magnitude equal to one, $\lambda_{i,i+1} = e^{\pm j2\pi f}$, generates an undamped oscillatory component in the state response. The magnitude of the homogeneous system response neither decays nor grows but continues to oscillate for all time at a frequency, f. Such a system is defined to be marginally stable.
3. An eigenvalue $\lambda = 1$ generates the exponent $\lambda_k = 1^k = 1$ that is a constant. The system response neither decays or grows, and again the system is defined to be marginally stable.

4.2.2 Hypothesis for Accuracy-Enhancement

From the analyses above, we observe that Equations 4.2 and 4.5 have a constant vector of the gateway clock rate, $\Delta\mathbf{1}$ and since Δ is always known, we predict that if we include Δ or a fraction of Δ in each node average during the local time update, the first part of $\mathbf{V}_k$ is reduced and this will decrease the overall node error with respect to the gateway clock. Hence the local update Equations 3.2 and 3.3 are modified to include a parameter, e, which is a fraction of Δ. These are given by;

$$t_i(k) = \frac{1}{n} \sum_{j=1, j \neq i}^{N-2} t_j(k-1) + \frac{1}{n} \Delta k + e, \text{ and} \tag{4.6}$$

$$\mathbf{T}_k = \mathbf{A} \ \mathbf{T}_{k-1} + \Delta k \mathbf{B} + e\mathbf{1} \tag{4.7}$$

If a similar logic of derivation is followed, the vector $\mathbf{R}$ becomes, $\mathbf{R} = \Delta\mathbf{B} - \Delta\mathbf{1} + e\mathbf{1}$. Equations 4.2 and 4.5 then respectively become;

$$\mathbf{E}_{k+1} = \mathbf{A}\mathbf{E}_k + \Delta\mathbf{B} - \Delta\mathbf{1} + e\mathbf{1}, \text{ and} \tag{4.8}$$

$$\mathbf{E}_k = \mathbf{A}^k \mathbf{E}_0 + \sum_{j=0}^{k-1} \mathbf{A}^{k-j-1} (\Delta\mathbf{B} - \Delta\mathbf{1} + e\mathbf{1}), \ k \geq 1$$

$$\mathbf{E}_k = \mathbf{A}^k \mathbf{E}_0 + e \sum_{j=0}^{k-1} \mathbf{A}^{k-j-1} \mathbf{1} - \Delta \sum_{j=0}^{k-1} \mathbf{A}^{k-j-1} \mathbf{1} + \Delta \sum_{j=0}^{k-1} \mathbf{A}^{k-j-1} \mathbf{B}, \ k \geq 1 \tag{4.9}$$

□ **Remark 4.1**: It is expected that, an increase in the variable e towards Δ will lead to a corresponding decrease in overall error and since Equation 4.10 is general, the decrease in error is expected to also occur in the dip region. Since an in-depth mathematical analysis of the dip feature is not yet complete, we use simulations and practical experiments to investigate the behavior of this decrease in dip region error. As we will show in section 4.4 that, the parameter, e is observed to give a trade-off between accuracy in synchronization in the dip region and the number of communication cycles needed to reach the dip region.

Although the crucial aspect of the proposed synchronizer is the dip region, it is worth mentioning that, with the inclusion of e, the steady state error vector given by Equation 3.6 becomes;

$$\mathbf{E}_{ss} = [\mathbf{I} - \mathbf{A}]^{-1} \times [e\mathbf{1} + \Delta\mathbf{B} - \Delta\mathbf{1}] \tag{4.10}$$

Also, if $e = \Delta$, then $\mathbf{R} = \Delta\mathbf{B}$ and Equations 4.9 and 4.10 respectively become;

$$\mathbf{E}_{k+1} = \mathbf{A}\mathbf{E}_k + \Delta\mathbf{B}, \text{ and} \tag{4.11}$$

$$\mathbf{E}_k = \mathbf{A}^k\mathbf{E}_0 + \sum_{j=0}^{k-1} \mathbf{A}^{k-j-1}\Delta\mathbf{B}, \ k \geq 1 \tag{4.12}$$

The a steady state error vector then becomes;

$$\mathbf{E}_{ss} = [\mathbf{I} - \mathbf{A}]^{-1} \times [\Delta\mathbf{B}] = \Delta\mathbf{1} \tag{4.13}$$

It is observed from simulations that the dynamic system represented by Equations 4.9 and 4.10 does not exhibit the dip phenomenon and from Equation 4.14, the steady state error vector, $\mathbf{E}_{ss}$ is always $\mathbf{\Delta 1}$ irrespective of the network size and topology, i.e., if $e = \Delta, \ \mathbf{E}_{ss} = \Delta\mathbf{1}, \ \forall i \in N$.

4.3 Evaluation Methodology for Simulations and Practical Experimentation

To evaluate the performance of the suggested synchronization scheme, simulations and practical experiments are carried out on networks of different sizes and configurations. In this section, we give a detailed description of the framework, specifications and parameters employed for the simulations and practical experiments. This is followed by a description of the test network configurations considered for the evaluation.

4.3.1 Description and Specifications of Simulations

To evaluate the performance of all the suggested methods for node time update, simulations are carried out on MathWorks$^{\text{TM}}$ MATALB platform. In each simulation, it is assumed that there are perfect network conditions with no delays, jitters, fading and noise. Each node time is initialized as a random scalar between 0 and 1 , i.e. $t_i(k=0) = rand[0-1]$ and the gateway time is initialized to 0, i.e. $t_g(k=0) = 0$. The incremental time step and ticking rate of the gateway node Δ is taken to be $1ms$. Each node time estimate is carried out iteratively by taking the average of its neighborhood time values [1] . Respective equations representing the time estimate of nodes each network used in the simulations are presented in section 4.3.4. The simulation parameters are given in Table 4.1.

---

[1] Refer sections 3.2.1, 3.3.1 and 3.3.3

Table 4.1: Parameter Specifications for MATALB Simulation

Parameter	Value	Unit
Incremental Time-Step Size, Δ	0.001	sec
Initial slave time, $t_i(k=0)$	rand[0.0-1.0]	sec
Initial master time, $t_g(k=0)$	0	sec

4.3.2 Framework for Practical Experimentation

4.3.2.1 Hardware Platform

In our practical experiments, the platform based on a network comprising MicaZ nodes from manufactured by Memsic$^{\text{TM}}$, instrumented with 7.37MHz 8-bit Atmel Atmega128L microcontrollers are employed. The MicaZ nodes are equipped with 4kB RAM, 128kB program flash and Chipcon CC2420 radio chip [2] which provides a 250 kbps data rate at 2.4 GHz frequency. The 7.37MHz quartz oscillator on the MicaZ board is employed as the clock source for the timer used for timing measurements. This timer operates at 1/8 of that frequency and thus each timer tick occurs at approximately every 921 kHz (approx. 1 μs). TinyOS-2.1.2 [3] installed on Ubuntu Linux Distribution 14 is used as the base operating system for all experimental work. The CC2420 transceiver on the MicaZ board has the capability of timestamping synchronization packets at MAC layer with the timer used for timing measurements. This is a well-known method that increases the quality of time synchronization by reducing the effect of non-deterministic error sources arising from communication [41, 32]. Packet level time synchronization interfaces

---

[2] http://www.ti.com/product/CC2420

[3] http://www.tinyos.net/

provided by TinyOS are utilized to timestamp synchronization messages at MAC layer [58, 59]. The TMilli timer [60] is used for all timing and timestamping operations in our implementation source codes.

4.3.2.2 Testbed Setup

In each practical experiment, the testbed is composed of Memsic^{TM} MicaZ sensor nodes are placed in the communication range of a reference broadcaster. The reference node periodically broadcasts query packets which are received approximately at the same time by all nodes. Each sensor node then transmits a reply packet carrying its logical clock value at the reception time of the query packet to its neighboring nodes and the base station node. The base station node is connected to a PC collects these reply packets and forwards them through the serial port for logging. At the end of each experiment, we analyze the logged experimental data and plot error profiles using MATLAB. The realization of specific network topologies is done by forcing nodes to accept only the packets from their neighboring nodes.

4.4 Test Networks

The proposed methods of local time update are tested using networks of varying sizes and topologies. The sizes considered are 4, 9 and 16 node networks for grid, hexagonal and random topologies. The mathematical representation for the asynchronous average consensus [4] used for the time updates for each network is presented.

4.4.1 4-Node Network

The first network considered is a small network of 1 gateway node and 3 ordinary nodes.

4.4.1.1 Grid Topology

The distribution of the nodes in this topology is shown in Figure 4.1. The mathematical representation of this configuration is also discussed. From Figure 4.1, node 1 is connected to node 2 and gateway node, therefore the time average value for node 1 is as given by Equation 4.14.

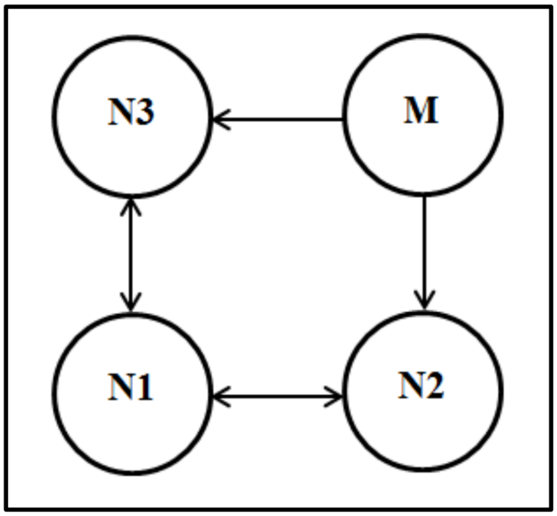


Figure 4.1: 4-Node Grid Topology

---

[4] Described in section 3.3.3, where the system becomes synchronous as represented by Equation 3.3 if $J_{ik} = 1,\ \forall i \in N$, and therefore the presented asynchronous equations for each network represents a more general form of all the update methods. Also, for the sake of generality, the added parameter, e of the first contribution is excluded from all equations.

$$t_1(k) = J_{1k}\ \frac{t_2(k-1) + t_3(k-1)}{2} \tag{4.14}$$

Similarly the time average values of nodes 2 and 3 are derived based on their connections and given respectively by Equations 4.15 and 4.16.

$$t_2(k) = J_{2k}\ \frac{t_1(k-1) + \Delta k}{2} \tag{4.15}$$

$$t_3(k) = J_{3k}\ \frac{t_1(k-1) + \Delta k}{2} \tag{4.16}$$

The equations representing each node time average can be written in a a general state equation given by Equation 4.17.

$$\begin{bmatrix} t_1(k) \\ t_2(k) \\ t_3(k) \end{bmatrix} = \begin{bmatrix} J_{1k} & 0 & 0 \\ 0 & J_{2k} & 0 \\ 0 & 0 & J_{3k} \end{bmatrix} \left\{ \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} t_1(k-1) \\ t_2(k-1) \\ t_3(k-1) \end{bmatrix} + \Delta k \begin{bmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} \right\} \tag{4.17}$$

The first part of the right hand side of the equation depends on the connectivity between a node and its neighbors whereas the second part depends on the connectivity between a node and the gateway node given that a link exist between them. This is all multiplied by the activation matrix which determines the nodes that wake up at time instant, k.

4.4.1.2 Random Topology

In this configuration, all nodes are interconnected as shown in Figure 4.2. Node 1 receives its time updates from nodes 2 and 3 plus gateway node. Node 2 receives from 1, 3 and gateway node and node 3 receives from nodes 1, 2 and gateway node. The respective time values for nodes 1, 2 and 3 are given by Equations 4.18, 4.19 and 4.20. Equation 4.21 represents the combined time update state equation. This distribution of nodes becomes the same after the first iteration since all nodes are interconnected.

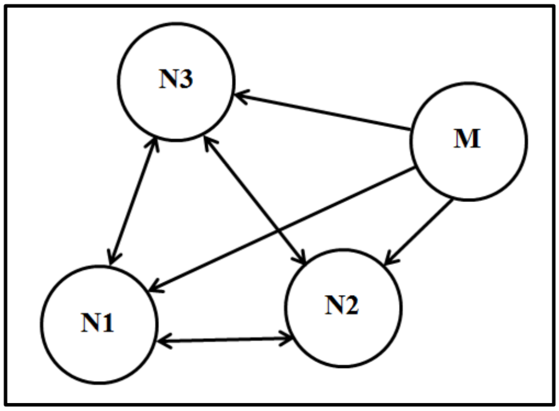


Figure 4.2: 4-Node Random Topology

$$t_1(k) = J_{1k}\ \frac{\Delta k + t_2(k-1) + t_3(k-1)}{3} \tag{4.18}$$

$$t_2(k) = J_{2k}\ \frac{\Delta k + t_2(k-1) + t_3(k-1)}{3} \tag{4.19}$$

$$t_3(k) = J_{3k}\ \frac{\Delta k + t_2(k-1) + t_3(k-1)}{3} \tag{4.20}$$

$$\begin{bmatrix} t_1(k) \\ t_2(k) \\ t_3(k) \end{bmatrix} = \begin{bmatrix} J_{1k} & 0 & 0 \\ 0 & J_{2k} & 0 \\ 0 & 0 & J_{3k} \end{bmatrix} \left\{ \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & 0 \end{bmatrix} \begin{bmatrix} t_1(k-1) \\ t_2(k-1) \\ t_3(k-1) \end{bmatrix} + \Delta k \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} \right\} \tag{4.21}$$

4.4.1.3 Hexagonal Topology

In this configuration, all nodes are interconnected as shown in Figure 4.3. Node 1 receives its time updates from nodes 2 and 3. Node 2 receives from 1, 3 and gateway and node 3 receives from nodes 1, 2 and gateway node. The respective time values for nodes 1, 2 and 3 are given by Equations 4.22, 4.23 and 4.24. Equation 4.25 represents the combined time update state equation.

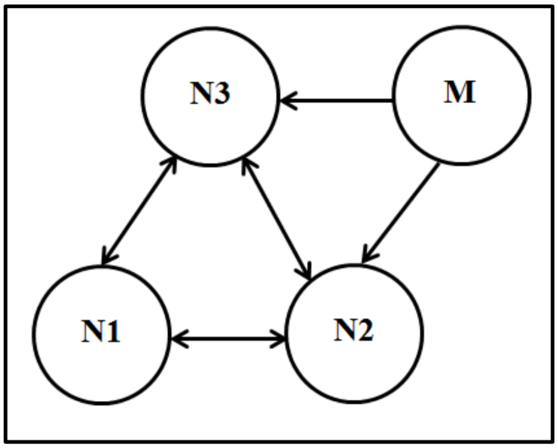


Figure 4.3: 4-Node Hexagonal Topology

$$t_1(k) = J_{1k}\,\frac{t_2(k-1) + t_3(k-1)}{2} \tag{4.22}$$

$$t_2(k) = J_{2k}\,\frac{\Delta k + t_1(k-1) + t_3(k-1)}{3} \tag{4.23}$$

$$t_3(k) = J_{3k}\,\frac{\Delta k + t_1(k-1) + t_2(k-1)}{3} \tag{4.24}$$

$$\begin{bmatrix} t_1(k) \\ t_2(k) \\ t_3(k) \end{bmatrix} = \begin{bmatrix} J_{1k} & 0 & 0 \\ 0 & J_{2k} & 0 \\ 0 & 0 & J_{3k} \end{bmatrix} \left\{ \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & 0 \end{bmatrix} \begin{bmatrix} t_1(k-1) \\ t_2(k-1) \\ t_3(k-1) \end{bmatrix} + \Delta k \begin{bmatrix} 0 \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} \right\} \tag{4.25}$$

4.4.2 9-Node Network

The mathematical representation and node distribution of a 9 node network for grid, random and hexagonal topologies are presented. The state equation for all three configurations derived using the asynchronous average consensus algorithm is given in Equation 4.26. All networks follow the same state equation with differences only in the entries of the matrices **A** and **B**.

$$[\mathbf{T}_k]^{8\times1} = [\mathbf{J}_k]^{8\times8}\left\{[\mathbf{A}]^{8\times8}[\mathbf{T}_{k-1}]^{8\times1} + \Delta k[\mathbf{B}]^{8\times1}\right\} \tag{4.26}$$

4.4.2.1 Grid Topology

In this topology, 8 slave nodes and 1 master node are connected in a grid topology as shown in Figure 4.4. The connectivity matrices **A** and **B** are given respectively as;

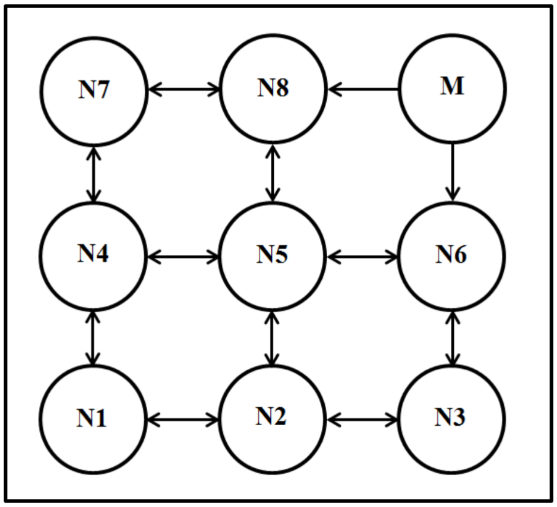


Figure 4.4: 9-Node Grid Topology

$$\mathbf{A} = \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{3} \\ 0 \\ \frac{1}{3} \end{bmatrix}$$

4.4.2.2 Random Topology

Here, 8 slave nodes and 1 master node are connected in a random configuration as shown in Figure 4.5.The connectivity matrices **A** and **B** are given respectively as;

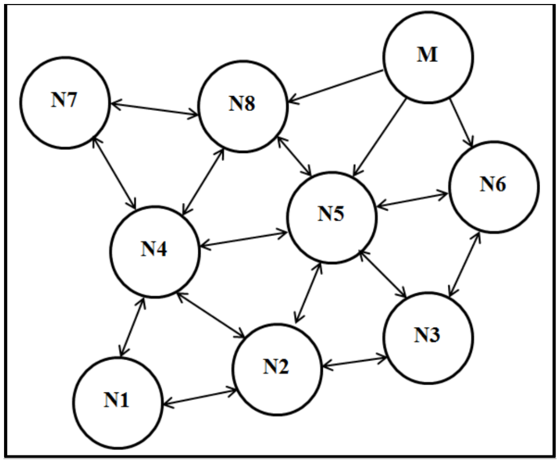


Figure 4.5: 9-Node Random Topology

$$
\mathbf{A} = \begin{bmatrix}
0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\
\frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 \\
0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\
\frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} \\
0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & 0 & \frac{1}{6} \\
0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0
\end{bmatrix}
\quad
\mathbf{B} = \begin{bmatrix}
0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{6} \\ \frac{1}{3} \\ 0 \\ \frac{1}{4}
\end{bmatrix}
$$

4.4.2.3 Hexagonal Topology

In this topology, 8 slave nodes and 1 master node are connected in a hexagonal configuration as shown in Figure 4.6. The connectivity matrices **A** and **B** are given respectively as;

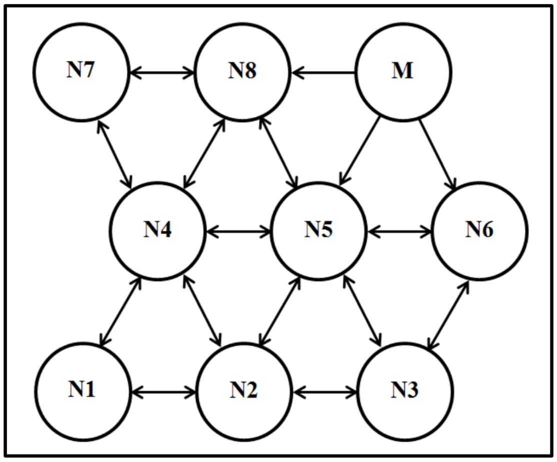


Figure 4.6: 9-Node Hexagonal Topology

$$\mathbf{A} = \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} \\ 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & 0 & \frac{1}{6} \\ 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{6} \\ \frac{1}{3} \\ 0 \\ \frac{1}{4} \end{bmatrix}$$

4.4.3 16-Node Network

Similar to the 9 node network, the mathematical representation and node distribution of the 16 node network for grid, random and hexagonal topologies are presented in this section. The state equation for all three configurations derived using the asynchronous average consensus algorithm is given in Equation 4.27. All networks follow the same state equations with differences only in the entries of the matrices **A** and **B**.

$$[\mathbf{T}_k]^{15\times1} = [\mathbf{J}_k]^{15\times15}\left\{[\mathbf{A}]^{15\times}[\mathbf{T}_{k-1}]^{15\times1} + \Delta k[\mathbf{B}]^{15\times1}\right\} \qquad (4.27)$$

4.4.3.1 Grid Topology

In this topology, 15 slave nodes and 1 master node are connected in a grid topology as shown in Figure 4.7. The connectivity matrices **A** and **B** are given respectively as;

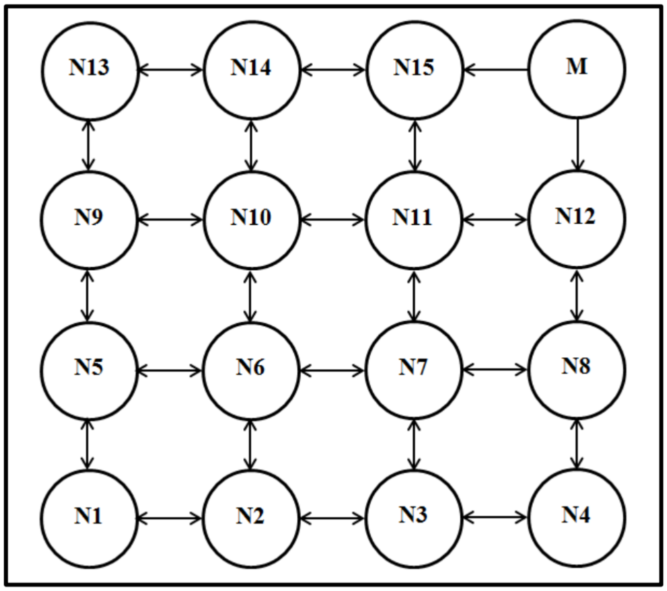


Figure 4.7: 16-Node Grid Topology

$$\mathbf{A} = \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{3} \\ 0 \\ 0 \\ \frac{1}{3} \end{bmatrix}$$

4.4.3.2 Random Topology

In this topology, 15 slave nodes and 1 master node are connected in a random topology as shown in Figure 4.8. The connectivity matrices **A** and **B** are given respectively as;

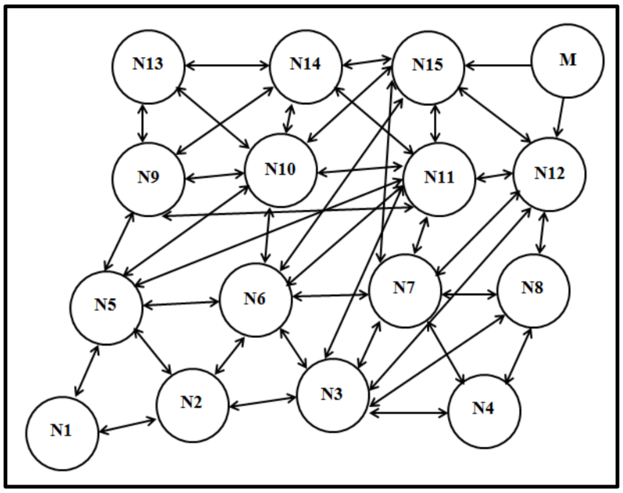


Figure 4.8: 16-Node Random Topology

$$\mathbf{A} = \begin{bmatrix}
0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{7} & 0 & \frac{1}{7} & 0 & \frac{1}{7} & \frac{1}{7} & \frac{1}{7} & 0 & 0 & \frac{1}{7} & \frac{1}{7} & 0 & 0 & 0 \\
0 & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{6} & \frac{1}{6} & 0 & 0 & 0 & \frac{1}{6} & 0 & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & 0 & 0 & 0 \\
0 & \frac{1}{7} & \frac{1}{7} & 0 & \frac{1}{7} & 0 & \frac{1}{7} & 0 & 0 & \frac{1}{7} & \frac{1}{7} & 0 & 0 & 0 & \frac{1}{7} \\
0 & 0 & \frac{1}{7} & \frac{1}{7} & 0 & \frac{1}{7} & 0 & \frac{1}{7} & 0 & 0 & \frac{1}{7} & \frac{1}{7} & 0 & 0 & 0 \\
0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{5} & 0 & 0 & 0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & 0 \\
0 & 0 & 0 & 0 & \frac{1}{7} & \frac{1}{7} & 0 & 0 & \frac{1}{7} & 0 & \frac{1}{7} & \frac{1}{7} & 0 & \frac{1}{7} & \frac{1}{7} \\
0 & 0 & \frac{1}{9} & 0 & \frac{1}{9} & \frac{1}{9} & \frac{1}{9} & 0 & \frac{1}{9} & \frac{1}{9} & 0 & \frac{1}{9} & 0 & \frac{1}{9} & \frac{1}{9} \\
0 & 0 & \frac{1}{6} & 0 & 0 & 0 & \frac{1}{6} & \frac{1}{6} & 0 & 0 & \frac{1}{6} & 0 & 0 & 0 & \frac{1}{6} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{10} & 0 & 0 & 0 & \frac{1}{3} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & 0 & \frac{1}{3} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{7} & \frac{1}{7} & 0 & 0 & \frac{1}{7} & \frac{1}{7} & \frac{1}{7} & 0 & \frac{1}{7} & 0
\end{bmatrix}
\quad
\mathbf{B} = \begin{bmatrix}
0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{6} \\ 0 \\ 0 \\ \frac{1}{7}
\end{bmatrix}$$

4.4.3.3 Hexagonal Topology

In this topology, 15 slave nodes and 1 master node are connected in a hexagonal topology as shown in Figure 4.9. The connectivity matrices **A** and **B** are given respectively as;

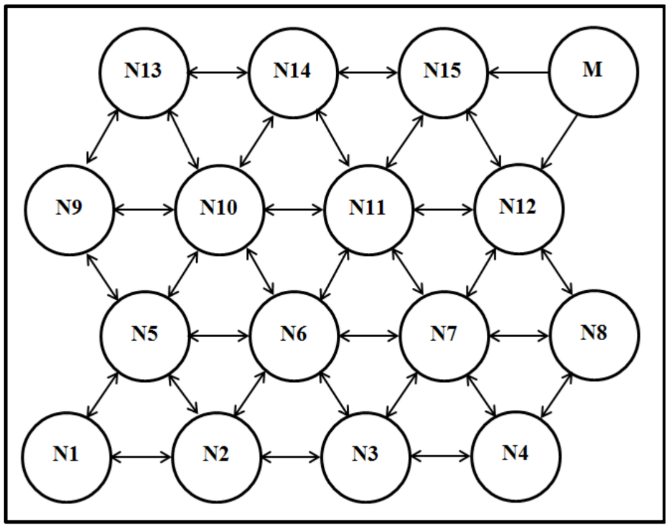


Figure 4.9: 16-Node Grid Topology

$$\mathbf{A} = \begin{bmatrix}
0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & \frac{1}{5} & 0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & 0 & \frac{1}{6} & 0 & 0 & \frac{1}{6} & \frac{1}{6} & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & 0 & \frac{1}{6} & 0 & 0 & \frac{1}{6} & \frac{1}{6} & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{6} & \frac{1}{6} & 0 & 0 & \frac{1}{6} & 0 & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{6} & \frac{1}{6} & 0 & 0 & \frac{1}{6} & 0 & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} & 0 & 0 & 0 & \frac{1}{5} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0
\end{bmatrix}
\quad
\mathbf{B} = \begin{bmatrix}
0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{5} \\ 0 \\ 0 \\ \frac{1}{4}
\end{bmatrix}$$

4.4.4 Synchronous Time Update (STU): Implementation Algorithm

In order to implement the system described in section 4.2, we present a fully distributed method for local time update using synchronous average consensus named Synchronous Time Update (STU) where each node computes it's time estimate in a synchronous manner. In this method all nodes are made to update their estimates within the same specified period and hence the state of network nodes can be represented by Equation 3.3. The pseudo-code of Synchronous Time Update is given by **Algorithm 2**.

Algorithm 2: STU Protocol Pseudo-Code for Node i

```
Initialization
    $t_i \longleftarrow t_i(0)$; $clockSum \longleftarrow 0$; $totalReceived \longleftarrow 0$;

If $< t_j >$ is received
$clockSum \longleftarrow clockSum + (t_j)$;
$totalReceived \longleftarrow totalReceived + 1$;
$t_{av} \longleftarrow clockSum/totalReceived + e$;


Upon every $\Delta$ seconds
If $totalReceived > 1$ and $IncomingID = Nodej$
$t_i \longleftarrow t_{av}$

$clockSum \longleftarrow 0$; $totalReceived \longleftarrow 0$;
broadcast $< t_i >$
```

Initially when the node is powered on, two variables, *clockSum* and *numReceived* required to calculate the average synchronization time estimate to the neighboring nodes are initialized to zero. Whenever a synchronization message from any neighboring node is received, the time estimate t_j of a neighboring node j at the time of reception, is saved. The received time, t_j is then added to the *clockSum* variable and the number of received clock values, *totalReceived* is incremented. To exchange time synchronization packets with its neighboring nodes, node i transmits a broadcast packet of its current time information approximately every Δ seconds. In every Δ seconds, and when the number of received clock values is more than one, the time variable of node i, t_i is updated by setting it to the average estimate, t_{av}. Finally, the time estimate of node i, t_i is transmitted and the $clockSum$ variable and the number of received clock values, $totalReceived$ are initialized.

☐ **Remark 4.2**: In this method, knowledge of network nodes and connectivity is needed and node initialization or wakeup for all network nodes has to be done within a short period of time for all nodes. This method is an ideal conception of the protocol and cannot be deployed in the practical sense since it requires a synchronized network and cannot be used for synchronization but is presented here for comparison with the developed asynchronous schemes.

4.5 Accuracy-Enhanced Method for Time Synchronization: Results

In this section, we assess how variations in the added parameter, e affects the minimum error in the dip region and the number of communication cycles or iterations. This is done by plotting the error profiles of node time error with e varied from 0.1Δ to 0.9Δ and comparing the minimum error in the dip region and the number of iterations needed to reach that error. The assessment is done using both simulations and practical experiments and is carried out on grid node networks of 4, 9 and 16 nodes. For the sake of clarity, the error profiles for nodes closest to the gateway node are plotted for each test [5].

---

[5] Nodes closest to gateway node exhibit the most accurate synchronization [52]

4.5.1 Simulation Results: Error Profiles for Varying Added Parameter

Simulations are carried out using the specifications stated in section 4.3.1 the simulations are carried out based on the equations in section 4.4.

4.5.1.1 Simulation Results for 4 Node Network

Figure 4.10 shows the simulation error profiles for node 2 [6], where the parameter e is varied from 0.1Δ to 0.9Δ.

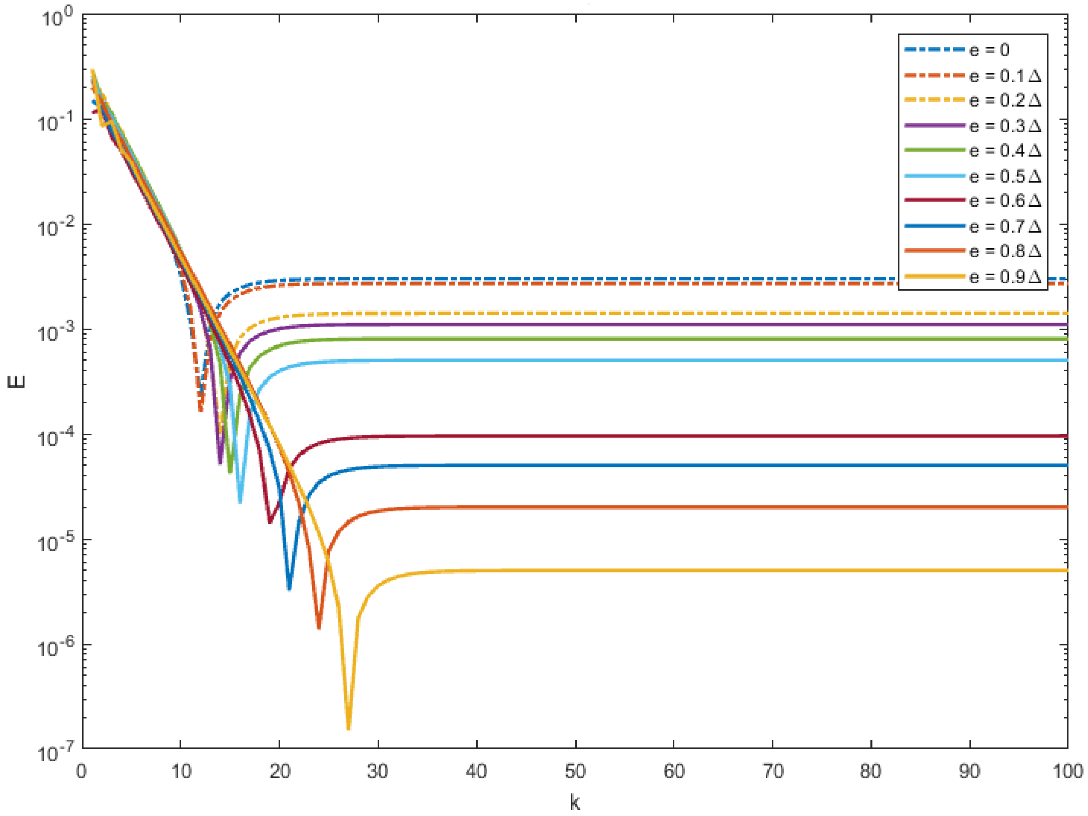


Figure 4.10: Simulation Error Profiles for Varying Added Parameter for 4 Node Grid Topology

---

[6] Refer Figure 4.1, section 4.4.4.1

It can be observed that, the synchronization error in the dip region decreases consistently as e is increased from 0.1Δ to 0.9Δ which is due to the reduction in the V_k term in Equation 4.5. It is also observed there is a corresponding increase in the number of communications needed to reach the dip region as e is increased from 0.1Δ to 0.9Δ. For instance the dip region error and number of communications for $e = 0$ are respectively 13 and 4×10^{-4} where as that at $e = 0.5\Delta$ are 19 and 3×10^{-5} and if $e = 0.9\Delta$ are 27 and 2×10^{-7} as summarized in Table 4.2. Hence if $e = 0.9\Delta$ compared to if e is null[7], about two times communications are required at $e = 0.9\Delta$ but a reduction in error by approximately a factor of about 10^3.

Table 4.2: Simulation Results Summary of Number Iterations and Error Values

Dip Region Results for 4 Node Grid Network										
e	**0**	**0.1Δ**	**0.2Δ**	**0.3Δ**	**0.4Δ**	**0.5Δ**	**0.6Δ**	**0.7Δ**	**0.8Δ**	**0.9Δ**
Iterations	13	15	17	17	18	19	20	21	24	27
Error Value	4.E-04	3.E-04	1.E-04	7.E-05	6.E-05	3.E-05	2.E-05	5.E-06	2.E-06	2.E-07

4.5.1.2 Simulation Results for 9 Node Network

To further probe the behavior of the system with the inclusion of e, we carried out simulations on a 9 node grid network. Here, we show the error profiles for node 8 [8] as shown in Figure 4.11. Here we observe a similar pattern of reduction in dip minimum error with a corresponding increase in the number of communications

---

[7] Which represents the original local update presented in [51, 52]

[8] Closest with node 6 to gateway node, **M**, Refer Figure 4.4, section 4.4.4.1

needed to reach the dip region as e is increased from 0.1Δ to 0.9Δ. Here, if $e = 0$ compared to if $e = 0.9\Delta$, the dip region error and number of communications are respectively 44 and 6×10^{-4} for $e = 0$ whereas that at $e = 0.9\Delta$ are 82 and 4×10^{-7} as summarized in Table 4.3.

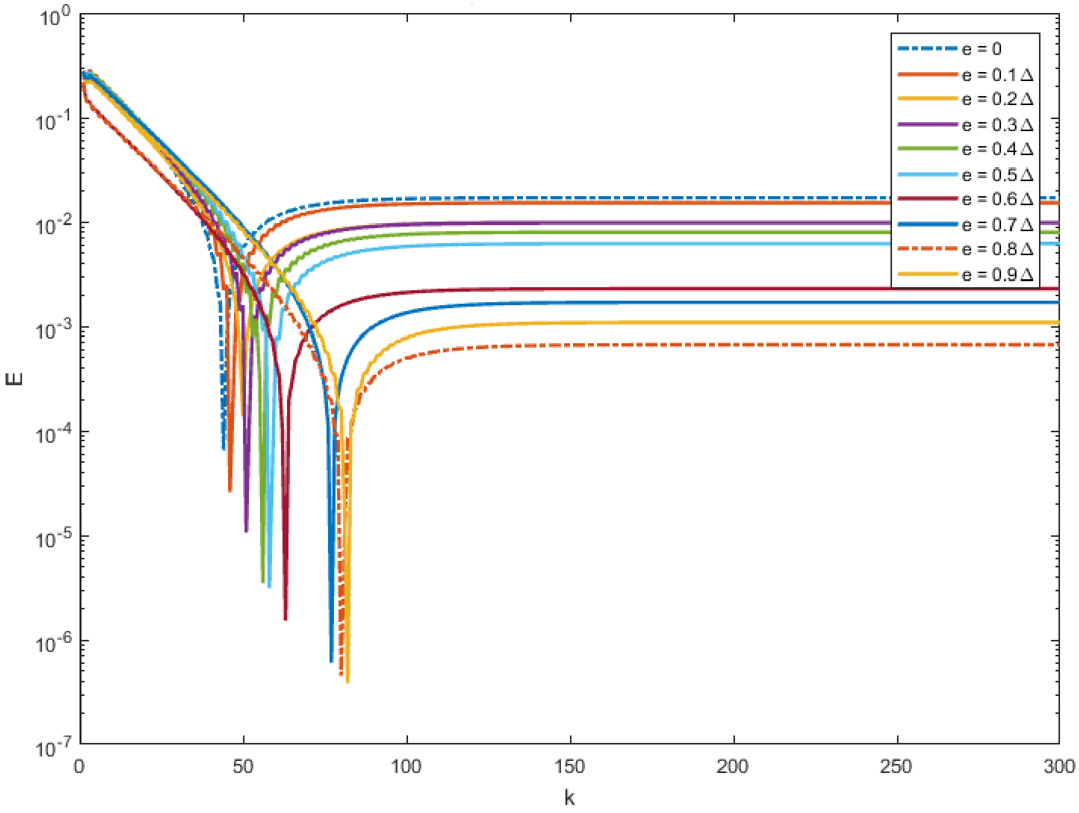


Figure 4.11: Simulation Error Profiles for Varying Added Parameter for 9 Node Grid Topology

Table 4.3: Simulation Results Summary of Number Iterations and Error Values

Dip Region Results for 9 Node Grid Network										
e	**0**	**0.1Δ**	**0.2Δ**	**0.3Δ**	**0.4Δ**	**0.5Δ**	**0.6Δ**	**0.7Δ**	**0.8Δ**	**0.9Δ**
Iterations	44	46	50	51	56	58	63	77	80	82
Error Value	6.E-04	2.E-05	1.E-05	1.E-05	4.E-06	3.E-06	2.E-06	6.E-07	5.E-07	4.E-07

4.5.1.3 Simulation Results for 16 Node Network

On an even larger network of 16 node grid network, we show the error profiles for node 15 [9] as shown in Figure 4.11, we observe a similar pattern of reduction in dip minimum error with a corresponding increase in the number of communications needed to reach the dip region as e is increased from 0.1Δ to 0.9Δ. It is observed that, if $e = 0$ compared to if $e = 0.9\Delta$, the dip region error and number of communications are respectively 76 and 5×10^{-4} where as that at $e = 0.9\Delta$ are 165 and 8×10^{-8} as summarized in Table 4.4.

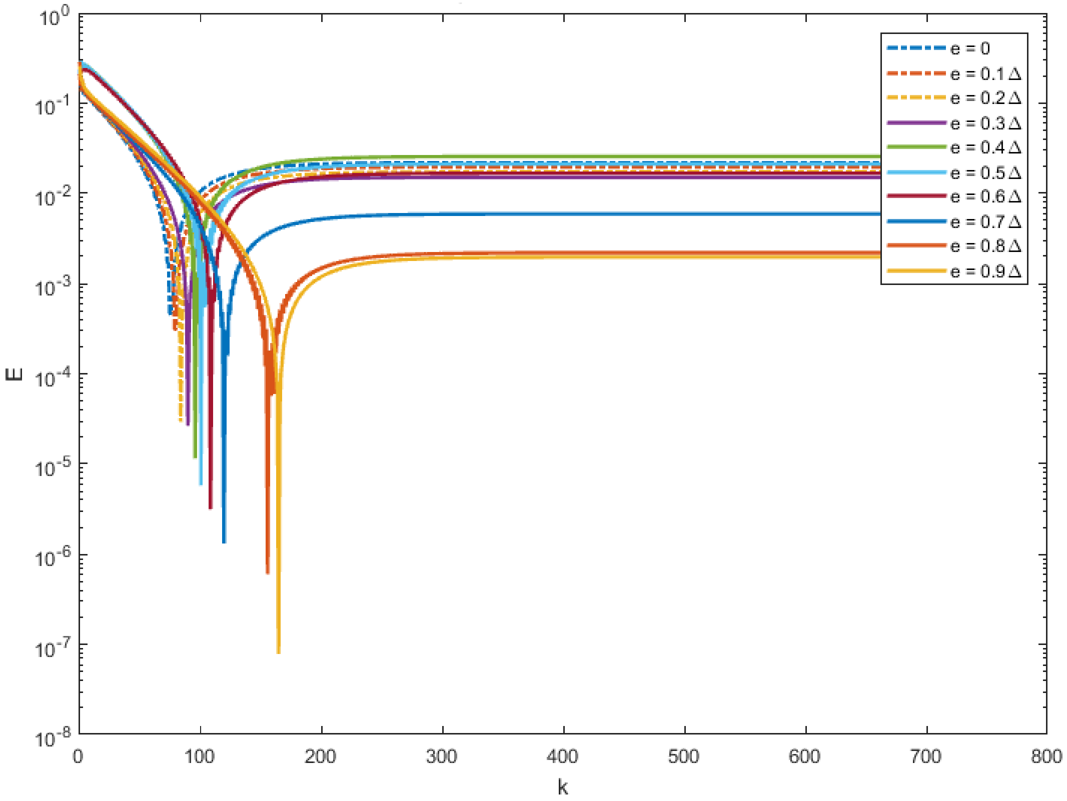


Figure 4.12: Simulation Error Profiles for Varying Added Parameter for 16 Node Grid Topology

---

[9] Closest with node 12 to gateway node, **M**, Refer Figure 4.7, section 4.4.4.1

Table 4.4: Simulation Results Summary of Number Iterations and Error Values

Dip Region Results for 16 Node Grid Network										
e	**0**	**0.1Δ**	**0.2Δ**	**0.3Δ**	**0.4Δ**	**0.5Δ**	**0.6Δ**	**0.7Δ**	**0.8Δ**	**0.9Δ**
Iterations	76	80	84	90	96	101	109	120	156	165
Error Value	5.E-04	3.E-04	3.E-05	3.E-05	1.E-05	6.E-06	3.E-06	1.E-06	6.E-07	8.E-08

4.5.2 Practical Results: Error Profiles for Varying Added Parameter

To further study the behavior of this phenomenon, we conducted practical experiments based on framework presented in section 4.3.2. The implementation algorithm employed for the practical experiments is **Algorithm 2**.

4.5.2.1 Practical Results for 4 Node Network

Figure 4.13 shows the practical error profiles for node 2 [10], where the parameter e is varied from 0.1Δ to 0.9Δ. Similar to the simulation results, we observe the pattern of reduction in dip minimum error with a corresponding increase in the number of communications needed to reach the dip region as e is increased from 0.1Δ to 0.9Δ, although the consistency is not as clear as in the simulation results. It is also observed that, no dip appears for 0.8Δ and 0.9Δ[11]. Here, if $e = 0$ compared to if $e = 0.7\Delta$, the dip region error and number of communications are respectively 17 and 8×10^{-4} for $e = 0$ whereas that at $e = 0.7\Delta$ are 24 and 2×10^{-5} as summarized in Table 4.5.

---

[10] Refer Figure 4.1, section 4.4.4.1

[11] This is expected for a small network, although we see the steady state error approaching Δ as predicted by the the analysis in section 4.2

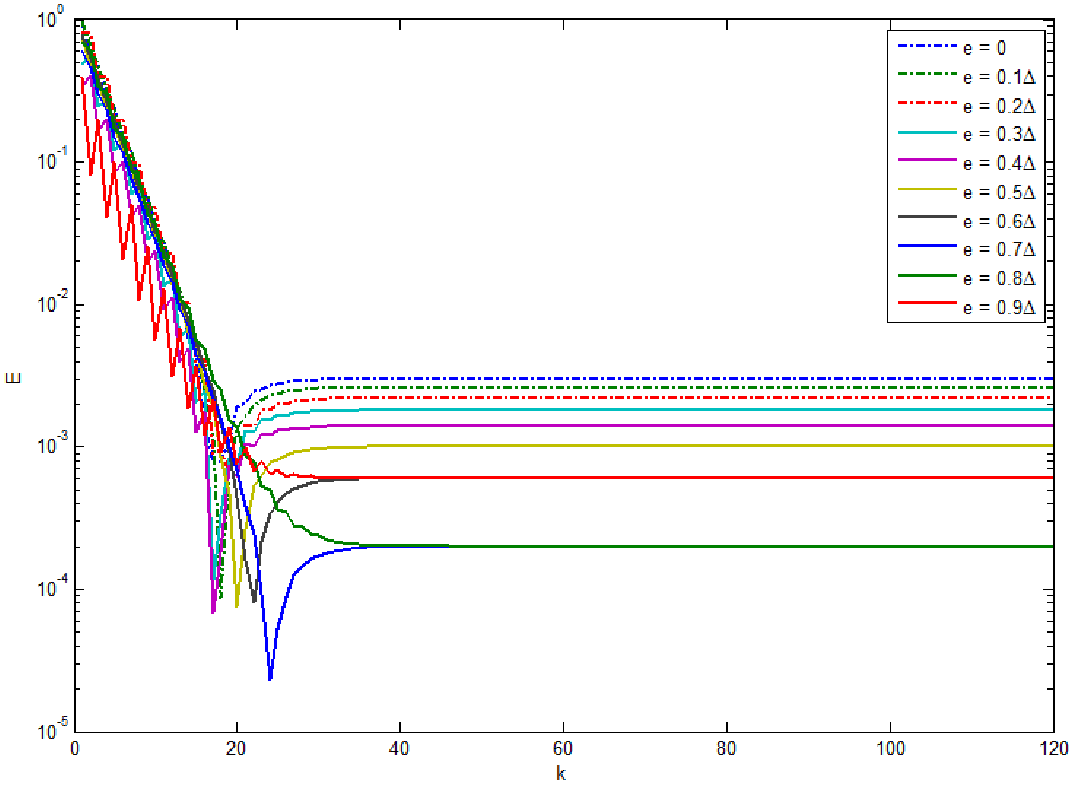


Figure 4.13: Practical Error Profiles for Varying Added Parameter for 4 Node Grid Topology

Table 4.5: Practical Results Summary of Number Iterations and Error Values

Dip Region Results for 4 Node Grid Network										
e	**0**	**0.1Δ**	**0.2Δ**	**0.3Δ**	**0.4Δ**	**0.5Δ**	**0.6Δ**	**0.7Δ**	**0.8Δ**	**0.9Δ**
Iterations	17	18	20	17	17	20	22	24	-	-
Error Value	8.E-04	9.E-05	6.E-04	1.E-04	7.E-05	7.E-05	8.E-05	2.E-05	-	-

4.5.2.2 Practical Results for 9 Node Network

To further probe the behavior of the system with the inclusion of e practically, we carried out experiments on a 9 node grid network. Here, we show the error profiles for node 8 [12] as shown in Figure 4.14. Here we observe a similar pattern

[12] Similar to simulations, node 8 and node 6 are closest to the gateway node, **M**, Refer Figure 4.4, section 4.3.4.1

of reduction in dip minimum error with a corresponding increase in the number of communications needed to reach the dip region as e is increased from 0.1Δ to 0.9Δ. Here, if $e = 0$ compared to if $e = 0.9\Delta$, the dip region error and number of communications are respectively 51 and 9×10^{-4} for $e = 0$ whereas that at $e = 0.9\Delta$ are 97 and 3×10^{-5} as summarized in Table 4.6.

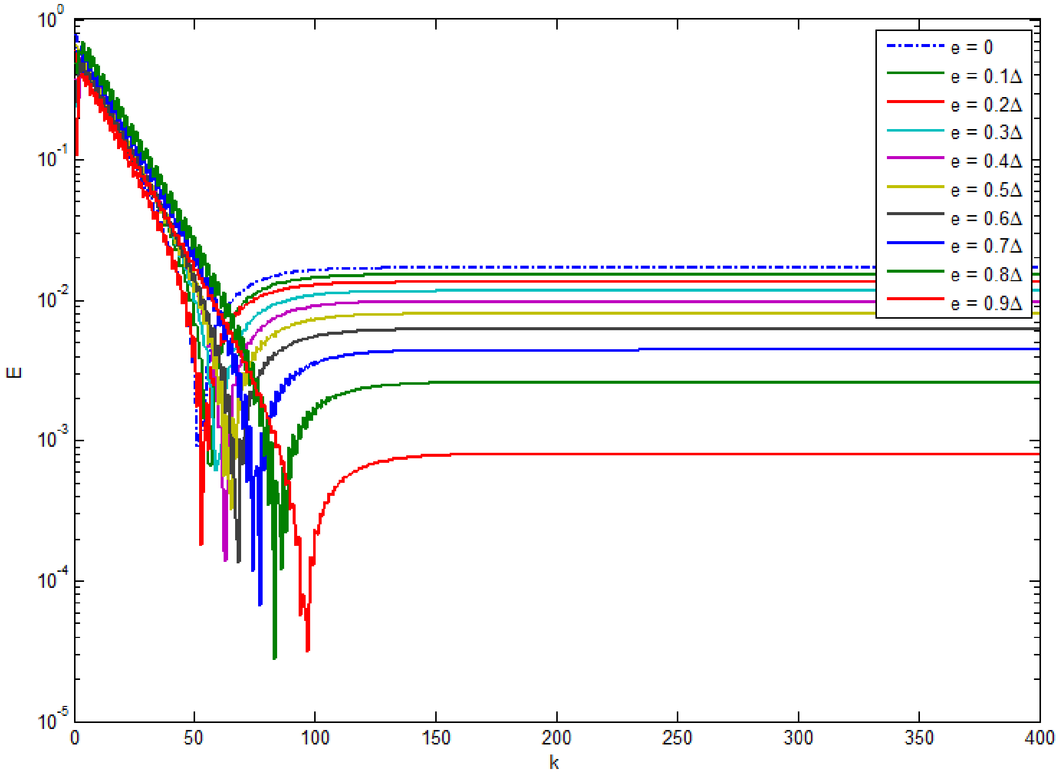


Figure 4.14: Practical Error Profiles for Varying Added Parameter for 9 Node Grid Topology

Table 4.6: Practical Results Summary of Number Iterations and Error Values

Dip Region Results for 9 Node Grid Network										
e	**0**	**0.1Δ**	**0.2Δ**	**0.3Δ**	**0.4Δ**	**0.5Δ**	**0.6Δ**	**0.7Δ**	**0.8Δ**	**0.9Δ**
Iterations	51	57	53	59	63	65	68	77	83	97
Error Value	9.E-04	7.E-04	2.E-04	6.E-04	1.E-04	3.E-04	1.E-04	7.E-05	3.E-05	3.E-05

4.5.2.3 Practical Results for 16 Node Network

On larger network of 16 nodes on a grid topology, we show the error profiles for node 15 [13] as shown in Figure 4.15, we observe a similar pattern of reduction in the dip minimum error with a corresponding increase in the number of communications needed to reach the dip region as e is increased from 0.1Δ to 0.9Δ. It is observed that, if $e = 0$ compared to if $e = 0.9\Delta$, the dip region error and number of communications are respectively 102 and 5×10^{-4} where as that at $e = 0.9\Delta$ are 190 and 3×10^{-5} as summarized in Table 4.7.

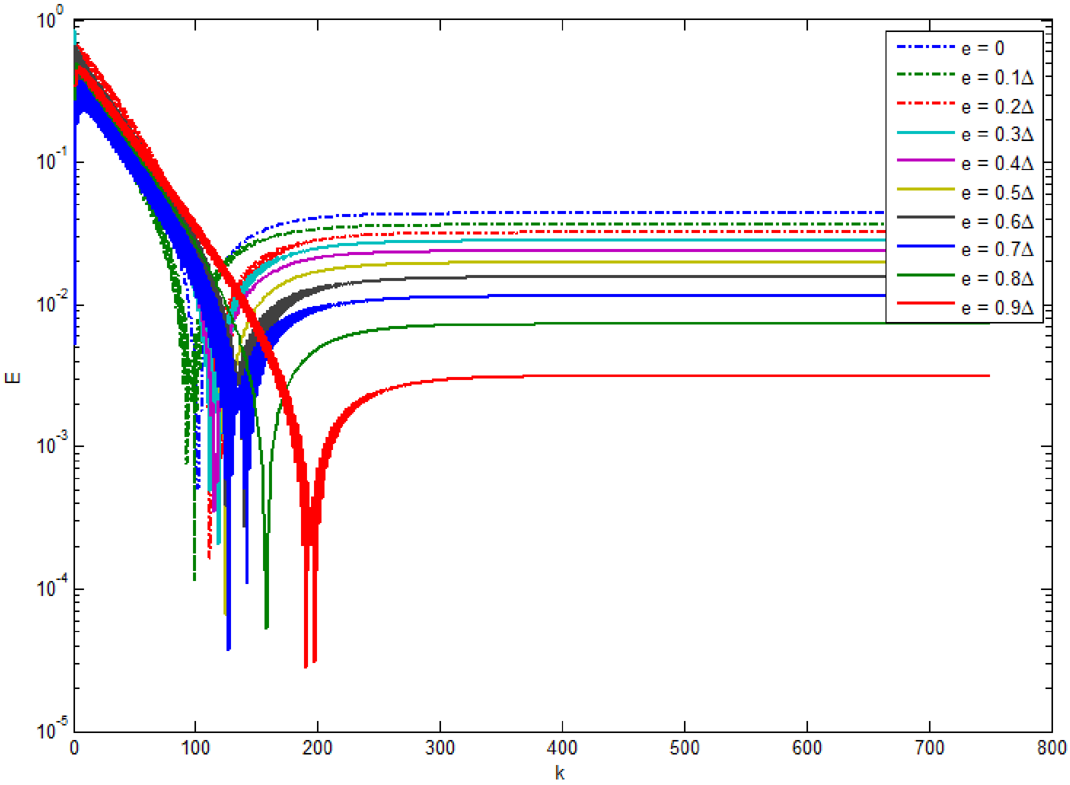


Figure 4.15: Practical Error Profiles for Varying Added Parameter for 16 Node Grid Topology

---

[13] Node 15 is closest with node 12 to gateway node, **M**, Refer Figure 4.7, section 4.4.4.1

Table 4.7: Practical Results Summary of Number Iterations and Error Values

Dip Region Results for 16 Node Grid Network										
e	**0**	**0.1Δ**	**0.2Δ**	**0.3Δ**	**0.4Δ**	**0.5Δ**	**0.6Δ**	**0.7Δ**	**0.8Δ**	**0.9Δ**
Iterations	102	99	112	119	115	125	140	127	158	190
Error Value	5.E-04	1.E-04	2.E-04	2.E-04	3.E-04	7.E-05	3.E-04	4.E-05	5.E-05	3.E-05

4.5.3 Relationship Between Parameter, *e* and Minimum Error in Dip Region

Figures 4.16 and 4.17 shows the graphical relationship between the parameter, *e* and the minimum dip synchronization error for simulations and practical experiments on 4, 9 and 16 node grid networks respectively.

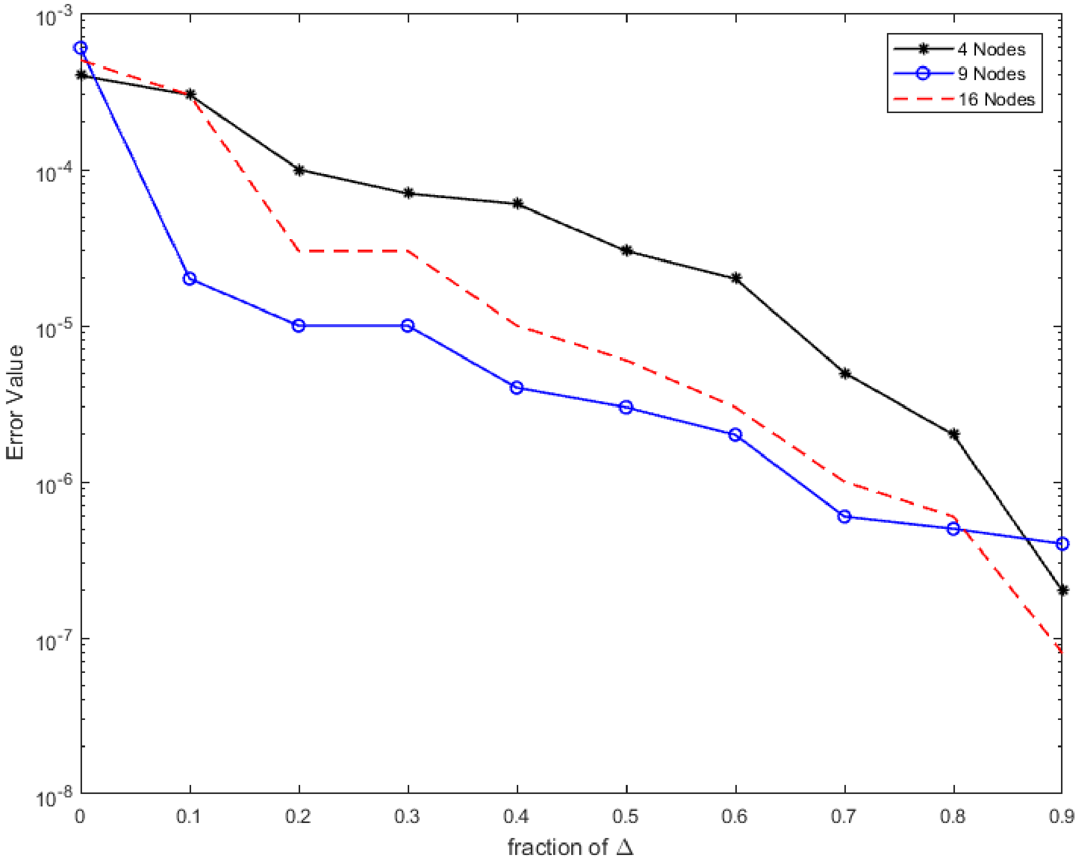


Figure 4.16: Relationship Between Added Parameter,*e* and Minimum Error in Dip Region for Simulations of 4, 9 and 16 Node Networks

Clearly, as e increases, the minimum error decreases and hence we can infer that, there exist a negative correlation between the added parameter, e and the minimum error in dip region for all the considered networks sizes for both simulations and practical experimental results. As expected, we see a more defined relationship in the simulation results as compared to the practical results.

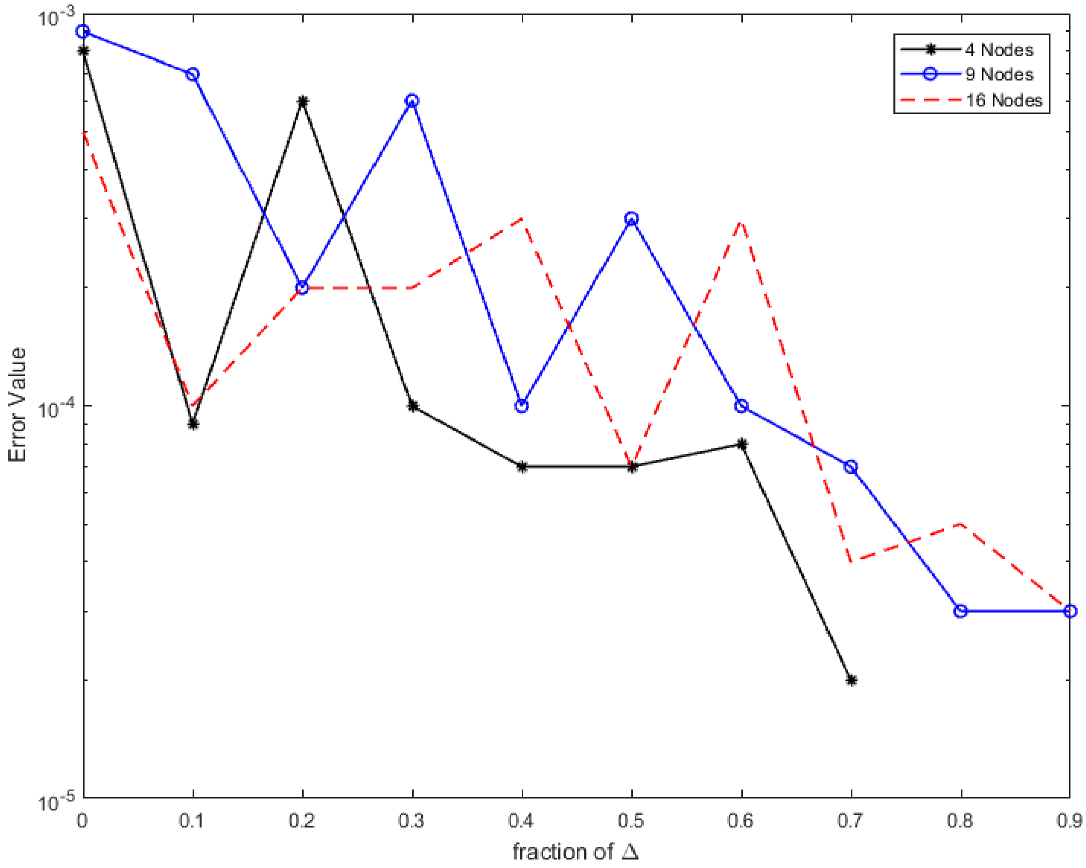


Figure 4.17: Relationship Between Added Parameter,e and Minimum Error in Dip Region for Experiments on 4, 9 and 16 Node Networks

4.6 Summary

In this chapter, we presented detailed analysis and discussions on the first contribution of this thesis where we assessed how variations in the added parameter, e

reduces further the minimum error in the dip region at the expense of increasing the number of communication cycles or iterations. The effect of the parameter was shown by plotting the error profiles of node time error with e varied from 0.1Δ to 0.9Δ and comparing the minimum errors in the dip region and the number of iterations needed to reach those errors. the method is tested through simulations and practical experimentation. The method employed for simulations and practical experiments was first presented then the results obtained from simulations and experiments were discussed.

We compared the extreme case of lowest synchronization error when $e = 0.9\Delta$ to the original method when e is null. I was observed that, on average, about two times the number of communications is required at $e = 0.9\Delta$ with a corresponding reduction in error by approximately a factor of 5×10^{-4} as compared to the original case when $e = 0$ in simulations for 4, 9 and 16 grid networks. For the practical results, it was also observed that, on average, about twice the number of communications is required at $e = 0.9\Delta$ with a corresponding reduction in error by a factor of approximately 5×10^{-2} as compared to when $e = 0$, although the results for the 4 node network deviated slightly from this pattern. Hence the variable, e, representing a fraction of the ticking rate of the gateway node, Δ, gives a trade-off between the dip synchronization error and the number of communication cycles required to reach that error. This presents a flexibility in the protocol design because some applications of wireless sensor networks like distributed beamforming, object tracking and temporal order delivery require a high level of synchronization accuracy [11] whereas other applications like medical, weather and marine sensing/monitoring require minimal number of communications for energy conservation [61, 21].

Despite, the consistency shown in the results presented in this chapter, the local node time update was carried out using the synchronous average consensus algorithm which is not practical for real time wireless sensor networks. In Chapter 5, we take away the need for the synchronous approach to local time update by extending it to some practical asynchronous methods for local node time update. Detailed analysis and discussions of each method is presented with simulation and practical experimental results.

CHAPTER 5

LOCAL NODE TIME UPDATE USING ASYNCHRONOUS CONSENSUS: METHODS AND RESULTS

5.1 Introduction

In the preceding chapter, the synchronous average consensus approach was used for the local node update in the implementation of the suggested synchronization protocol. However, this method is not practical since all synchronous average consensus algorithms require simultaneous updates of all nodes (synchronous) to operate [55, 7, 62]. In this chapter, we remove this requirement by extending to asynchronous consensus approach for local node time update which does not require synchronous update. [63] and hence is a well suited method for distributed consensus based synchronization algorithms. We the present the design and implementation of three practical methods for the local node time update namely, *Timed Sequential Asynchronous Update, (TSAU)*, *Unidirectional Asynchronous Flooding, (UAF)* and *Bidirectional Asynchronous Update, (BAF)*. In this work, we focus on the simulation and practical experimental performance evaluations of each method.

5.2 Proposed Asynchronous Methods for Local Node Time Update

In this section, a detailed operation of our developed methods of local node time update for the the proposed method of synchronization are presented. The general mathematical framework of the asynchronous time update specific to the proposed protocol was presented in section 3.3.3. The pseudo-code and description of each method aided by relevant diagrams is presented here. It is assumed that for all methods that the master (gateway) node ticks uniformly at a rate Δ, represented by $t_g(k) = \Delta \times k$, and not subject to drifts and hence Δ is constant, i.e. $\Delta(k) = \Delta, \ \forall k$. This assumption is practical since many sensor network architectures consist of a gateway or base-station that has access to a stable and precise clock reference, for example a GPS receiver [59, 64]. In the absence of an external clock source, the hardware clock of a selected sensor node, i.e. the gateway(root) node might serve as a source. As discussed in [51], in our suggested approach to synchronization, the gateway node clock, $t_g(k)$ acts as the source or input to the network during synchronization [1].

5.2.1 Timed Sequential Asynchronous Update (TSAU)

The first method TSAU, network nodes are made to compute time estimates one at a time asynchronously in a sequential manner. The sequence is done using proximity to the gateway node, i.e. the closer a node is to the gateway node, the earlier it updates. This is illustrated by Figure 5.1. The pseudo code of TSAU is given by **Algorithm 3**.

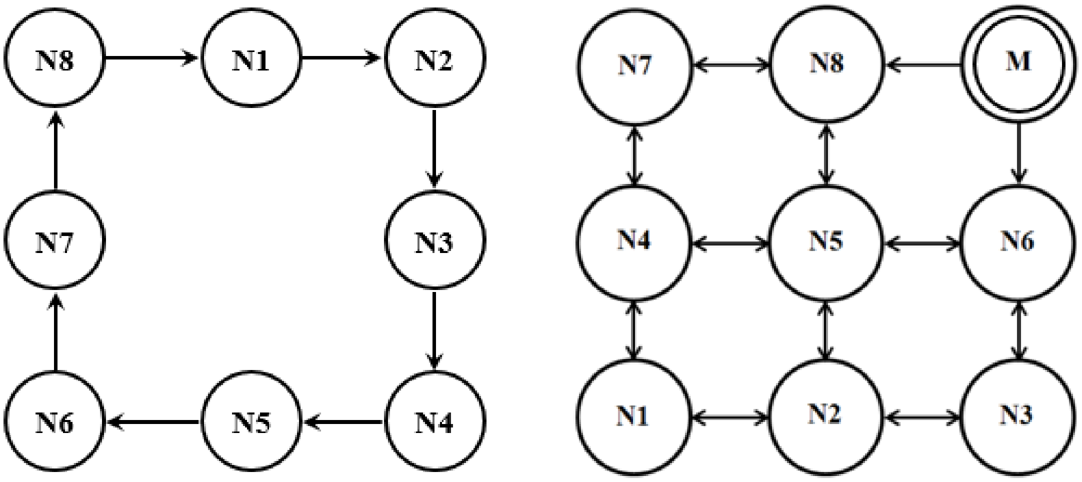
5.2.1.1 Operation Mechanism

All the variables used for STU in **Algorithm 2** are also used here except another variable labeled $UpdateTime$ is introduced. In order for nodes to update in a sequential manner, node IDs are assigned based on proximity to the gateway node, and we program each node to update when the variable $UpdateTime$ is a multiple of Δ, and this variable is calculated by $UpdateTime+ = (N-1) \times \Delta$, where N is the number of network nodes including the gateway node. The $UpdateTime$ for a node with ID, i is initialized to $i \times \Delta$. This allows the activation of the nodes to be carried out in an atomically in a sequential manner.

☐ **Remark 5.1**: It should be noted that this method operates in a completely blind fashion since it requires neither to know the sender node nor stores its time

---

[1] Refer Equations 3.3 and 3.9, and discussions in sections 3.2.1 and 3.3.3



Asynchronous Wake-up Cycle

Neighborhood Layout

Figure 5.1: Asynchronous Wake-up Cycle and Operation Mechanism for TSAU

information. Therefore is expected to show some robustness to changes in network topology. However, for this algorithm to work properly, the network topology has to be maintained throughout its operation [2]. If deployed in a network whose graph is time variant, a wake-up activation sequence which might require different communication packet(s) will be required to control the sequential update of the nodes. In that case, a fixed network topology is not necessary but full knowledge of the network nodes and connectivity would be needed.

5.2.2 Unidirectional Asynchronous Flooding (UAF)

The second method, UAF, is now presented. In this method, nodes update asynchronously based on a wake-up activation protocol regulated by the gateway node. This method improves upon TSAU, whereby it is designed to make nodes at approximately the same proximity to the gateway node update at the same com-

---

[2] Refer to Line 7 of **Algorithm 3**

Algorithm 3: TSAU Pseudo-Code for Node i

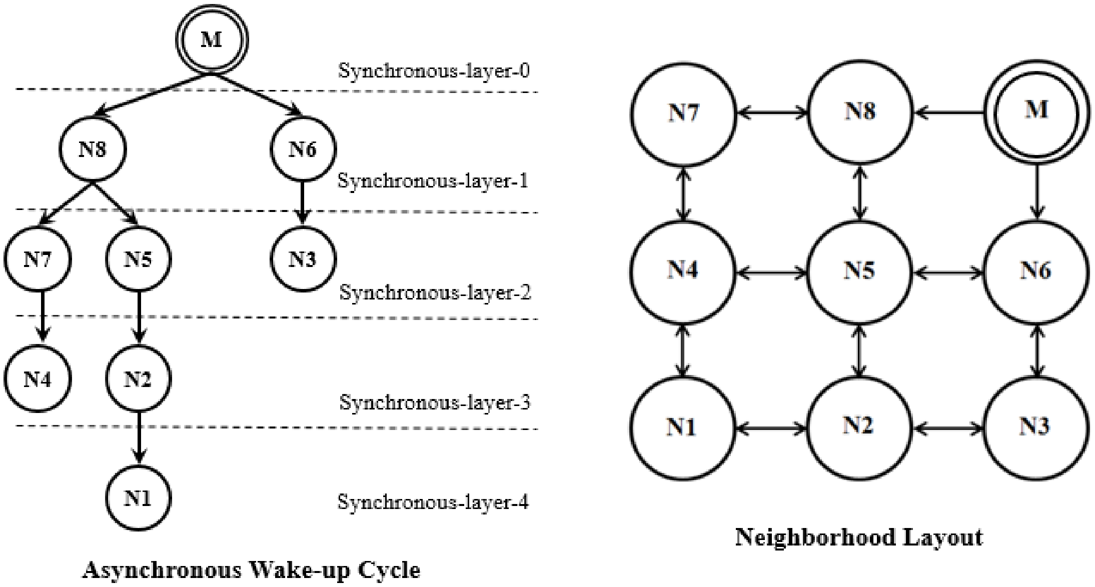
```
Initialization
$t_i \longleftarrow t_i(0)$; $UpdateTime \longleftarrow \mathbf{i} \times \Delta$; $clockSum \longleftarrow 0$; $totalReceived \longleftarrow 0$;
If $< UpdateTime >$ is a multiple $\Delta$
If $< t_j >$ is received
$clockSum \longleftarrow clockSum + (t_j)$;
$totalReceived \longleftarrow totalReceived + 1$;
$t_{av} \longleftarrow clockSum/totalReceived$;
$UpdateTime \longleftarrow UpdateTime + (N - 1) \times \Delta$;

Upon every $\Delta$ seconds
If $totalReceived > 1$ and $IncomingID = Nodej$
$t_i \longleftarrow t_{av}$

$clockSum \longleftarrow 0$; $totalReceived \longleftarrow 0$;
broadcast $< t_i >$
```

munication instant since it is fair to assume that, the transmission and reception times of messages to and from their neighbors and/or to the gateway node are the same. Here, the gateway node is made to regulate the activation of nodes by flooding the network with wake-up messages in every cycle of asynchronous update to begin the cycle and re-initiates the cycle once a current cycle completes and hence the name *Unidirectional*.

5.2.2.1 Operation Mechanism

The description of this asynchronous activation cycle procedure is illustrated by Figure 5.2. As shown in Figure 5.2, layers of connectivity are defined for all network nodes based on their proximity to the gateway node.

All nodes belonging to the same connectivity layer wake up at the same time to carry out the averaging process hence the name *Synchronous Layer*. To elaborate



Asynchronous Wake-up Cycle

Neighborhood Layout

Figure 5.2: Asynchronous Wake-up Cycle and Operation Mechanism for UAF

further on the operation of the method, we outline its stages of operation in the pseudo-code of **Algorithm 4** which is described as follows;

1. Each node, i has a binary status variable labeled as s_i that is set to, $s_i = 0$. Let us assume an upper bound L on connectivity layer, where L depends on the network size and topology. For example, for the network in Figure 5.2, $L = 4$.

2. The gateway node initializes update timer T_s and triggers the update of the nodes connected to it.

3. Once a node i updates, it triggers the update of its nearest neighbor nodes, j whose status bit variable, s_j are a complement of its own, i.e., $s_i = \acute{s}_j$. Once the flooding of the status bits variable begin, if i receives $< t_j, s_j >$ such that, $s_i = \acute{s}_j$ then node i accepts the clock value of j and computes its average, then updates its clock with the computed average and set its s_i to

s_j. If on the other hand, $s_i = s_j$, then it means that nodes i and j belong to the same connectivity layer or the connectivity layer of j lies above that of i therefore it doesn't wake-up to compute and update its clock and hence conserve energy.

4. Each node then broadcasts whatever values of $< t_i, s_i >$ it has every Δ seconds.

5. This process continues until the timer of the gateway node is $Ts > L \times \Delta$. When this event is true, the gateway node initializes its update time T_s and trigger the update of the nearest nodes and hence the whole process begins again.

Algorithm 4: UAF Pseudo-Code for Node i

```
☐ Initialization
$t_i \longleftarrow t_i(0)$; $s_i \longleftarrow 0$; $clockSum \longleftarrow 0$; $totalReceived \longleftarrow 0$;
☐ Upon receiving $< t_j, s_j >$
If $< s_j \neq s_i >$ then
$clockSum \longleftarrow clockSum + (t_j)$
$totalReceived \longleftarrow totalReceived + 1$
$t_{av} \longleftarrow clockSum/totalReceived$
$s_i \longleftarrow s_j$
else if $< s_j = s_i >$ then $t_{av} \longleftarrow t_i$ endif

☐ Upon every $\Delta$ seconds
$t_i \longleftarrow t_{av}$

$clockSum \longleftarrow 0$; $totalReceived \longleftarrow 0$;
broadcast $< t_i, s_i >$
```

☐ **Remark 5.2**: It should be noted here that, the only knowledge needed for this method is a loose upper bound on the connectivity layers at the gateway node and at least one spanning path from any node in the network to the gateway node. Also, no knowledge of size of the network or it's constituents is also needed.

5.2.3 Bidirectional Asynchronous Flooding (BAF)

In this section we present another version of our proposed synchronization scheme which does not require any regulation of the update wake-up sequence by the gateway node and hence can effectively operate in a sparsely distributed random network. In this method, all variables are similar to those in **Algorithm 4** except another variable is introduced to make the network self-regulating in carrying out the update wake-up cycle and hence removes the need for the centralized wake-up regulation seen in UAF. To achieve this, once the update wake-up cycle is initiated by the gateway node, the node(s) in the first and the last connectivity layers automatically carry out the regulation and does not depend on the gateway node for the regulation. Hence we have two realizations of connectivity layers: a Forward Synchronous Layer and a Backward Synchronous Layer, hence the name *Bidirectional*.

5.2.3.1 Operation Mechanism

The pseudo-code is presented in **Algorithm 5** and described as follows;

1. Each node, i has a binary status variable labeled as s_i that is set to, $s_i = 0$ and also has a counter variable c_i that is also initially set to zero

2. The gateway node triggers the update of the nodes by broadcasting its clock

values

3. Once any node receives the gateway clock value (i.e. nodes in the Forward-S Layer-1 as shown in Figure 5.3), the node updates its clock and negates its status bit variable.

4. After doing stage 3, the node triggers the update of its nearest neighboring nodes whose status bit is the complement of its own.

5. When a node i receives a packet from another node j, it compares its s_i with the received s_j and if $s_i \neq s_j$ it means node j belongs to a layer that triggers the update of node i's layer.

6. If the event in 4 is true, then node i accepts the time estimate of j and computes its average, then save its computed average time estimate and set it's s_i to s_j and also set its c_i to $c_i + 1$. If on the other hand, if $s_i = s_j$, then it means that nodes i and j belong to the same connectivity layer or the connectivity layer of j does to trigger the update of i for the current wake-up cycle and therefore it doesn't wake-up to compute and update its clock and hence conserves energy.

7. The process continues until a node finds that its c_i variable is the highest compared to the connected node and their s_i's are the same, then this is the furthest node from the gateway node.

8. This furthest node sets its $c_i = 0$, negate its s_i and trigger a backward flooding.

9. In the backward flooding, all network nodes now have the complement of their initial s_i's , i.e. $\acute{s}_i$ hence the process automatically continues until the fist layer is reached, which in-turn trigger the next forward flooding.

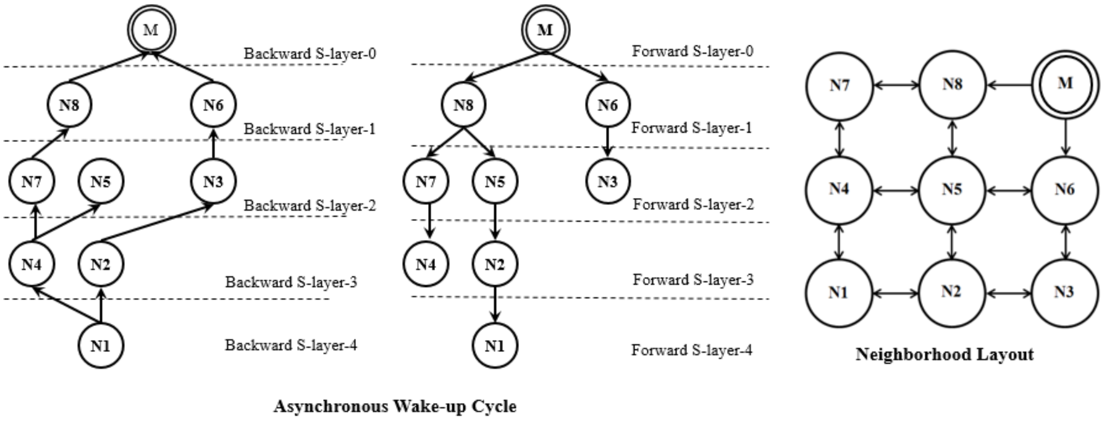


Figure 5.3: Asynchronous Wake-up Cycle and Operation Mechanism for BAF

Algorithm 5: BAF Pseudo-Code for Node *i*

```
☐ Initialization
$t_i \longleftarrow t_i(0)$; $s_i \longleftarrow 0$;$c_i \longleftarrow 0$;$clockSum \longleftarrow 0$; $totalReceived \longleftarrow 0$;
☐Upon receiving $< t_j, s_j, c_j >$
If $< s_j \neq s_i >$ then
$clockSum \longleftarrow clockSum + (t_j)$
$totalReceived \longleftarrow totalReceived + 1$
$t_{av} \longleftarrow clockSum/totalReceived$
$s_i \longleftarrow s_j$
$c_i \longleftarrow c_j + 1$
else if $< s_j = s_i >$ then $t_{av} \longleftarrow t_i$ endif
if $< s_j = s_i >$ and $c_i > c_j, \forall j$ then $c_i \longleftarrow 0$ and $s_i \longleftarrow \bar{s}_i$ endif

☐ Upon every $\Delta$ seconds
$t_i \longleftarrow t_{av}$

$clockSum \longleftarrow 0$; $totalReceived \longleftarrow 0$
broadcast $< t_i, s_i, c_i >$
```

☐ **Remark 5.3**: As observed from the operation of BAF, once the gateway node triggers the initial asynchronous process, the network nodes automatically continues to carry-out the update wake-up cycles whiles making use of the proximity to the gateway node, asynchronicity and energy conservation from controlled and limited computations. This feature make this version of our proposed scheme very robust in that, even if the connectivity layers of nodes are not maintained with time due to node failure and changes in network topology, new layers of connectivity are automatically formed based on the capacity of the nodes in terms of transmission range and the update wake-up cycle continues to be regulated. The only added requirement here is the increased bytes introduced by the added counter variable, c_j.

5.2.4 Performance Metrics for Evaluation

To compare the performance of the developed schemes with the previously developed synchronous method, three parameters are employed. First the minimum error of each node in the dip region is compared, followed by the number of iterations needed to reach that minimum error. Then finally the variance in the number of communication cycles needed to reach the dip region is compared for all network nodes.

5.2.4.1 Minimum Error in Dip Region

The minimum error in the dip region gives a sense of how accurate a protocol is. Hence an efficient method is expected to give very small error values in the dip region. This error is determined by comparing the time values of each node to the gateway node and taking the minimum. For N network nodes, this can be measured by taking the average of the minimum error in the dip region of all the nodes, i.e., $\mathbb{E}_{min}^{dip} = \frac{1}{N}\sum_{i\in V}\min\limits_{i\in V}(t_g(k) - t_i(k))$.

5.2.4.2 Communication Cycles to Reach Dip Region

This parameter is obtained by looking at the number of communication cycles at which the minimum error in the dip region occurs and can be measured by taking the average of this value for all the nodes and is denoted, k_{min}^{dip}. The lower this parameter, the lower the energy needed by a protocol to reach the dip region and hence the more efficient the protocol.

5.2.4.3 Variance in Communication Cycles to Reach Dip Region

For a very efficient protocol, all nodes in the network are expected to reach the dip region using approximately the same number of communication cycles. If this is achieved, the local clock reset and sleep for all node can occur at same time and hence nearly all nodes will synchronize with the gateway node using approximately the same number of pooling cycles. This parameter is calculated by take the variance in the number iterative or communication cycles needed to reach the minimum error in the dip region for all network nodes. This parameter is denoted, $\mathbb{V}_{k_{min}^{dip}}$.

5.3 Simulation Evaluation of Proposed Methods of Local Time Update

In this section we present simulation results for all designed methods of local node time update. Results are presented for 4, 9 and 16 node networks configured in three topologies:grid, random and hexagonal [3]. Simulations are done based on the equations presented in section 4.3.6. In the result of each configuration, we present a graph of the error profile and the node time versus the gateway time.

5.3.1 Simulation Results for Timed Sequential Asynchronous Update (TSAU)

5.3.1.1 Results for 4 Node Network

Figures 5.4, 5.5, 5.6 show the error, $\mathbf{E}_k$ profile[4] plot and node time, $\mathbf{T}_k$ values plot both against the gateway time of the 4 node network for grid, random and hexagonal topologies respectively. We observe from these figures that, the dip phenomenon is exhibited by all network configurations for Timed Sequential Asynchronous Update (TSAU) with all topologies registering, average minimum error in $\mathbb{E}_{min}^{dip}$ in the 10^{-4} range although $\mathbb{E}_{min}^{dip}$, for random where all nodes are interconnected is lowest followed by hexagonal and then the grid network.

This pattern is also shown in the number of communications required to reach the dip region, k_{min}^{dip}, although here the grid topology requires less than the hexag-

---

[3] Refer section 4.3.4

[4] Refer Equation 3.4, section 3.2.1

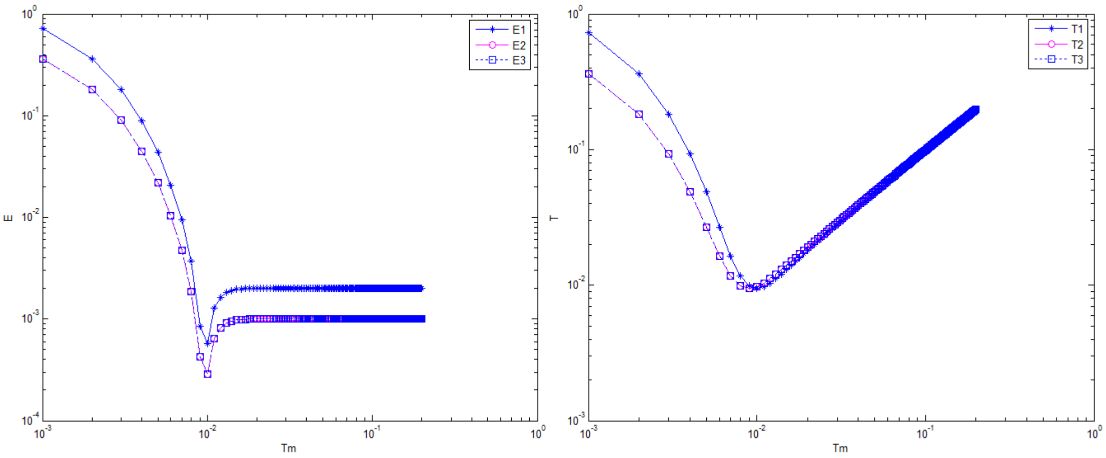


Figure 5.4: Node Time and Error Profiles for Grid Topology

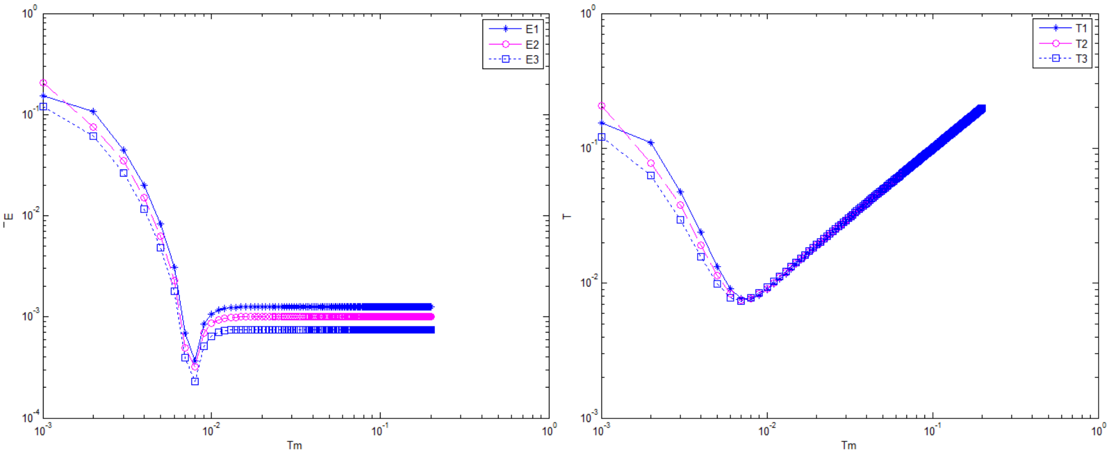


Figure 5.5: Node Time and Error Profiles for Random Topology

onal topology. An important feature to observed here is that, the variance in communication cycles to reach dip region, $\mathbb{V}_{k_{min}^{dip}}$ is zero for all networks. This implies all nodes can synchronize and go to sleep at the time for these type of networks when using TSAU for local node time update. The summary of the dip region results for the 4 node network for the three topologies is given in Table 5.1.

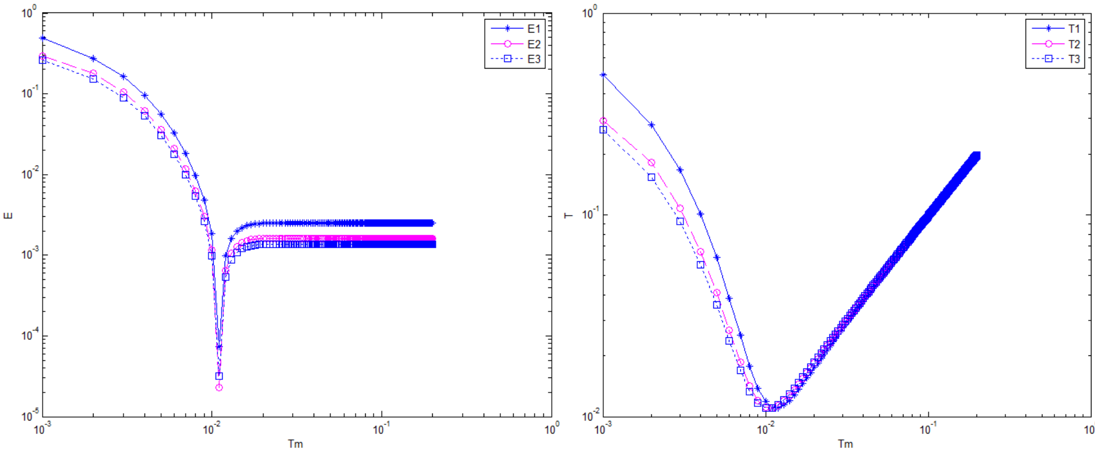


Figure 5.6: Node Time and Error Profiles for Hexagonal Topology

Table 5.1: 4 Node Network Dip Region Results Comparison for Timed Sequential Asynchronous Update

Topologies	**Grid**		**Hex**		**Random**	
Nodes	Iterations	Error	Iterations	Error	Iterations	Error
N1	10	5.76E-04	12	2.99E-04	8	3.69E-04
N2	10	2.88E-04	12	2.16E-04	8	3.24E-04
N3	10	2.88E-04	12	1.72E-04	8	2.31E-04
Statistics						
Mean	10	3.84E-04	12	2.29E-04	8	3.08E-04
Minimum	10	2.88E-04	12	1.72E-04	8	2.31E-04
Maximum	10	5.76E-04	12	2.99E-04	8	3.69E-04
Variance	0.00	2.77E-08	0.00	4.19E-09	0.00	4.97E-09

5.3.1.2 Results for 9 Node Network

Figures 5.7, 5.8, 5.9 show the error profile plot and node time plot both against the gateway time of the 9 node network for grid, random and hexagonal topologies respectively.

We observe from these figures that, the dip phenomenon is exhibited by all network configurations for Timed Sequential Asynchronous Update (TSAU) with all topologies registering, average minimum error in $\mathbb{E}_{min}^{dip}$ in the 10^{-4} range. It is

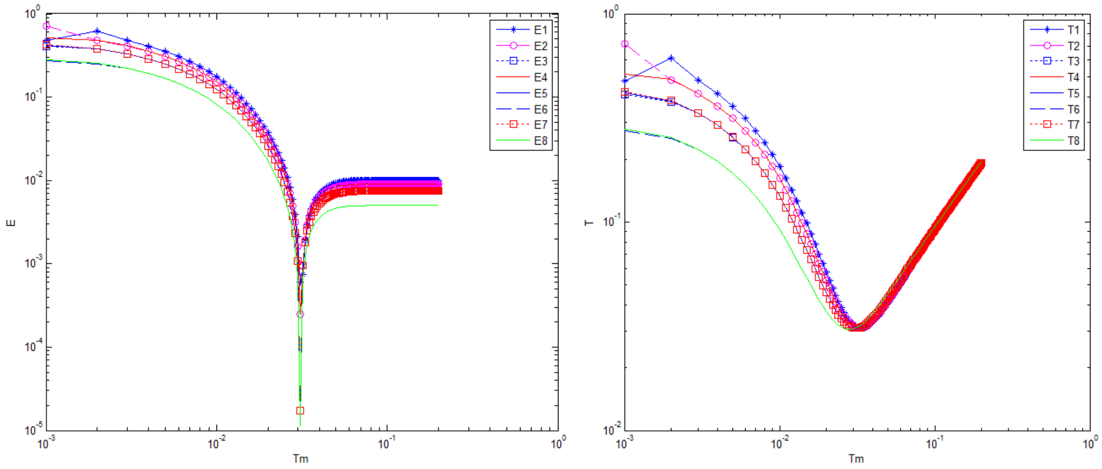


Figure 5.7: Node Time and Error Profiles for Grid Topology

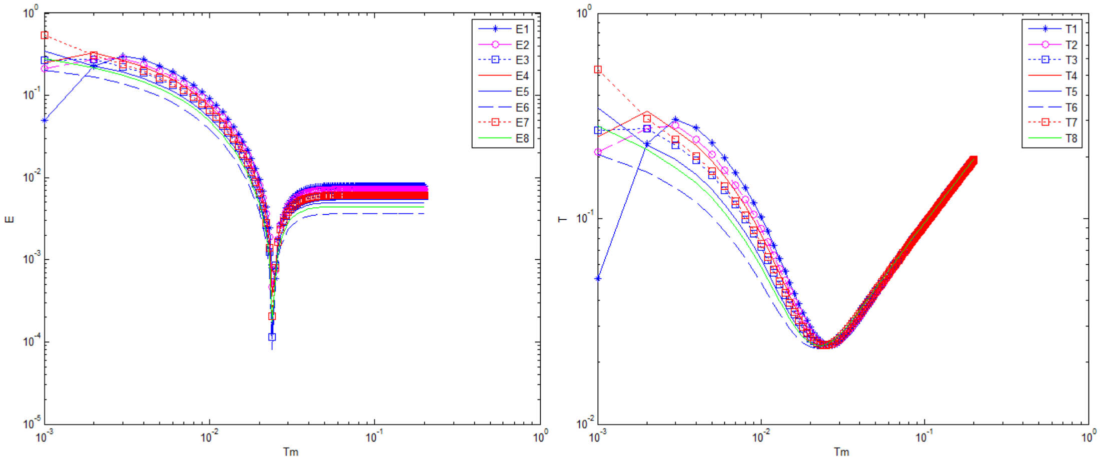


Figure 5.8: Node Time and Error Profiles for Random Topology

observe that $\mathbb{E}_{min}^{dip}$, k_{min}^{dip} and $\mathbb{V}_{k_{min}^{dip}}$ for random and hexagonal topologies are almost the same and better as compared to the grid topology. This stems from the fact that, these topologies have numerous connection as compared to the grid topology and hence achieve better synchronization at less number communications.

The summary of the dip region results for the 9 node network for the three topologies is given in Table 5.2.

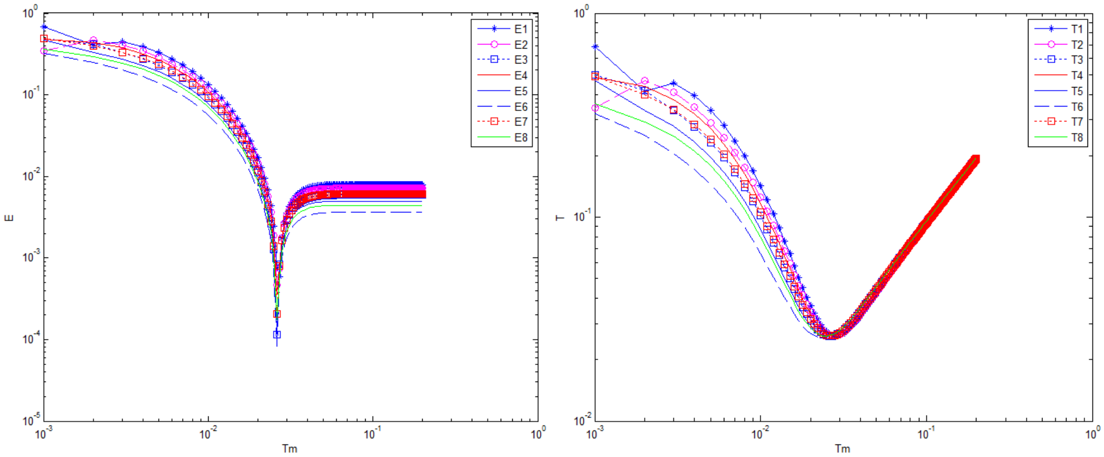


Figure 5.9: Node Time and Error Profiles for Hexagonal Topology

5.3.1.3 Results for 16 Node Network

Figures 5.10, 5.11, 5.12 show the error profile plots and node time plots both against the gateway time of the 16 node network for grid, random and hexagonal topologies respectively.

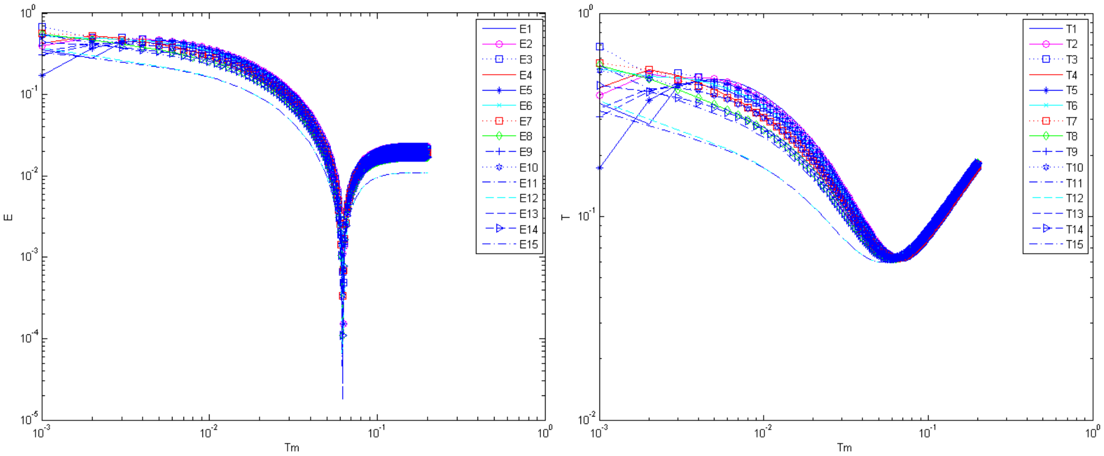


Figure 5.10: Node Time and Error Profiles for Grid Topology

We observe from these figures that, the dip phenomenon is exhibited by all network configurations for Timed Sequential Asynchronous Update (TSAU) with all topologies registering, average minimum error in $\mathbb{E}_{min}^{dip}$ in the 10^{-4} range.

Table 5.2: 9 Node Network Dip Region Results Comparison for Timed Sequential Asynchronous Update

Topologies	**Grid**		**Hex**		**Random**	
Nodes	Iteration	Error	Iteration	Error	Iteration	Error
N1	34	2.30E-04	27	5.87E-04	25	5.88E-04
N2	34	4.74E-04	26	4.65E-04	24	4.63E-04
N3	33	4.04E-04	26	1.15E-04	24	1.14E-04
N4	34	4.74E-04	26	3.62E-04	24	3.60E-04
N5	33	4.04E-04	26	1.29E-04	24	1.28E-04
N6	33	2.69E-04	26	8.14E-05	24	8.07E-05
N7	33	4.04E-04	26	2.05E-04	24	2.04E-04
N8	33	2.69E-04	26	1.74E-04	24	1.73E-04
Statistics						
Mean	33	3.66E-04	26	2.65E-04	24	2.64E-04
Minimum	33	2.30E-04	26	8.14E-05	24	8.07E-05
Maximum	34	4.74E-04	27	5.87E-04	25	5.88E-04
Variance	0.27	9.27E-09	0.13	3.42E-08	0.13	3.44E-08

It is observe that $\mathbb{E}_{min}^{dip}$, k_{min}^{dip} and $\mathbb{V}_{k_{min}^{dip}}$ for random and hexagonal topologies are almost the same with the grid topology exhibiting the best results. The summary of the dip region results for the 16 node network for the three topologies is given in Table 5.3.

5.3.2 Simulation Results for Unidirectional Asynchronous Flooding (UAF)

5.3.2.1 Results for 4 Node Network

Figures 5.13, 5.14, 5.15 show the error, $\mathbf{E}_k$ profile [5]plot and node time, $\mathbf{T}_k$ values plot both against the gateway time of the 4 node network for grid, random and hexagonal topologies respectively. We observe from these figures that, the dip

---

[5] Refer Equation 3.4, section 3.2.1

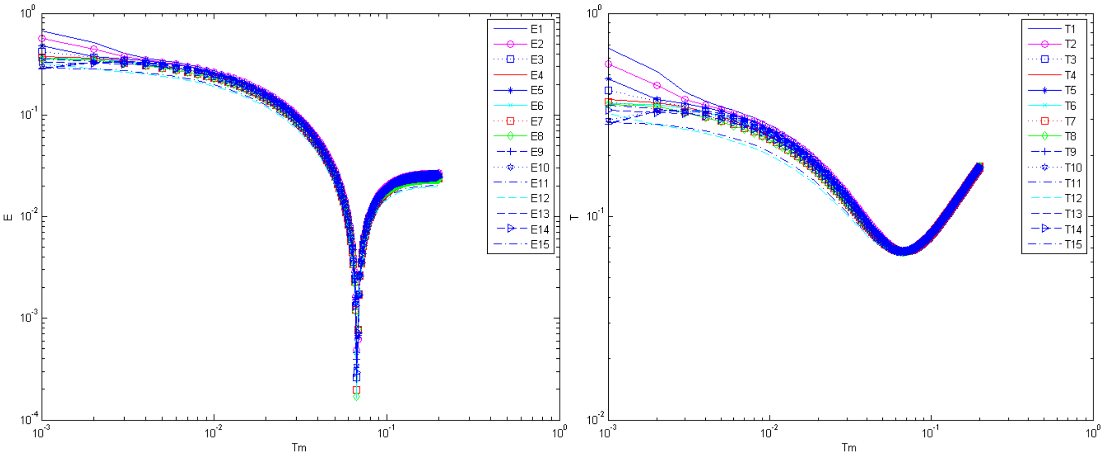


Figure 5.11: Node Time and Error Profiles for Random Topology

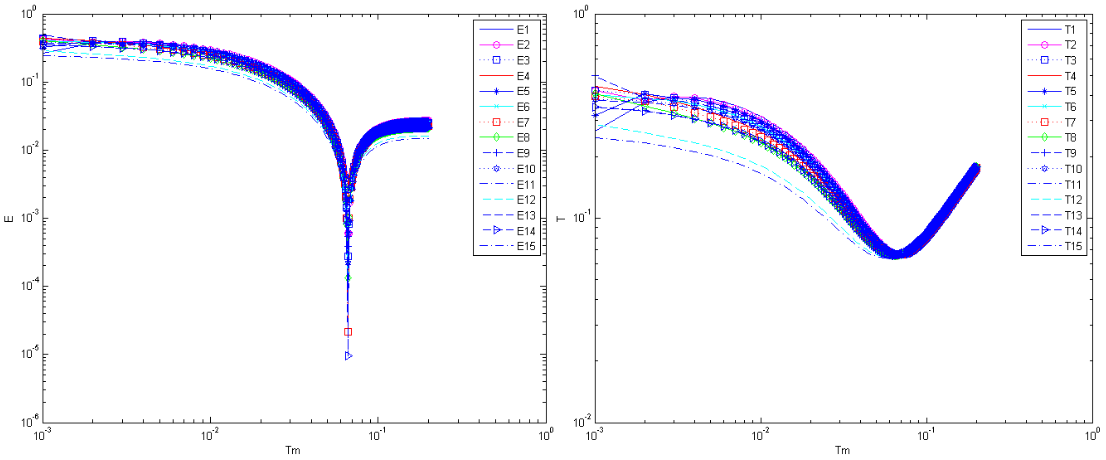


Figure 5.12: Node Time and Error Profiles for Hexagonal Topology

phenomenon is exhibited by all network configurations for Unidirectional Asynchronous Flooding (UAF) with all topologies registering, average minimum error in $\mathbb{E}_{min}^{dip}$ in the 10^{-4} range although $\mathbb{E}_{min}^{dip}$, for random where all nodes are interconnected is lowest followed by hexagonal and then the grid network. This is also observed for the other two parameters, k_{min}^{dip} and $\mathbb{V}_{k_{min}^{dip}}$. An important observation for UAF is, the dip region results for all nodes in the 4 node network are equal. This implies that, nodes can synchronize, sleep and wake-up at the same time. The summary of the dip region results for the 16 node network for the three

Table 5.3: 16 Node Network Dip Region Results Comparison for Timed Sequential Asynchronous Update

Topologies	**Grid**		**Hex**		**Random**	
Nodes	Iteration	Error	Iteration	Error	Iteration	Error
N1	63	1.12E-04	67	4.30E-04	68	5.30E-04
N2	63	1.51E-04	67	5.86E-04	67	4.83E-04
N3	63	4.86E-04	66	2.72E-04	67	2.62E-04
N4	62	3.59E-04	66	2.51E-05	67	2.14E-04
N5	63	1.51E-04	66	5.42E-04	67	4.56E-04
N6	63	4.09E-04	66	2.72E-04	67	3.19E-04
N7	62	3.33E-04	66	2.15E-05	67	1.96E-04
N8	62	1.08E-04	66	1.33E-04	67	1.69E-04
N9	63	4.86E-04	66	3.89E-04	67	3.96E-04
N10	62	3.33E-04	66	2.23E-04	67	3.39E-04
N11	62	1.72E-05	66	6.04E-05	67	2.67E-04
N12	62	3.04E-05	66	1.21E-04	67	1.65E-04
N13	62	3.59E-04	66	1.89E-04	67	3.58E-04
N14	62	1.08E-04	66	9.43E-06	67	2.91E-04
N15	62	3.04E-05	66	4.76E-05	67	2.25E-04
			Statistics			
Mean	62	2.32E-04	66	2.21E-04	67	3.11E-04
Minimum	62	1.72E-05	66	9.43E-06	67	1.65E-04
Maximum	63	4.86E-04	67	5.86E-04	68	5.30E-04
Variance	0.26	2.85E-08	0.12	3.65E-08	0.07	1.32E-08

topologies is given in Table 5.4.

5.3.2.2 Results for 9 Node Network

Figures 5.16, 5.17, 5.18 show the error, $\mathbf{E}_k$ profile plot and node time, $\mathbf{T}_k$ plot both against the gateway time of the 9 node network for grid, random and hexagonal topologies respectively. We observe from these figures that, the dip phenomenon is exhibited by all network configurations for Unidirectional Asynchronous Flooding (UAF) with all topologies registering, average minimum error in $\mathbb{E}_{min}^{dip}$ in the 10^{-4} range. An important observation for UAF is, the dip region error for nodes in the

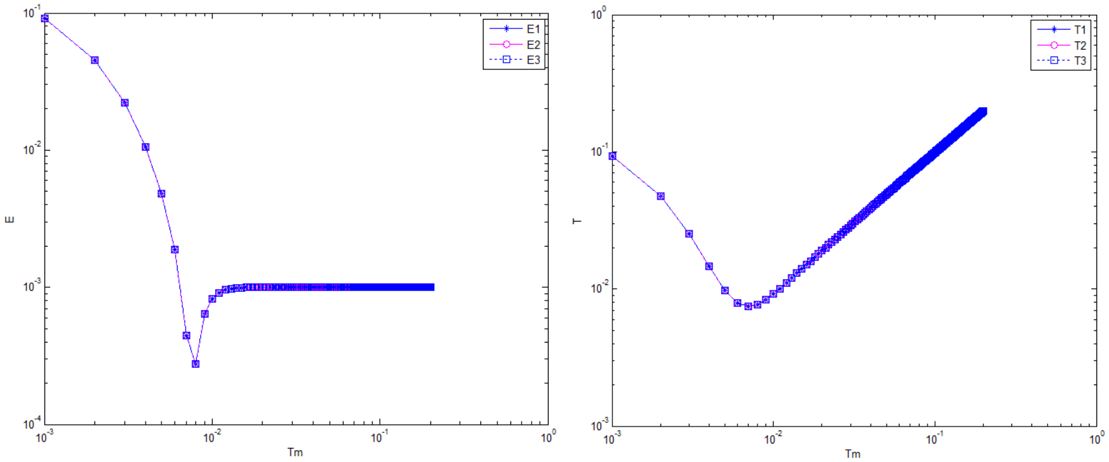


Figure 5.13: Node Time and Error Profiles for Grid Topology

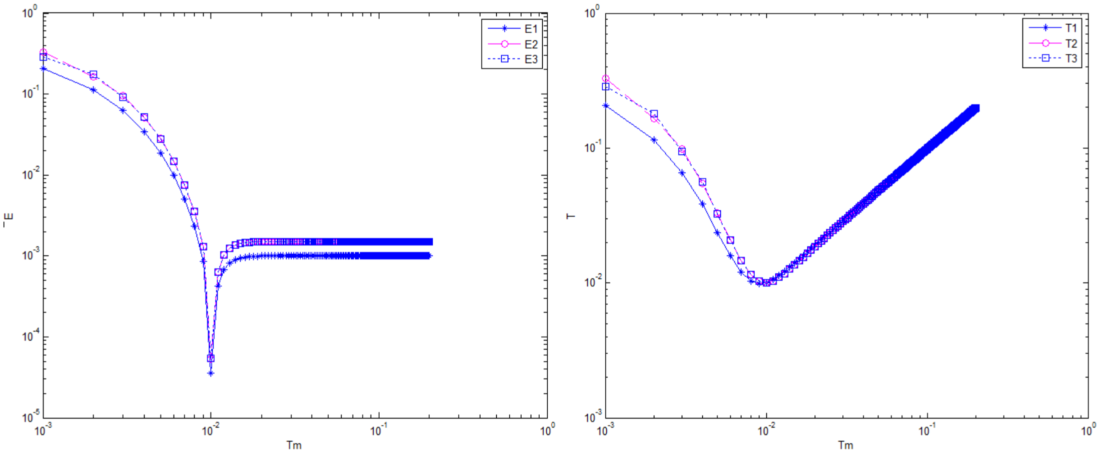


Figure 5.14: Node Time and Error Profiles for Random Topology

network are equal very close but it is observed that, $\mathbb{V}_{k_{min}^{dip}}$ for all networks is null which except for the random topology where, $\mathbb{V}_{k_{min}^{dip}} = 0.13$. With this advantage, nodes can synchronize, sleep and wake-up nearly at the same time. The summary of the dip region results for the 16 node network for the three topologies is given in Table 5.5.

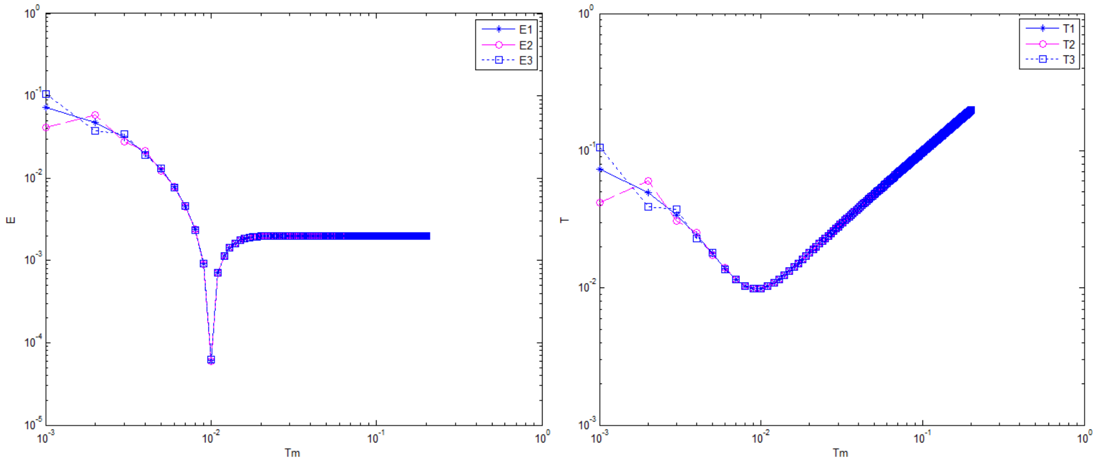


Figure 5.15: Node Time and Error Profiles for Hexagonal Topology

Table 5.4: 4 Node Network Dip Region Results Comparison for Unidirectional Asynchronous Flooding

Topologies	**Grid**		**Hex**		**Random**	
Nodes	Iteration	Error	Iteration	Error	Iteration	Error
N1	8	2.77E-04	13	2.15E-04	10	3.58E-05
N2	8	2.77E-04	13	2.15E-04	10	5.26E-05
N3	8	2.77E-04	13	2.15E-04	10	5.47E-05
Statistics						
Mean	8	2.77E-04	13	2.15E-04	10	4.77E-05
Minimum	8	2.77E-04	13	2.15E-04	10	3.58E-05
Maximum	8	2.77E-04	13	2.15E-04	10	5.47E-05
Variance	0.00	0.00E+00	0.00	2.01E-14	0.00	1.08E-10

5.3.2.3 Results for 16 Node Network

Figures 5.19, 5.20, 5.21 show the error, $\mathbf{E}_k$ profile plot and node time, $\mathbf{T}_k$ plot both against the gateway time of the 16 node networks.

We observe from these figures that, the dip phenomenon is exhibited by all network configurations for Unidirectional Asynchronous Flooding (UAF) with all topologies registering, average minimum error in $\mathbb{E}_{min}^{dip}$ in the 10^{-4} range. An important observation for UAF is, the dip region error for nodes in the network

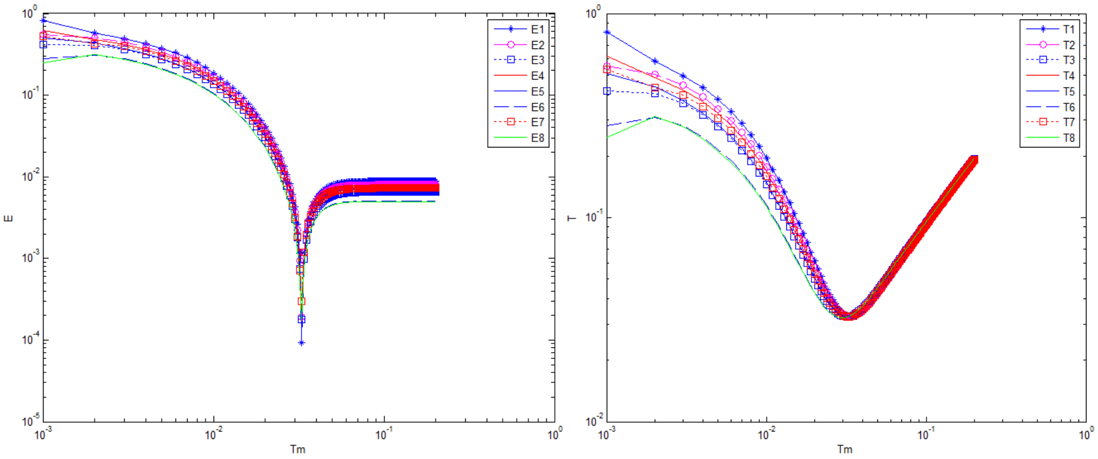


Figure 5.16: Node Time and Error Profiles for Grid Topology

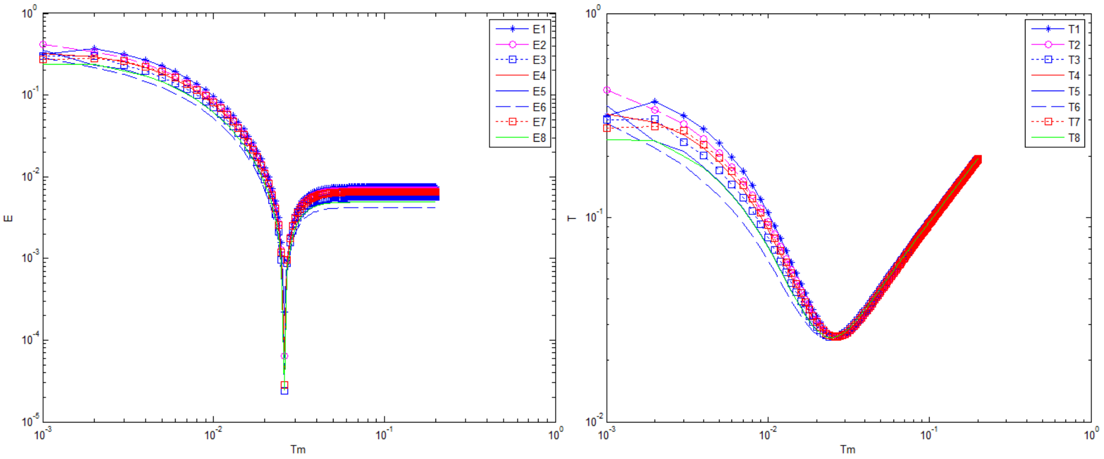


Figure 5.17: Node Time and Error Profiles for Random Topology

are equal very close but more importantly, $\mathbb{E}_{min}^{dip}$ for all networks is null which is unprecedented for a large network. With this advantage, nodes can synchronize, sleep and wake-up at the same time. The summary of the dip region results for the 16 node network for the three topologies is given in Table 5.6.

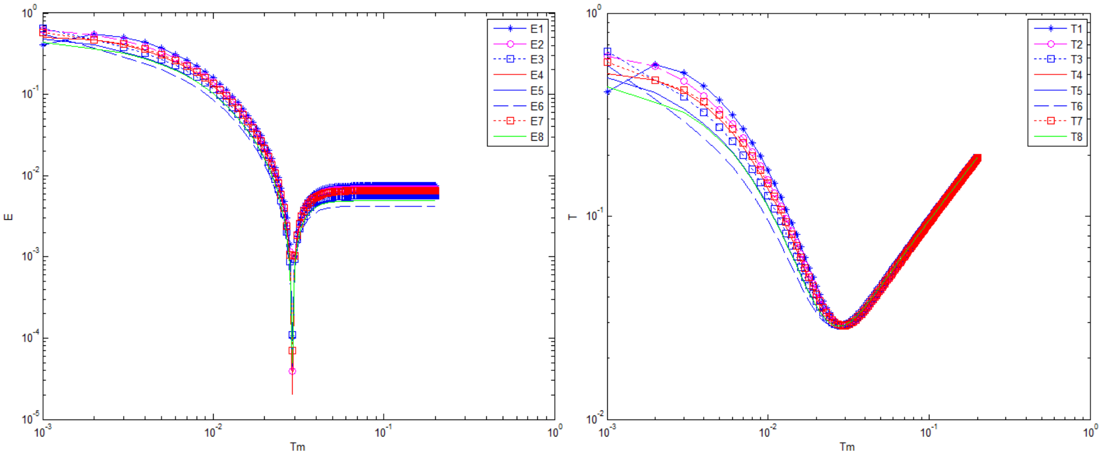


Figure 5.18: Node Time and Error Profiles for Hexagonal Topology

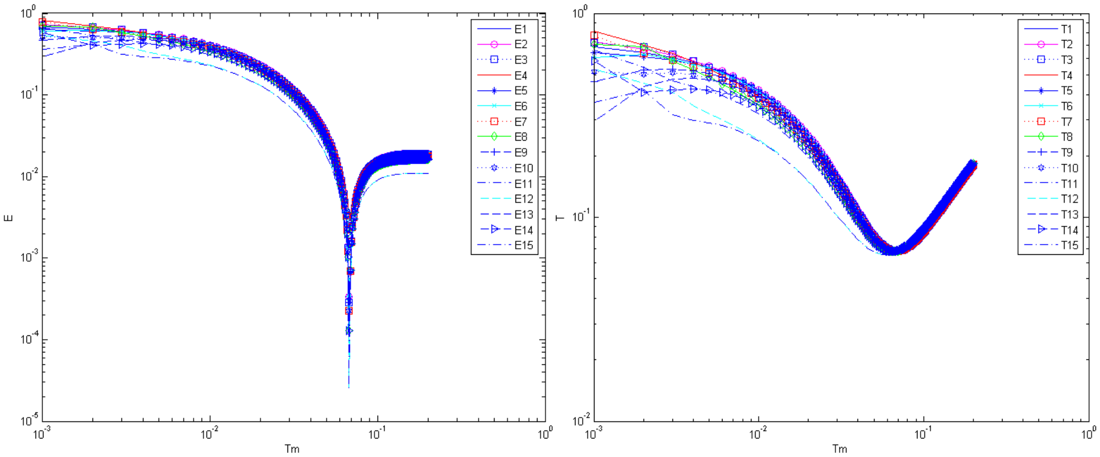


Figure 5.19: Node Time and Error Profiles for Grid Topology

5.3.3 Simulation Results for Bidirectional Asynchronous Flooding (BAF)

5.3.3.1 Results for 4 Node Network

Figures 5.22, 5.23, 5.24 show the error, $\mathbf{E}_k$ profile [6]plot and node time, $\mathbf{T}_k$ values plot both against the gateway time of the 4 node network for grid, random and hexagonal topologies respectively. We observe from these figures that, the dip phe-

---

[6] Refer Equation 3.4, section 3.2.1

Table 5.5: 9 Node Network Dip Region Results Comparison for Unidirectional Asynchronous Flooding

Topologies	**Grid**		**Hex**		**Random**	
Nodes	Iteration	Error	Iteration	Error	Iteration	Error
N1	33	9.27E-05	29	1.03E-04	25	5.88E-04
N2	33	1.93E-04	29	3.95E-05	24	4.63E-04
N3	33	1.77E-04	29	1.09E-04	24	1.14E-04
N4	33	1.85E-04	29	2.04E-05	24	3.60E-04
N5	33	1.62E-04	29	4.54E-05	24	1.28E-04
N6	33	1.62E-04	29	1.05E-04	24	8.07E-05
N7	33	3.00E-04	29	7.05E-05	24	2.04E-04
N8	33	1.75E-04	29	4.95E-05	24	1.73E-04
Statistics						
Mean	33	1.81E-04	29	6.78E-05	24	2.64E-04
Minimum	33	9.27E-05	29	2.04E-05	24	8.07E-05
Maximum	33	3.00E-04	29	1.09E-04	25	5.88E-04
Variance	0.00	3.27E-09	0.00	1.17E-09	0.13	3.44E-08

nomenon is exhibited by all network configurations for Bidirectional Asynchronous Flooding (BAF) with all topologies registering, average minimum error in $\mathbb{E}^{dip}_{min}$ in the 10^{-4} range although $\mathbb{E}^{dip}_{min}$, for random where all nodes are interconnected is lowest followed by hexagonal and then the grid network.

This is also observed for the other two parameters, k^{dip}_{min} and $\mathbb{V}_{k^{dip}_{min}}$. An important observation for BAF is that the dip region results for all nodes in the 4 node network are equal. This implies that, nodes can synchronize, sleep and wake-up at the same time. The summary of the dip region results for the 16 node network for the three topologies is given in Table 5.7.

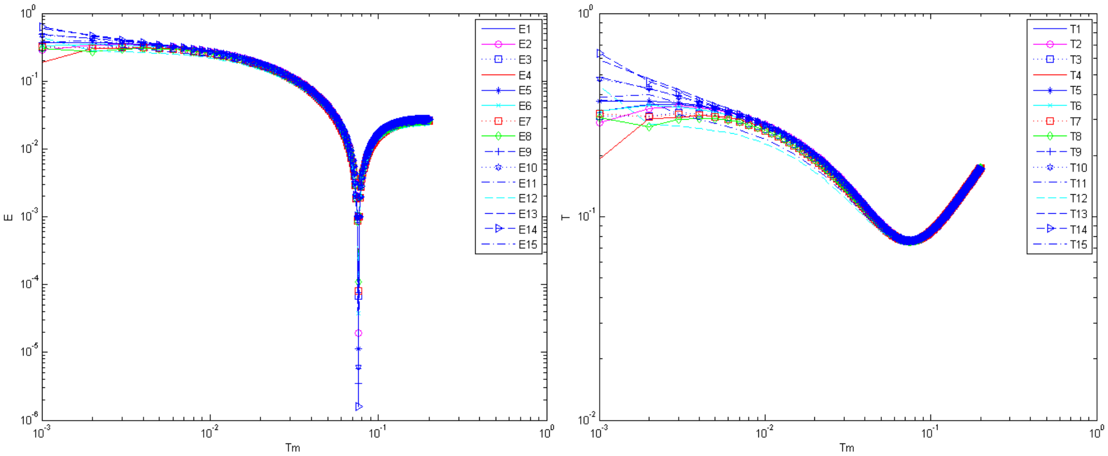


Figure 5.20: Node Time and Error Profiles for Random Topology

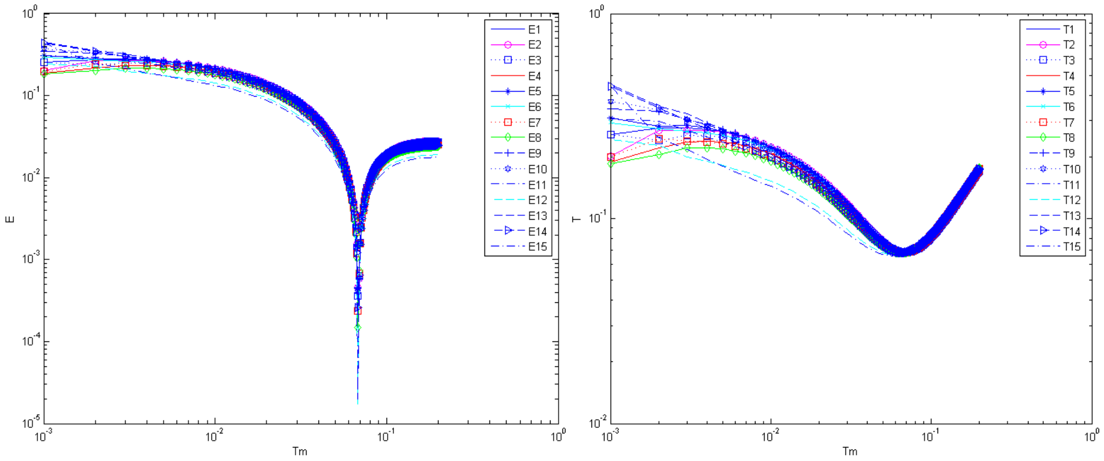


Figure 5.21: Node Time and Error Profiles for Hexagonal Topology

5.3.3.2 Results for 9 Node Network

Figures 5.25, 5.26, 5.27 show the error, $\mathbf{E}_k$ profile plot and node time, $\mathbf{T}_k$ plot both against the gateway time of the 9 node network for grid, random and hexagonal topologies respectively. We observe from these figures that, the dip phenomenon is exhibited by all network configurations for Bidirectional Asynchronous Flooding (BAF) with all topologies registering, average minimum error in $\mathbb{E}_{min}^{dip}$ in the 10^{-4} range.

Table 5.6: 16 Node Network Dip Region Results Comparison for Unidirectional Asynchronous Flooding

Topologies	**Grid**		**Hex**		**Random**	
Nodes	Iteration	Error	Iteration	Error	Iteration	Error
N1	68	3.38E-04	68	4.55E-04	76	1.54E-05
N2	68	3.38E-04	68	4.52E-04	76	1.93E-05
N3	68	2.87E-04	68	3.59E-04	76	6.78E-05
N4	68	2.24E-04	68	2.45E-04	76	1.08E-04
N5	68	3.38E-04	68	4.58E-04	76	1.14E-05
N6	68	3.18E-04	68	3.85E-04	76	3.74E-05
N7	68	2.27E-04	68	2.39E-04	76	8.12E-05
N8	68	1.30E-04	68	1.49E-04	76	1.09E-04
N9	68	2.87E-04	68	4.47E-04	76	3.50E-06
N10	68	2.27E-04	68	4.00E-04	76	6.15E-06
N11	68	8.84E-05	68	2.17E-04	76	3.88E-05
N12	68	2.57E-05	68	1.64E-05	76	1.02E-04
N13	68	2.24E-04	68	4.06E-04	76	1.24E-05
N14	68	1.30E-04	68	2.63E-04	76	1.60E-06
N15	68	2.57E-05	68	2.01E-05	76	6.50E-05
Statistics						
Mean	68	2.14E-04	68	3.01E-04	76	4.53E-05
Minimum	68	2.57E-05	68	1.64E-05	76	1.60E-06
Maximum	68	3.38E-04	68	4.58E-04	76	1.09E-04
Variance	0.00	1.21E-08	0.00	2.31E-08	0.00	1.61E-09

We observe from the results that, the dip region error for nodes in the network are equal very close but it is observed that, $\mathbb{V}_{k_{min}^{dip}}$ for the grid, random and hexagonal topologies are 2.48, 1.7 and 1.07 respectively. From these results, it is clear that, the more connected the network, the better its dip region performance with respect to k_{min}^{dip} and $\mathbb{V}_{k_{min}^{dip}}$. With this advantage, nodes can synchronize, sleep and wake-up nearly at the same time. The summary of the dip region results for the 16 node network for the three topologies is given in Table 5.8.

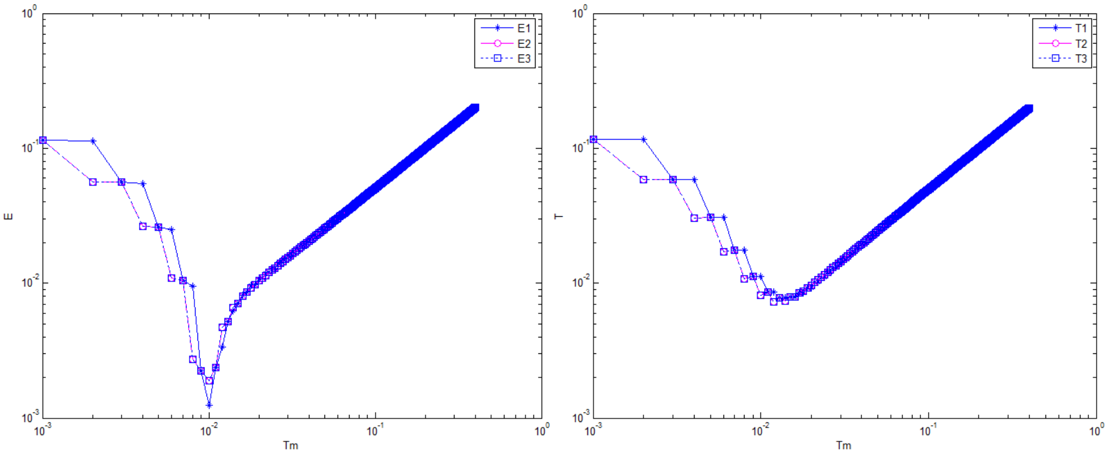


Figure 5.22: Node Time and Error Profiles for Grid Topology

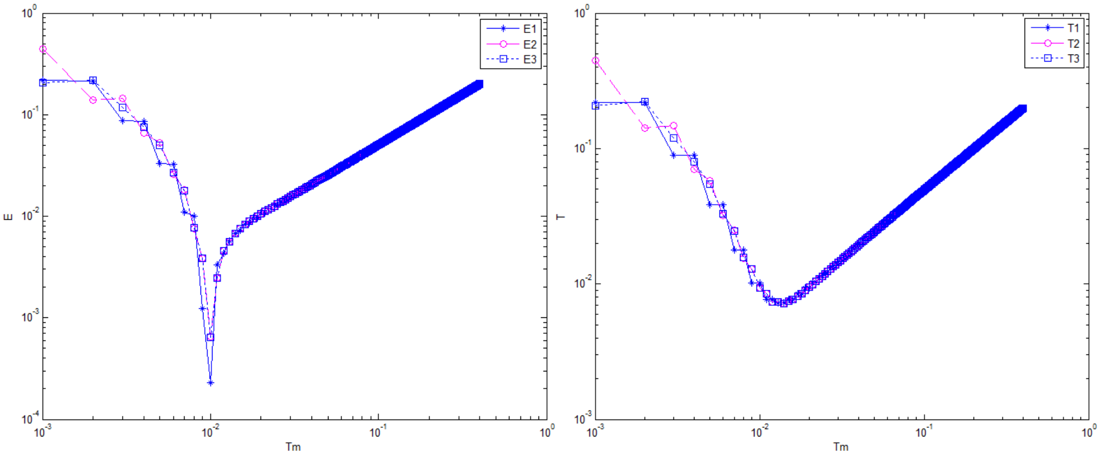


Figure 5.23: Node Time and Error Profiles for Random Topology

5.3.3.3 Results for 16 Node Network

Figures 5.28, 5.29, 5.30 show the error, $\mathbf{E}_k$ profile plot and node time, $\mathbf{T}_k$ plot both against the gateway time of the 16 node network for grid, random and hexagonal topologies respectively.

We observe from the results that, the dip region error for nodes in the network are equal very close but it is observed that, $\mathbb{V}_{k_{min}^{dip}}$ for the grid, random and hexagonal topologies are 8.60, 7.64 and 2.60 respectively. From these results, it is

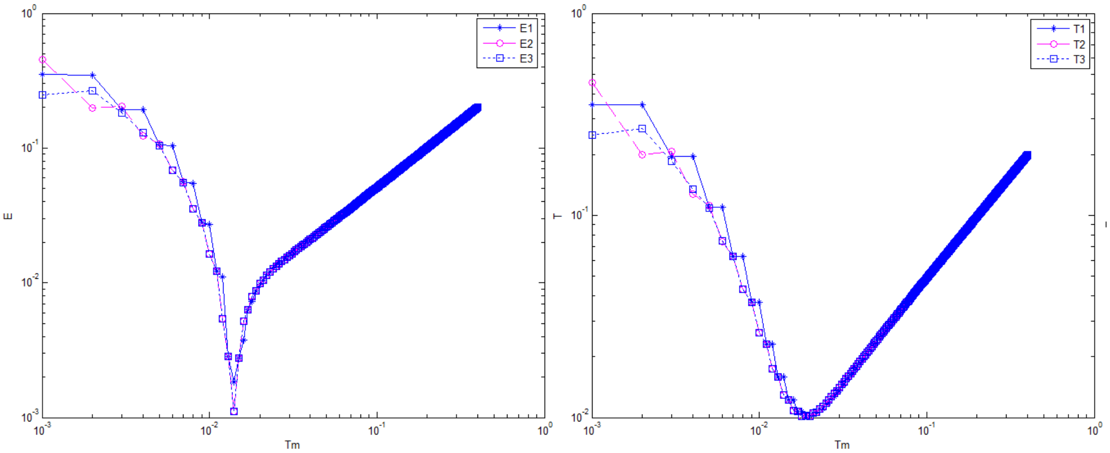


Figure 5.24: Node Time and Error Profiles for Hexagonal Topology

Table 5.7: 4 Node Network Dip Region Results Comparison for Bidirectional Asynchronous Flooding

Topologies	**Grid**		**Hex**		**Random**	
Nodes	Iteration	Error	Iteration	Error	Iteration	Error
N1	8	2.77E-04	13	2.15E-04	10	3.58E-05
N2	8	2.77E-04	13	2.15E-04	10	5.26E-05
N3	8	2.77E-04	13	2.15E-04	10	5.47E-05
Statistics						
Mean	8	2.77E-04	13	2.15E-04	10	4.77E-05
Minimum	8	2.77E-04	13	2.15E-04	10	3.58E-05
Maximum	8	2.77E-04	13	2.15E-04	10	5.47E-05
Variance	0.00	0.00E+00	0.00	2.01E-14	0.00	1.08E-10

clear that, the more connected the network, the better its dip region performance with respect to k_{min}^{dip} and $\mathbb{V}_{k_{min}^{dip}}$.

With this advantage, nodes can synchronize, sleep and wake-up nearly at the same time. The summary of the dip region results for the 16 node network for the three topologies is given in Table 5.9.

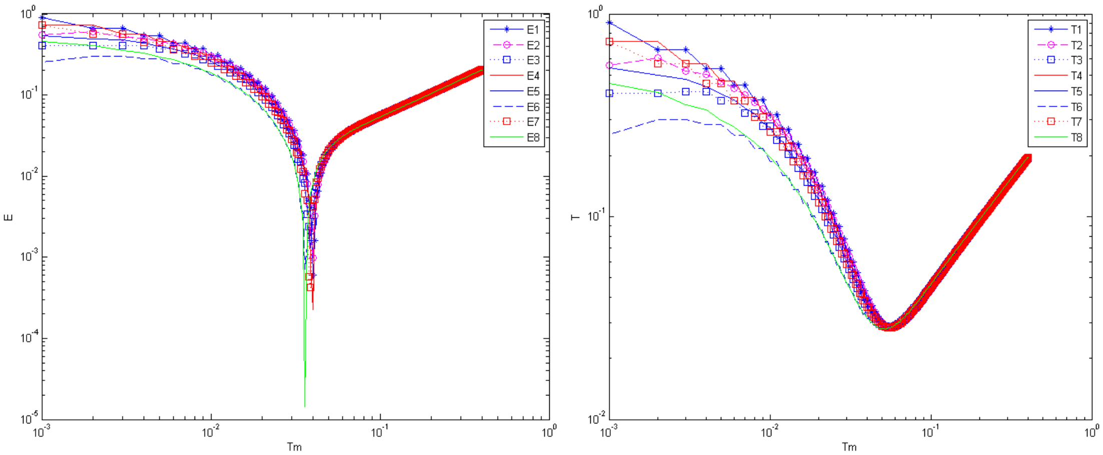


Figure 5.25: Node Time and Error Profiles for Grid Topology

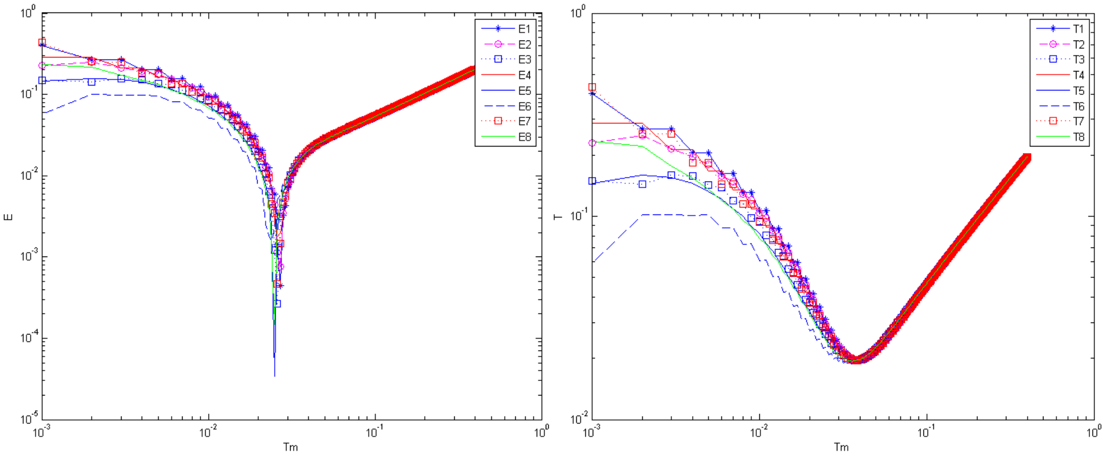


Figure 5.26: Node Time and Error Profiles for Random Topology

5.3.4 Simulation Comparison between Proposed Methods of Local Time Update

In this section, we compare the performance of TSAU, UAF and BAF based on the average minimum error in the dip region, $\mathbb{E}^{dip}_{min}$, the number of communication cycles needed to reach the minimum error, k^{dip}_{min} and the variance in k^{dip}_{min}, $\mathbb{V}^{dip}_{min}$ [7] on the presented simulation results of each method. For each method of local

---

[7] Detailed explanations on the importance of these parameters in the design of our proposed sync protocol is given in section 5.3.1

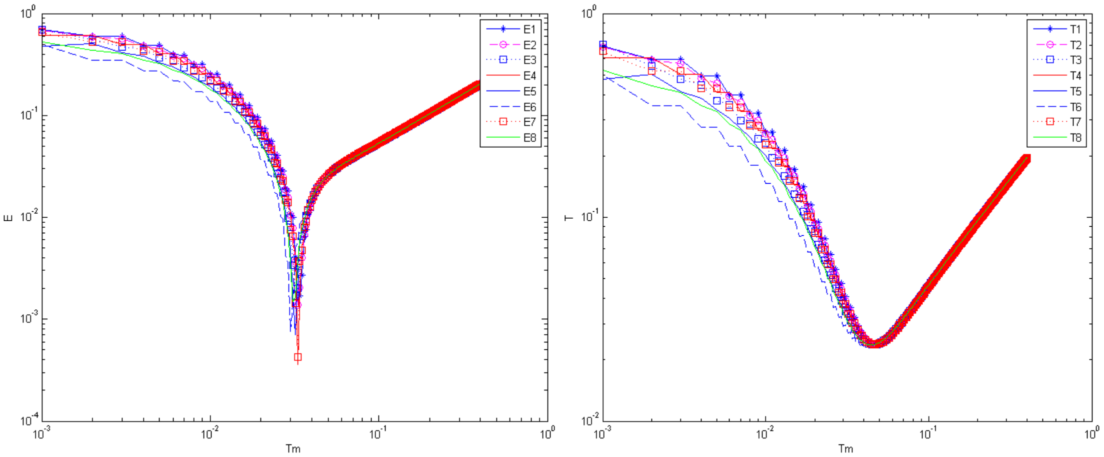


Figure 5.27: Node Time and Error Profiles for Hexagonal Topology

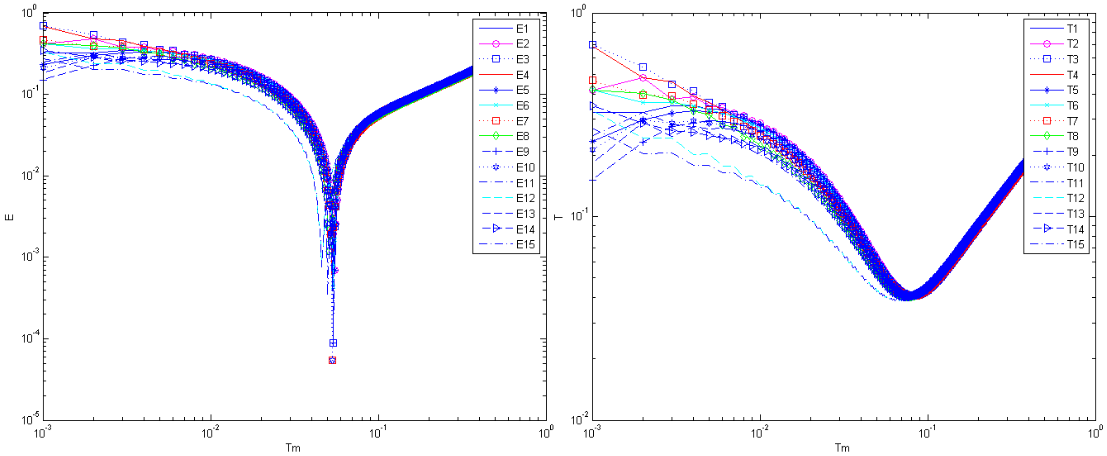


Figure 5.28: Node Time and Error Profiles for Grid Topology

time update, we present the results on these critical parameters for 4, 9, and 16 node networks. In addition to the above mentioned methods, we include the dip region results on the three critical parameters for the *Synchronous Time Update, (STU)* [8]. The comparison is done by employing bar graphs for each parameter.

---

[8] Theoretical descriptions and implementation algorithm of STU are respectively given in sections 3.2.1 and 4.3.7

Table 5.8: 9 Node Network Dip Region Results Comparison for Bidirectional Asynchronous Flooding

Topologies	**Grid**		**Hex**		**Random**	
Nodes	Iteration	Error	Iteration	Error	Iteration	Error
N1	40	6.05E-04	33	4.33E-04	27	4.42E-04
N2	40	9.86E-04	33	1.02E-03	27	7.66E-04
N3	39	1.77E-03	32	6.21E-04	26	2.66E-04
N4	40	2.25E-04	32	1.82E-03	26	1.76E-03
N5	39	1.30E-03	31	4.71E-04	25	3.40E-05
N6	36	7.16E-04	29	1.38E-03	24	1.58E-03
N7	39	4.25E-04	32	8.62E-04	26	4.62E-04
N8	36	1.44E-05	31	6.87E-04	25	1.49E-04
Statistics						
Mean	39	7.56E-04	32	9.12E-04	26	6.83E-04
Minimum	36	1.44E-05	29	4.33E-04	24	3.40E-05
Maximum	40	1.77E-03	33	1.82E-03	27	1.76E-03
Variance	2.84	3.35E-07	1.70	2.31E-07	1.07	4.24E-07

5.3.4.1 Minimum Error in Dip Region, $\mathbb{E}_{min}^{dip}$

In Figure 5.31, we compare the synchronization accuracy in the dip region for TSAU, UAF, BAF and STU. From the results, we observe that UAF registers the lowest error as compared to all the other methods for all networks and BAF registers the highest for the large networks. In the small network of 4 nodes, its accuracy in the dip region is nearly similar to that of UAF and slightly lower than the dip error of STU. TSAU is observed to have a higher error than UAF and STU for all the 4 and 9 node networks but has a similar minimum dip error as STU in the 16 node network.

We also observe that, for TSAU, synchronization accuracy increases as the network gets bigger. This pattern is not observed for the other methods. In general, we can conclude here that UAF showed the best accuracy in terms of

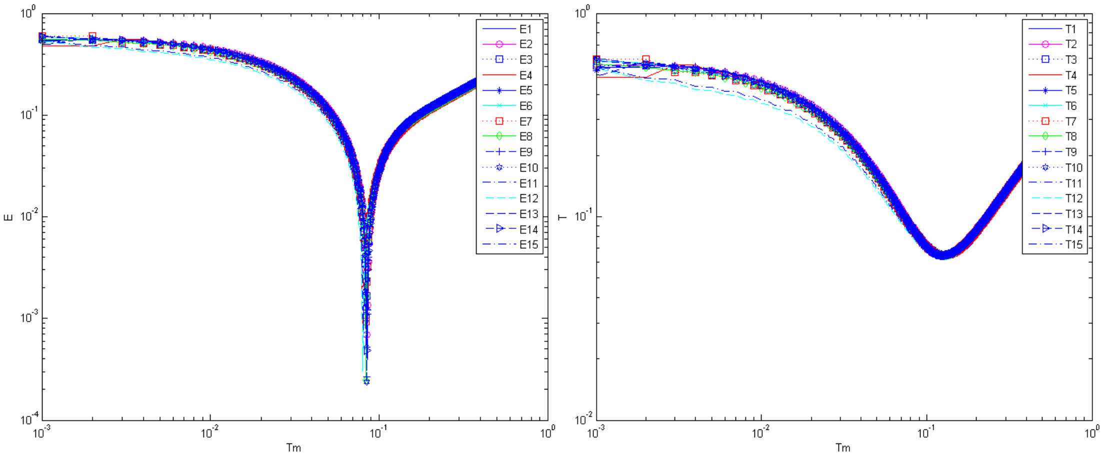


Figure 5.29: Node Time and Error Profiles for Random Topology

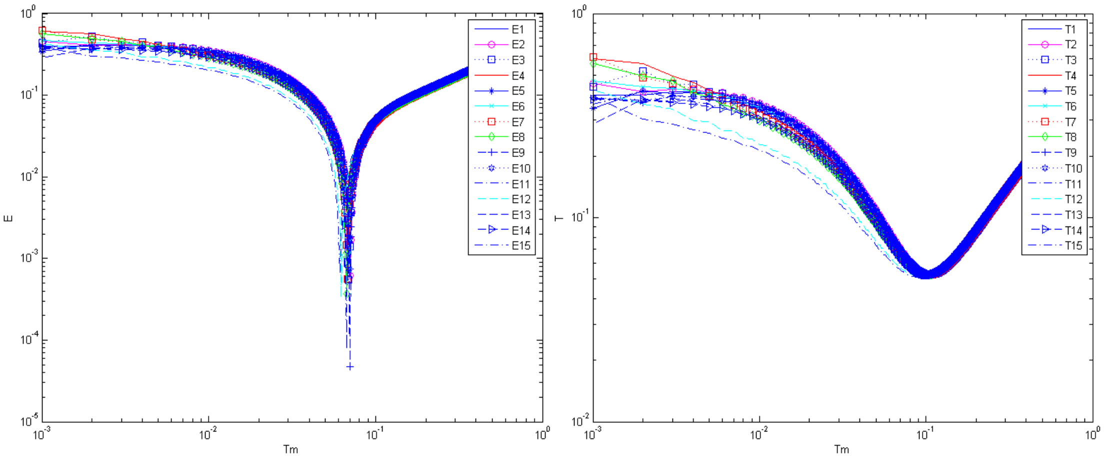


Figure 5.30: Node Time and Error Profiles for Hexagonal Topology

synchronization in the dip region. This is followed by STU then TSAU. BAF was observed to have the least synchronization accuracy especially for the large networks.

5.3.4.2 Communication Cycles to Reach Dip Region, k_{min}^{dip}

In Figure 5.32, we compare the number of iterations required to reach minimum error in the dip region for TSAU, UAF, BAF and STU. From the results, we observe that UAF and BAF required the least number of iterations to reach the

Table 5.9: 16 Node Network Dip Region Results Comparison for Bidirectional Asynchronous Flooding

Topologies	**Grid**		**Hex**		**Random**	
Nodes	Iteration	Error	Iteration	Error	Iteration	Error
N1	55	6.96E-04	70	1.63E-03	86	3.94E-04
N2	55	6.96E-04	70	6.17E-04	85	6.92E-04
N3	54	8.78E-05	69	5.45E-04	84	1.03E-03
N4	53	3.89E-04	68	6.81E-04	85	5.73E-04
N5	55	6.96E-04	70	7.42E-04	85	5.20E-04
N6	54	4.37E-04	69	7.34E-04	85	9.03E-04
N7	53	5.40E-05	68	5.65E-04	84	9.27E-04
N8	51	1.18E-03	67	3.69E-04	84	2.49E-04
N9	54	8.78E-05	70	4.77E-05	85	2.69E-04
N10	53	5.40E-05	69	7.42E-04	85	2.38E-04
N11	50	3.32E-04	67	7.71E-05	84	5.13E-04
N12	46	7.59E-04	62	3.37E-04	80	3.04E-04
N13	53	3.89E-04	69	6.59E-04	85	8.51E-04
N14	51	1.18E-03	67	9.00E-04	85	4.88E-04
N15	46	7.59E-04	61	6.66E-04	81	7.67E-04
			Statistics			
Mean	52	5.20E-04	68	6.21E-04	84	5.81E-04
Minimum	46	5.40E-05	61	4.77E-05	80	2.38E-04
Maximum	55	1.18E-03	70	1.63E-03	86	1.03E-03
Variance	8.60	1.40E-07	7.64	1.38E-07	2.60	7.15E-08

dip region for the 4 node network, followed by TSAU and the STU. For the 9 nodes network, UAF and TSAU required the least number of iterations to reach the dip region followed by BAF then STU. This pattern is similar to the 16 node network except BAF outperforms UAF by a margin of about 16 iterations and TSAP outperforms UAF by a margin of about 6 iterations. In general k_{min}^{dip} increases as the number of network nodes increase.

In general, we can infer here that, for the number of iterations needed to reach the minimum error in the dip region, BAF performs best, followed by TSAU then UAF. STU was observed to have the least performance as shown in Figure 5.32

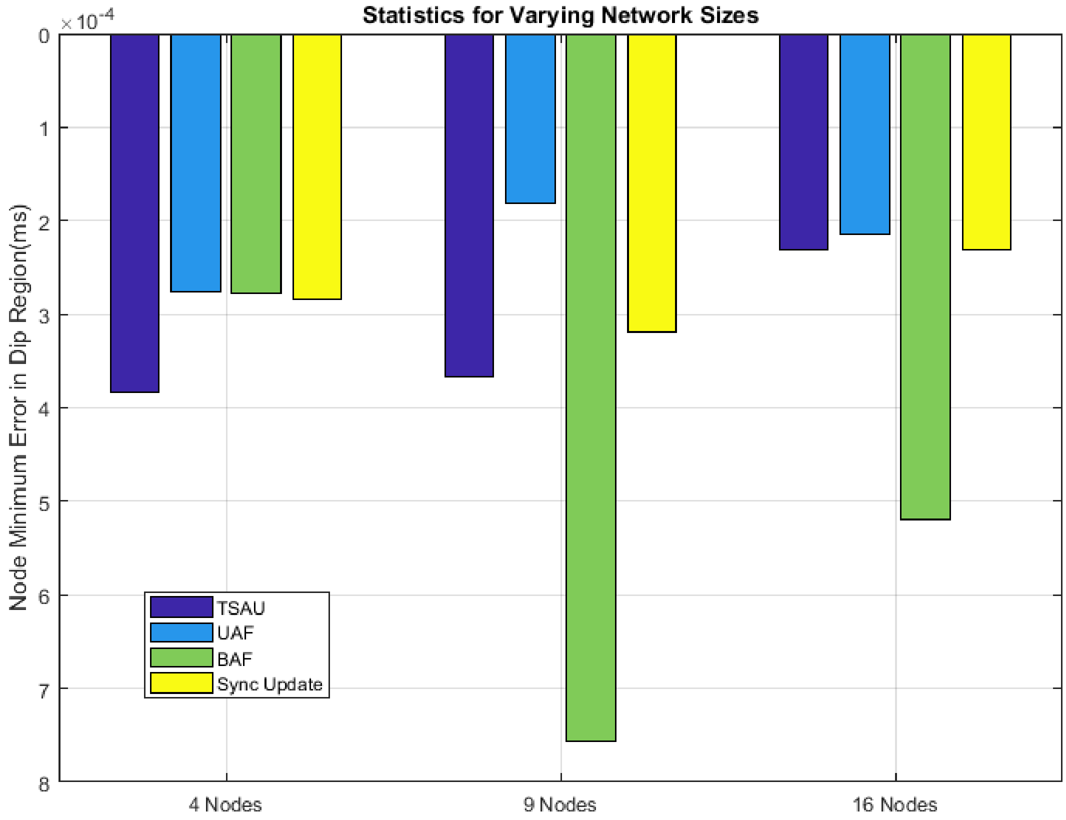


Figure 5.31: Simulation Comparison of Minimum Errors, $\mathbb{E}_{min}^{dip}$ for TSAU, UAF and BAF For Varying Network Sizes

although from the operation of STU [9], all nodes update at each iteration whereas in TSAU [10], UAF [11] and BAF[12] only a subset of the nodes update, in each iteration time values of nodes not activated do not update but since some nodes update, we assume it is fair to take an iteration in all the methods to be the same.

5.3.4.3 Variance in Communication Cycles to Reach Dip Region, $\mathbb{V}_{k_{min}^{dip}}$

In Figure 5.33, we compare the variances in the number of iterations required to reach minimum error in the dip region for TSAU, UAF, BAF and STU.

---

[9] Refer Algorithm 2, section 4.3.7
[10] Refer Algorithm 3, section 5.2.1
[11] Refer Algorithm 4, section 5.2.2
[12] Refer Algorithm 5, section 5.2.3

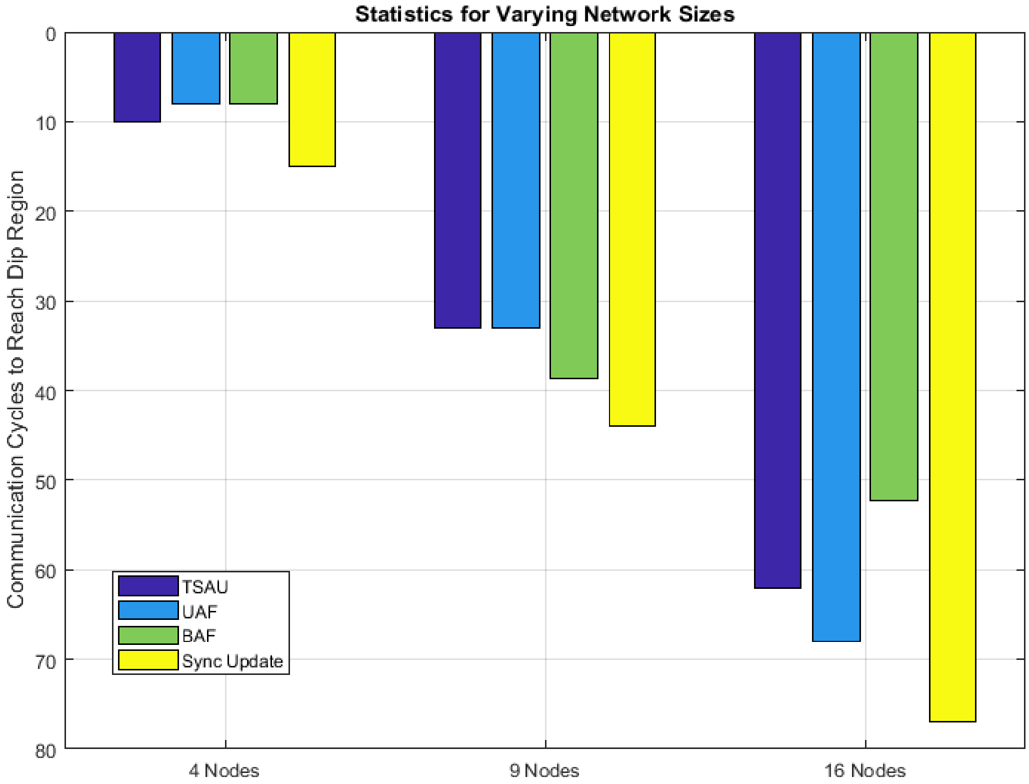


Figure 5.32: Simulation Comparison of Communication Cycles, k_{min}^{dip} for TSAU, UAF and BAF For Varying Network Sizes

First we observe that all methods had null variances for the small network. For the large networks, we observe that UAF performs best by having zero variance for all networks. This is a very crucial since all nodes can stop update, sleep, wakeup and activate at the same time. This unique feature if replicated in the practical results, will give UAF a clear advantage over all the other method. TSAP has the next best performance for the 9 and 16 node networks, followed by STU. The performance of BAF on the large networks was the least for this metric.

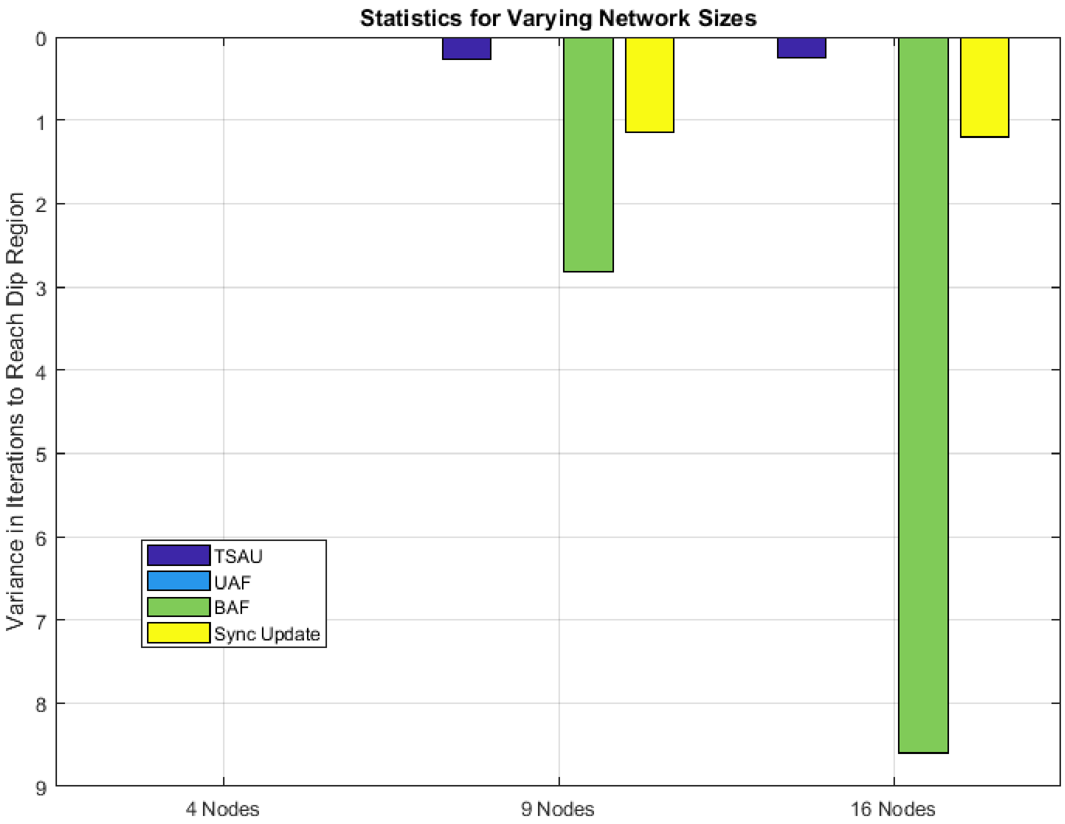


Figure 5.33: Simulation Comparison of Variances in Communication Cycles, $\mathbb{V}_{k_{min}^{dip}}$ for TSAU, UAF and BAF For Varying Network Sizes

5.4 Practical Evaluation of Proposed Methods of Local Time Update

In this section, we evaluate the performance of TSAP, UAF and BAF by looking at the error profiles and the evolution of the node time as a function of the gateway time. In this practical evaluation we adopt only the grid topologies shown in section 4.3.3 and the performance of each method is tested using the criteria given in section 5.3.1. A summary of the critical values in the dip region is also given for each method. Detailed explanations on the methods and materials used in the practical experiments are presented in section 4.3.2.

5.4.1 Practical Results for Timed Sequential Asynchronous Update (TSAU)

Figures 5.34, 5.35, 5.36 show the error, profile plot and node time values plot both against the gateway time of the 4, 9 and 16 node networks respectively. We observe from these figures that, the dip phenomenon is exhibited by all network configurations for Timed Sequential Asynchronous Update (TSAU) with all networks registering, average minimum error in $\mathbb{E}_{min}^{dip}$ in the 10^{-4} range. For the 4, 9 and 16 node networks, we observe $\mathbb{E}_{min}^{dip}$ to be 4.17×10^{-4}, 3.57×10^{-4} and 2.89×10^{-4} respectively. Hence the synchronization accuracy in the dip region can be said to improve with increased network size/density. We also observe k_{min}^{dip} of 13, 48 and 109 for the 4, 9 and 16 node networks respectively. Hence the number of iterations needed to reach the dip region increases with increased network size/density.

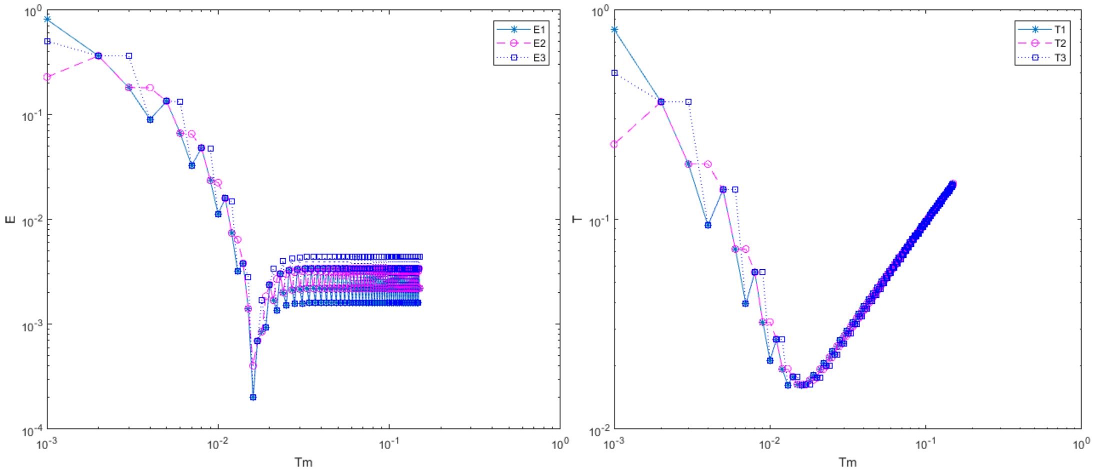


Figure 5.34: Practical Node Time and Error Profiles for 4 Grid Topology

Finally for $\mathbb{V}_{k_{min}^{dip}}$, we observe values of 0, 0.27 and 0.17 for the 4, 9 and 16 node

Table 5.10: 4 Node Grid Network Dip Region Results Comparison

Topologies	**Grid**	
Nodes	Iteration	Error
N1	13	6.21E-04
N2	13	3.10E-04
N3	13	3.10E-04
Statistics		
Mean	13	4.14E-04
Minimum	13	3.10E-04
Maximum	13	6.21E-04
Variance	0.00	3.21E-08

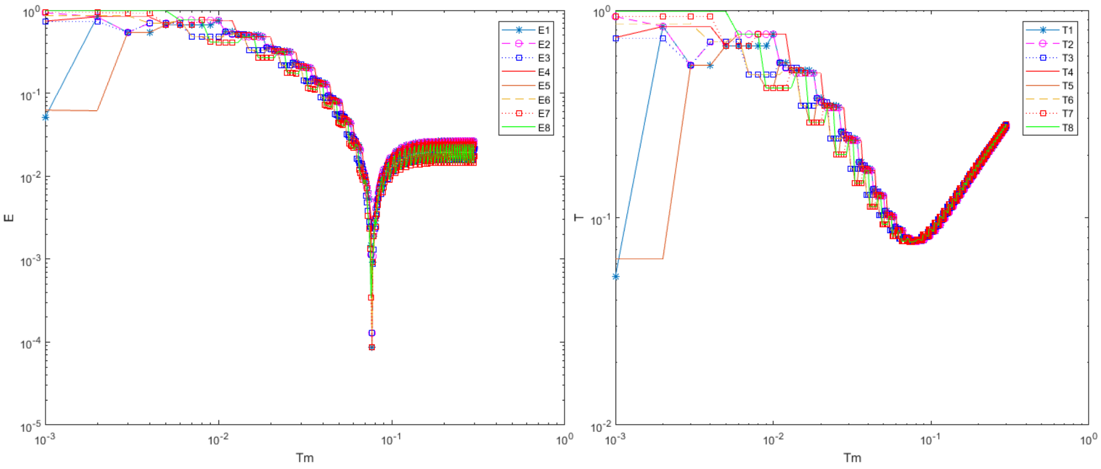


Figure 5.35: Practical Node Time and Error Profiles for 9 Grid Topology

networks respectively. Hence the variance of the number of iterations needed to reach the dip region is below 0.3 for all networks. This is an advantageous feature for this method if employed for local time update in the proposed protocol. The summary of the dip region results for the 4, 9 and 16 node networks are given in Tables 5.10, 5.11 and 5.12 respectively.

Table 5.11: 9 Node Grid Network Dip Region Results Comparison

Topology	**Grid**	
Nodes	Iteration	Error
N1	49	3.25E-04
N2	49	5.57E-04
N3	48	3.27E-04
N4	49	5.57E-04
N5	48	3.27E-04
N6	48	2.18E-04
N7	48	3.27E-04
N8	48	2.18E-04
Statistics		
Mean	48	3.57E-04
Minimum	48	2.18E-04
Maximum	49	5.57E-04
Variance	0.27	1.74E-08

5.4.2 Practical Results for Unidirectional Asynchronous Flooding (UAF)

Figures 5.37, 5.38, 5.39 show the error, profile plot and node time values plot both against the gateway time of the 4, 9 and 16 node networks respectively. We observe from these figures that, the dip phenomenon is exhibited by all network configurations for Unidirectional Asynchronous Flooding (UAF) with all networks registering, average minimum error in $\mathbb{E}_{min}^{dip}$ in the 10^{-4} range.

For the 4, 9 and 16 node networks, we observe $\mathbb{E}_{min}^{dip}$ to be 3.44×10^{-4}, 2.69×10^{-4} and 2.89×10^{-4} respectively. Hence the synchronization accuracy in the dip region can be said to improve with increased network size/density. We also observe k_{min}^{dip} of 18, 88 and 109 for the 4, 9 and 16 node networks respectively. Hence the number of iterations needed to reach the dip region increases with increased

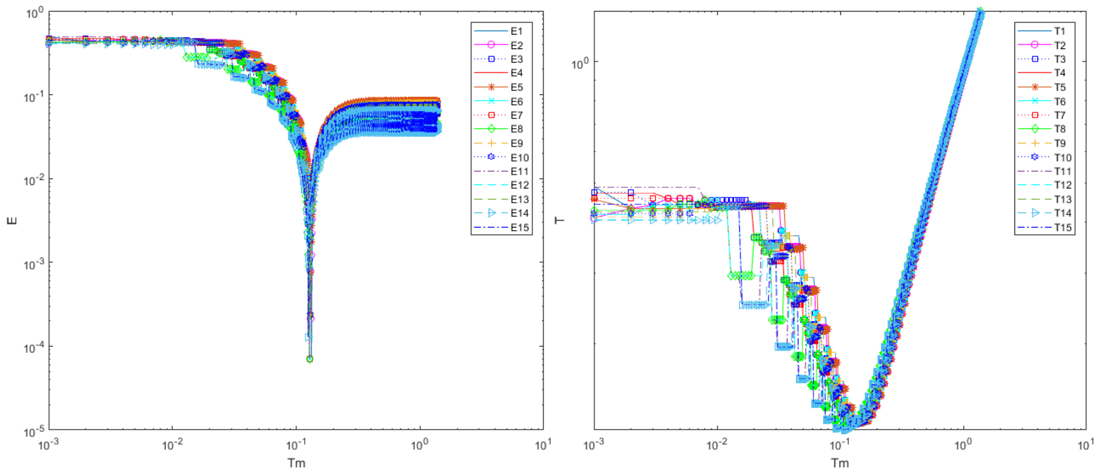


Figure 5.36: Practical Node Time and Error Profiles for 16 Grid Topology

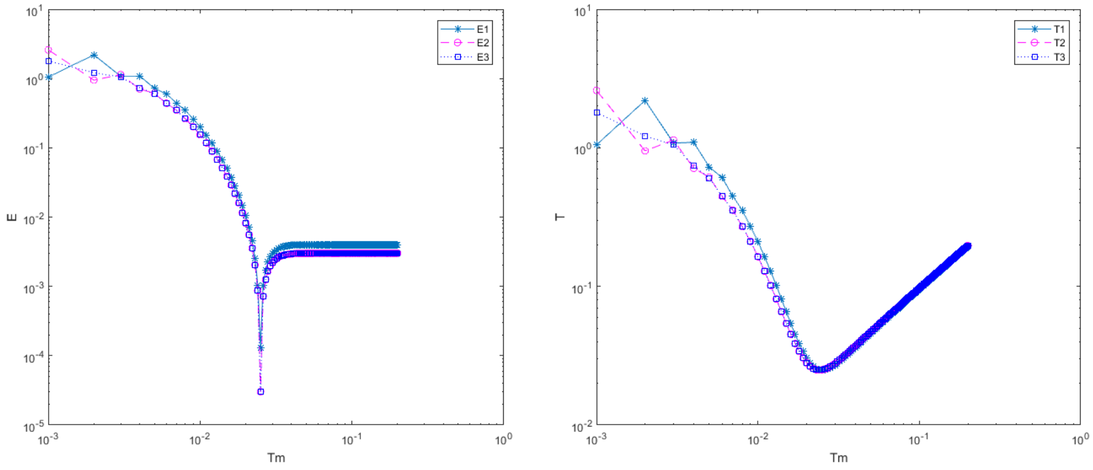


Figure 5.37: Practical Node Time and Error Profiles for 4 Grid Topology

network size/density. Finally for $\mathbb{V}_{k_{min}^{dip}}$, we observe values of 0, 0 and 0.24 for the 4, 9 and 16 node networks respectively. Hence the variance in the number of iterations needed to reach the dip region is below 0 for all networks except the 16 node network. This is similar to the simulation which gives this protocol a an advantageous feature for implementation in the proposed protocol [13]. The summary of the dip region results for the 4, 9 and 16 node networks are given in

---

[13] Refer Figure 3.1, section 3.2

Table 5.12: 16 Node Grid Network Dip Region Results Comparison

Topology	**Grid**	
Nodes	Iteration	Error
N1	109	6.08E-04
N2	109	3.19E-04
N3	109	5.69E-05
N4	109	2.99E-04
N5	109	3.19E-04
N6	109	2.91E-05
N7	109	3.16E-04
N8	109	4.53E-04
N9	109	5.69E-05
N10	109	3.16E-04
N11	108	3.04E-04
N12	108	2.55E-04
N13	109	2.99E-04
N14	109	4.53E-04
N15	108	2.55E-04
Statistics		
Mean	109	2.89E-04
Minimum	108	2.91E-05
Maximum	109	6.08E-04
Variance	0.17	2.41E-08

Tables 5.13, 5.14 and 5.15 respectively.

5.4.3 Practical Results for Bidirectional Asynchronous Flooding (BAF)

Figures 5.40, 5.41, 5.42 show the error, profile plot and node time values plot both against the gateway time of the 4, 9 and 16 node networks respectively. We observe from these figures that, the dip phenomenon is exhibited by all network configurations for Bidirectional Asynchronous Flooding (BAF) with all networks registering, average minimum error in $\mathbb{E}_{min}^{dip}$ in the 10^{-4} range.

Table 5.13: 4 Node Grid Network Dip Region Results Comparison

Topologies	**Grid**	
Nodes	Iteration	Error
N1	18	4.31E-04
N2	18	3.01E-04
N3	18	3.01E-04
	Statistics	
Mean	18	3.44E-04
Minimum	18	3.01E-04
Maximum	18	4.31E-04
Variance	0.00	5.60E-09

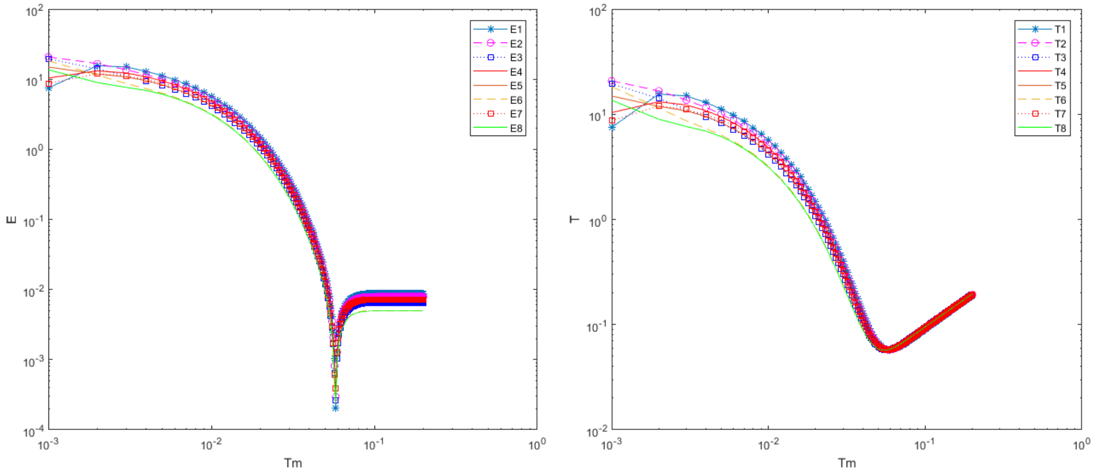


Figure 5.38: Practical Node Time and Error Profiles for 9 Grid Topology

For the 4, 9 and 16 node networks, we observe $\mathbb{E}_{min}^{dip}$ to be 4.1×10^{-4}, 2.2×10^{-4} and 1.14×10^{-4} respectively. Hence the synchronization accuracy in the dip region can be said to improve with increased network size/density. We also observe k_{min}^{dip} of 13, 50 and 96 for the 4, 9 and 16 node networks respectively. Hence the number of iterations needed to reach the dip region increases with increased network size/density. Finally for $\mathbb{V}_{k_{min}^{dip}}$, we observe values of 0, 2.79 and 13.11 for the 4, 9 and 16 node networks respectively. Hence the variance in the number of iterations needed to reach the dip region is high compared to the other methods

Table 5.14: 9 Node Grid Network Dip Region Results Comparison

Topology	**Grid**	
Nodes	Iteration	Error
N1	58	2.07E-04
N2	58	2.95E-04
N3	58	2.61E-04
N4	58	2.82E-04
N5	58	2.47E-04
N6	58	2.26E-04
N7	58	3.92E-04
N8	58	2.38E-04
Statistics		
Mean	58	2.69E-04
Minimum	58	2.07E-04
Maximum	58	3.92E-04
Variance	0.00	3.29E-09

Table 5.15: 16 Node Grid Network Dip Region Results Comparison

Topology	**Grid**	
Nodes	Iteration	Error
N1	105	4.05E-04
N2	105	4.05E-04
N3	105	4.29E-04
N4	105	4.62E-04
N5	105	4.05E-04
N6	105	4.12E-04
N7	105	4.50E-04
N8	104	3.69E-04
N9	105	4.29E-04
N10	105	4.50E-04
N11	104	3.07E-04
N12	104	1.35E-04
N13	105	4.62E-04
N14	104	3.69E-04
N15	104	1.35E-04
Statistics		
Mean	105	3.75E-04
Minimum	104	1.35E-04
Maximum	105	4.62E-04
Variance	0.24	1.11E-08

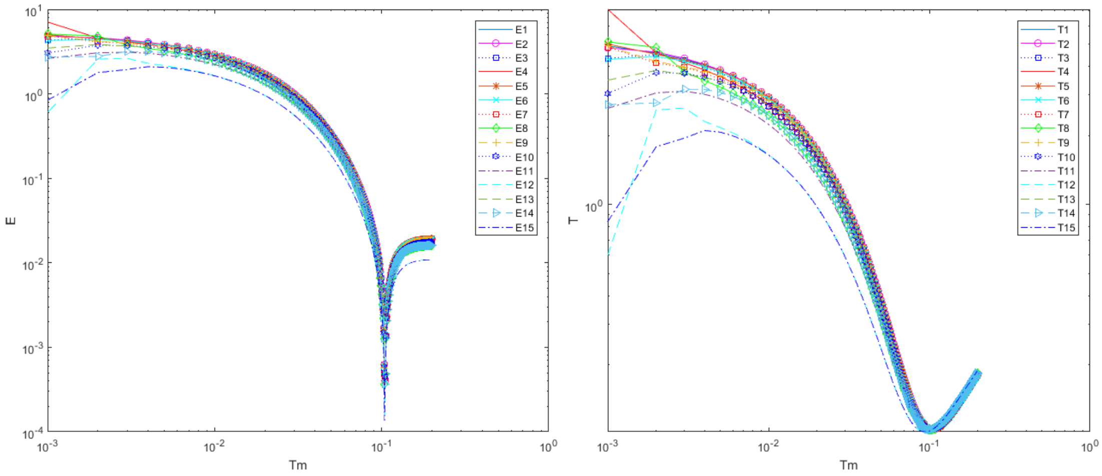


Figure 5.39: Practical Node Time and Error Profiles for 16 Grid Topology

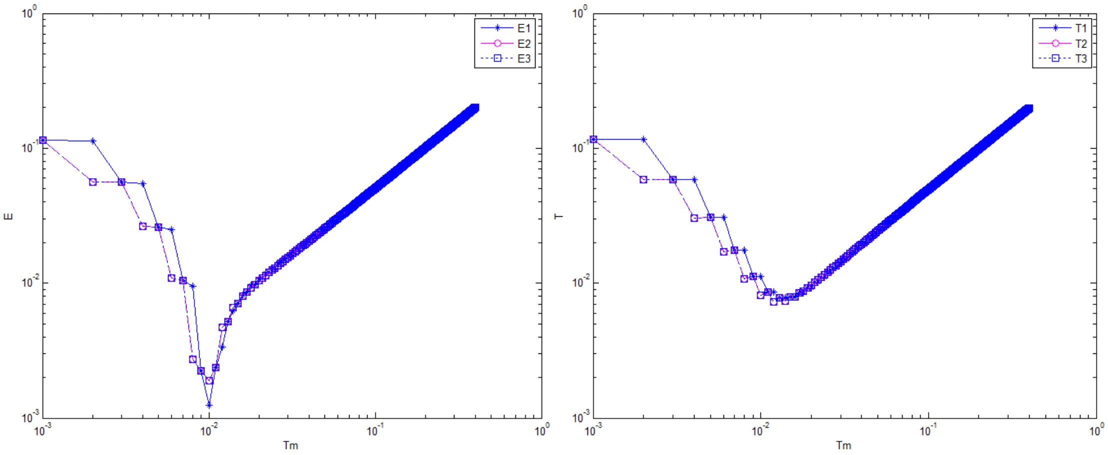


Figure 5.40: Practical Node Time and Error Profiles for 4 Grid Topology

for all networks except the 4 node network where it is zero. The summary of the critical parameters dip region results for the 4, 9 and 16 node networks are given in Tables 5.16, 5.17 and 5.18 respectively.

Table 5.16: 4 Node Grid Network Dip Region Results Comparison

Topologies	**Grid**	
Nodes	Iteration	Error
N1	13	6.21E-04
N2	13	3.10E-04
N3	13	3.10E-04
Statistics		
Mean	13	4.14E-04
Minimum	13	3.10E-04
Maximum	13	6.21E-04
Variance	0.00	3.21E-08

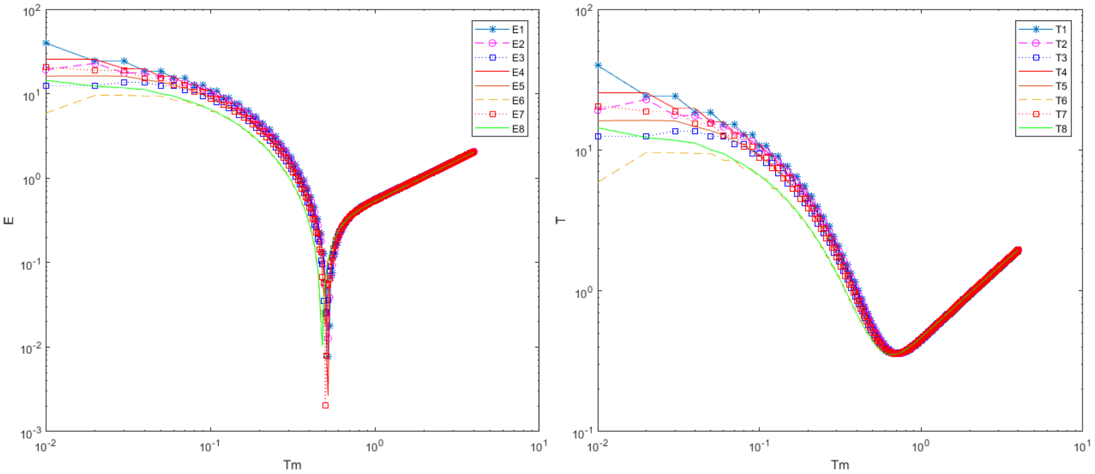


Figure 5.41: Practical Node Time and Error Profiles for 9 Grid Topology

5.4.4 Practical Comparison between Proposed Methods of Local Time Update

5.4.4.1 Minimum Error in Dip Region, $\mathbb{E}_{min}^{dip}$

In Figure 5.43, we compare the synchronization accuracy in the dip region for TSAU, UAF, BAF and STU. From the results, we observe that STU registers the lowest error as compared to all the other methods for the 4 node network. This is followed bu UAF, whereas BAF and TSAU have almost the same performance.

Table 5.17: 9 Node Grid Network Dip Region Results Comparison

Topology	**Grid**	
Nodes	Iteration	Error
N1	52	7.70E-03
N2	52	1.27E-03
N3	50	2.52E-03
N4	52	7.27E-04
N5	50	1.77E-03
N6	48	1.93E-03
N7	50	6.20E-04
N8	48	1.03E-03
Statistics		
Mean	50	2.20E-03
Minimum	48	6.20E-04
Maximum	52	7.70E-03
Variance	2.79	5.36E-06

For the 9 and 16 node networks, BAF registers the highest error. In the network of 9 nodes, the accuracy of UAF in the dip region is nearly the lowest followed by STU then TSAU.

TSAU is observed to have a lowest error than UAF and STU for the 16 node network. We also observe that, for TSAU, synchronization accuracy increases as the network gets bigger. This pattern is however not observed for the other asynchronous methods by is inverse for STU. In general, we can say conclude here that UAF showed the best accuracy in terms of synchronization in the dip region for the asynchronous schemes. This is followed by TSAU. BAF was observed to have the least synchronization accuracy especially for the large networks.

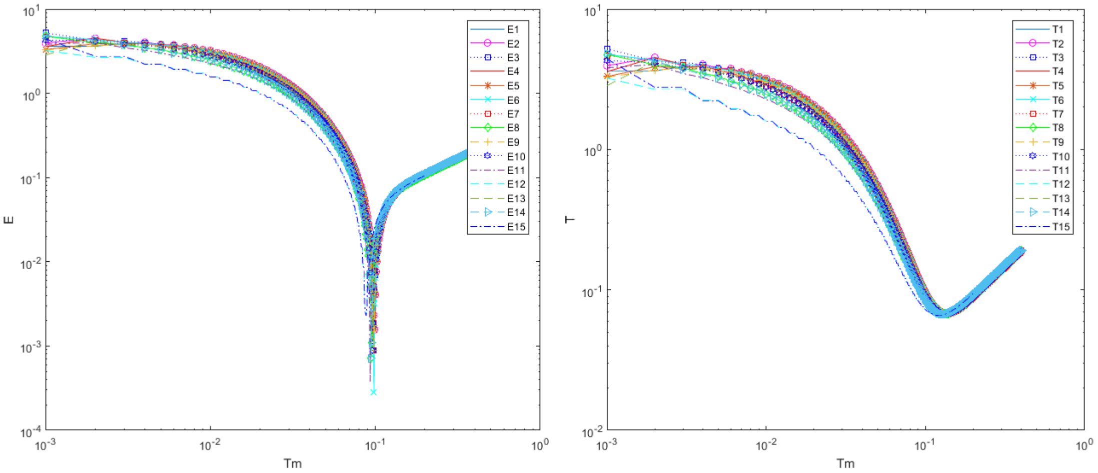


Figure 5.42: Practical Node Time and Error Profiles for 16 Grid Topology

5.4.4.2 Communication Cycles to Reach Dip Region, k_{min}^{dip}

In Figure 5.44, we compare the number of iterations required to reach minimum error in the dip region for TSAU, UAF, BAF and STU for the practical experiments. From the results, we observe that TSAU and BAF required the least number of iterations to reach the dip region for the 4 node network, followed by STU and the UAF. For the 9 nodes network, BAF and TSAU required the least number of iterations to reach the dip region followed by STU then UAF. For the 16 node network we observe a clear pattern of performance. STU registers the least value of k_{min}^{dip}, this is followed by BAF then UAF then TSAU. In general k_{min}^{dip} increases as the number of network nodes increase.

In general, we can infer here that, for the number of iterations needed to reach the minimum error in the dip region, STU performs best, followed by BAF then TSAU. UAF was observed to have the least performance as shown in Figure 5.44 although it outperforms TSAU in the 16 node network. From the operation of

Table 5.18: 16 Node Grid Network Dip Region Results Comparison

Topology	**Grid**	
Nodes	Iteration	Error
N1	99	1.57E-03
N2	99	1.57E-03
N3	98	8.87E-04
N4	97	1.09E-03
N5	99	1.57E-03
N6	98	2.83E-04
N7	96	8.93E-04
N8	94	7.12E-04
N9	98	8.87E-04
N10	96	8.93E-04
N11	93	3.81E-04
N12	88	2.28E-03
N13	97	1.09E-03
N14	94	7.12E-04
N15	88	2.28E-03
Statistics		
Mean	96	1.14E-03
Minimum	88	2.83E-04
Maximum	99	2.28E-03
Variance	13.11	3.61E-07

STU [14], all nodes update at each communication instant whereas in TSAU [15], UAF [16] and BAF[17] only a subset of the nodes update, at each communication instant the clocks or time values of nodes not updating in the asynchronous methods still progress or tick. This is because the clock of a node is still active even if the node sleeps [18]. Therefore it is fair to take an iteration in all the methods to be the same.

---

[14] Refer Algorithm 2, section 4.3.7

[15] Refer Algorithm 3, section 5.2.1

[16] Refer Algorithm 4, section 5.2.2

[17] Refer Algorithm 5, section 5.2.3

[18] A detailed exposition on clocks is given in section 2.2

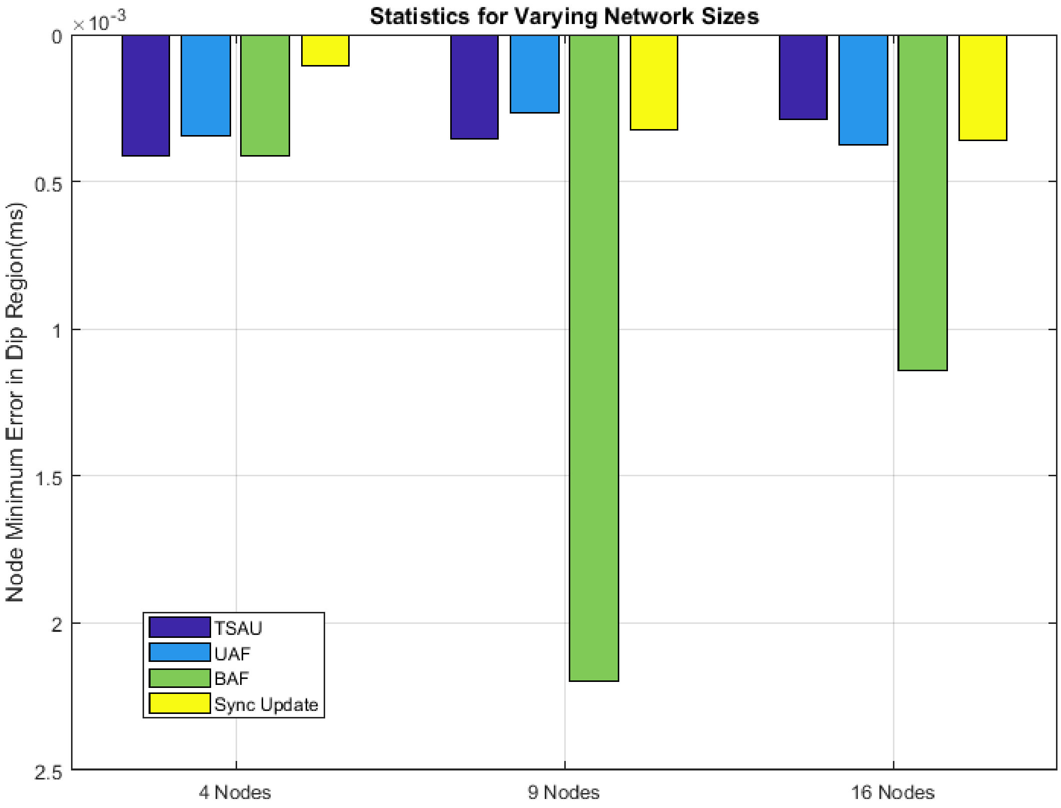


Figure 5.43: Practical Comparison of Minimum Errors, $\mathbb{E}_{min}^{dip}$ for TSAU, UAF and BAF For Varying Network Sizes

5.4.4.3 Variance in Communication Cycles to Reach Dip Region, $\mathbb{V}_{k_{min}^{dip}}$

In Figure 5.45, we compare the variances in the number of communication cycles required to reach minimum error in the dip region for TSAU, UAF, BAF and STU for our practical experiments on the grid networks. First we observe that all methods had null variances for the small network except for STU. For the large networks, we observe that UAF performs best by having zero variance for 4 and 9 node networks. This is a very crucial since all nodes can stop update, sleep, wakeup and activate at the same time. This unique feature in the practical results gives UAF a clear advantage over all the other methods. TSAP has the next best

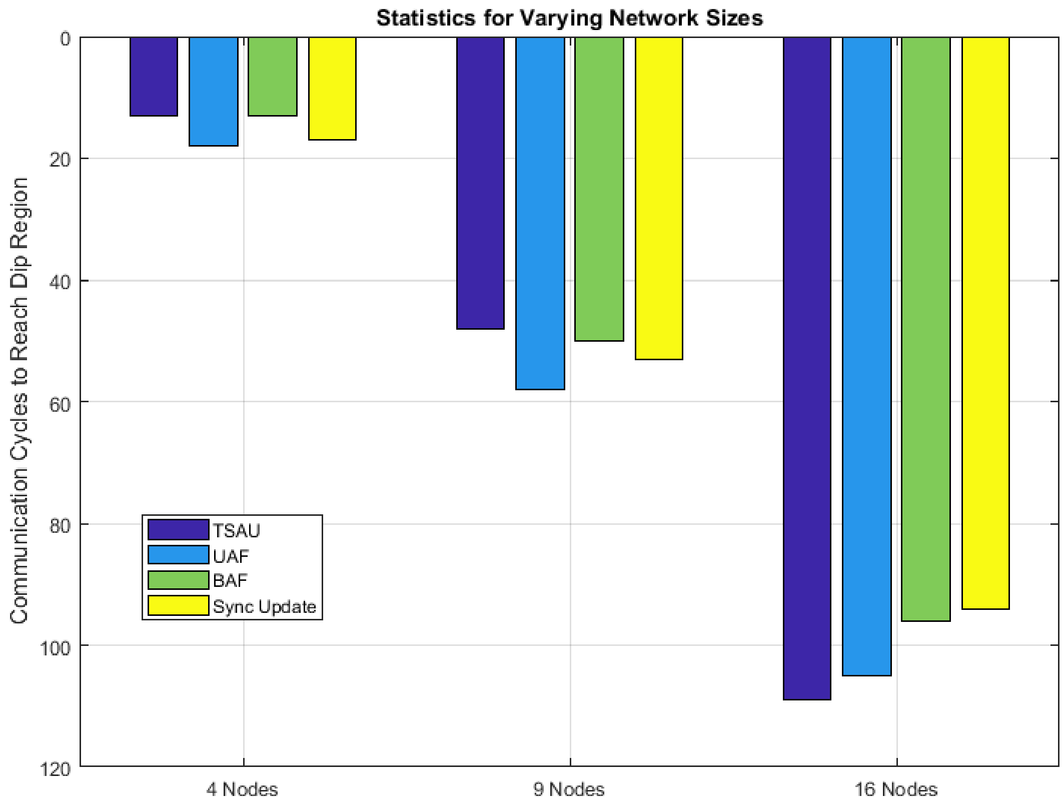


Figure 5.44: Practical Comparison of Communication Cycles, k_{min}^{dip} for TSAU, UAF and BAF For Varying Network Sizes

performance for the 9 and 16 node networks, followed by STU. The performance of BAF on the large networks is observed to be the least for this metric.

5.5 Simulation on Network with Stochastic Link Connectivity

To test the error profile created by the averaging procedure, in particular the transient dip in the error profile [19], the test network in Figure 5.46 is simulated on a network with random connectivity links. Node 1 is selected as the gateway

---

[19] Refer [51] Section II, C

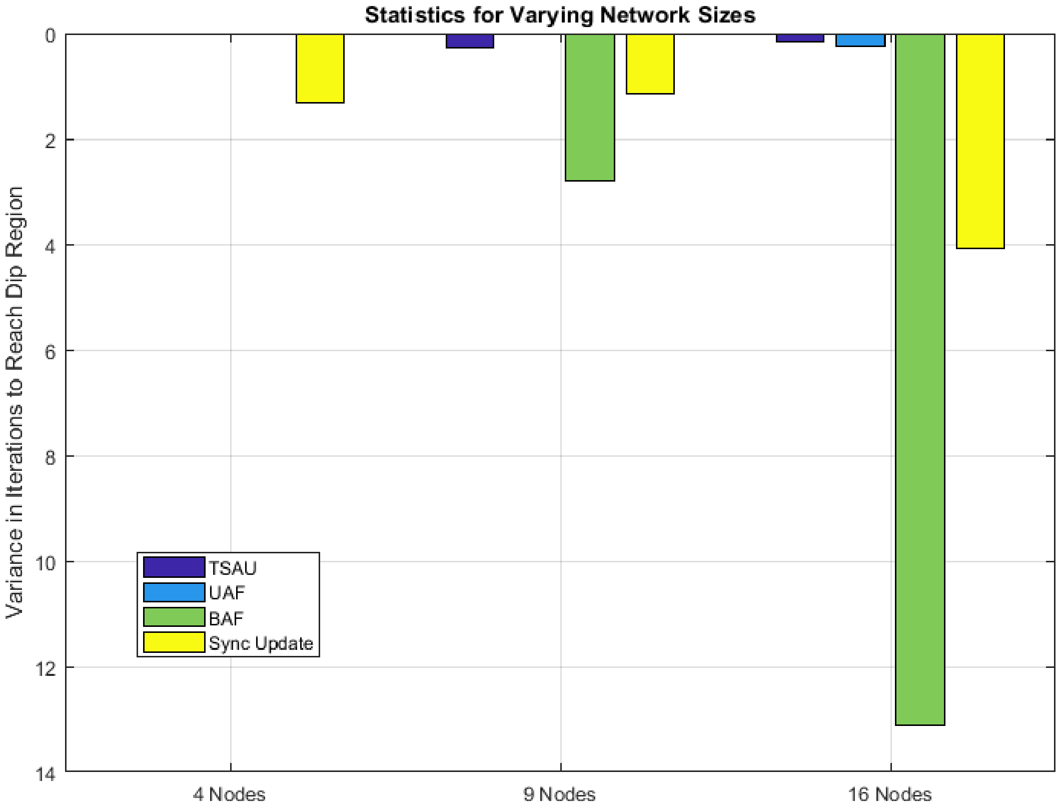


Figure 5.45: Practical Comparison of Variances in Communication Cycles, $\mathbb{V}_{k^{dip}_{min}}$ for TSAU, UAF and BAF For Varying Network Sizes

node and a communication rate of 1 KHz is used. A connectivity function, $f(\mathrm{C}_{ij})$ among nodes is modeled as a Bernoulli random variable as shown in Figure 5.46 with a discrete probability distribution function given by:

$$f(\mathrm{C}_{ij}) = \begin{cases} p & \text{if } \mathrm{C}_{ij} = 1 \\ 1 - p & \text{if } \mathrm{C}_{ij} = 0 \end{cases} \tag{5.1}$$

Where p is the probability that the channel is on and the nodes are communicating and C_{ij} is the channel connectivity binary variable which $'1'$ if the channel between nodes i and j is active and functional and $'0'$ otherwise.

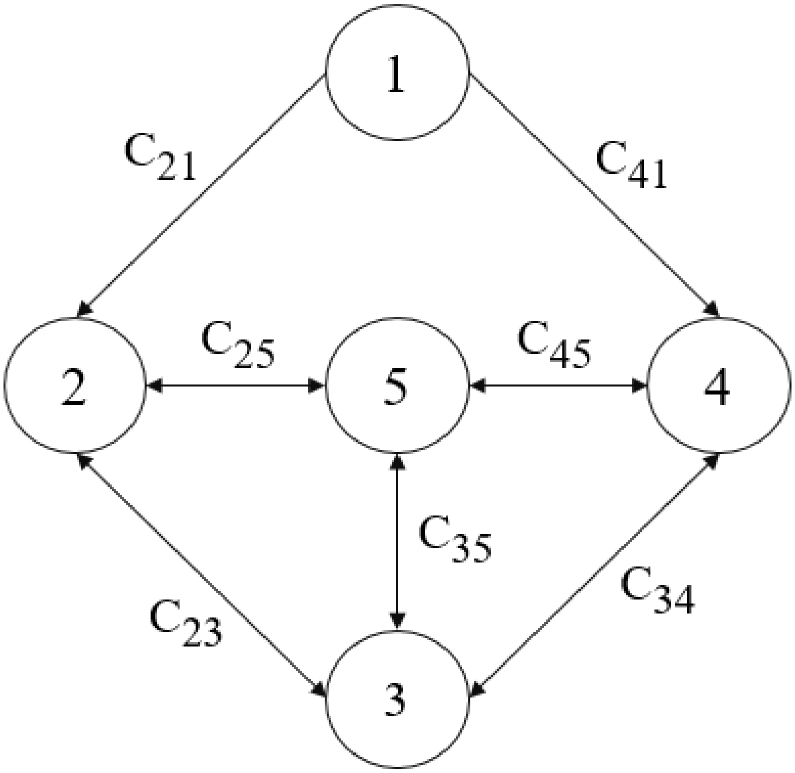


Figure 5.46: Network for Channel Availability Test

For each method, we present the error profile for each node taken at four values for p: $p = 1$ where the links are available 100% of the time, $p = 0.75$ and $p = 0.5$ and 0.25 where communication links malfunction 25%, 50% and 75% of the time, respectively. The error profile represents a graph of the error between the evolving time series of each node and the gateway node.

Figures 5.47, 5.48 and 5.49 show the error profiles for TSAU, UAF and BAF respectively. We observe that, the dip characteristic persists in each profile at all levels of connectivity in both methods. As expected, the error profile becomes less well defined as the probability of links being active decreases. It is observed that, for both all methods, error values range between 10^{-3} and 10^{-4}. Further, we observe that number of iterations required to reach the dip region is higher

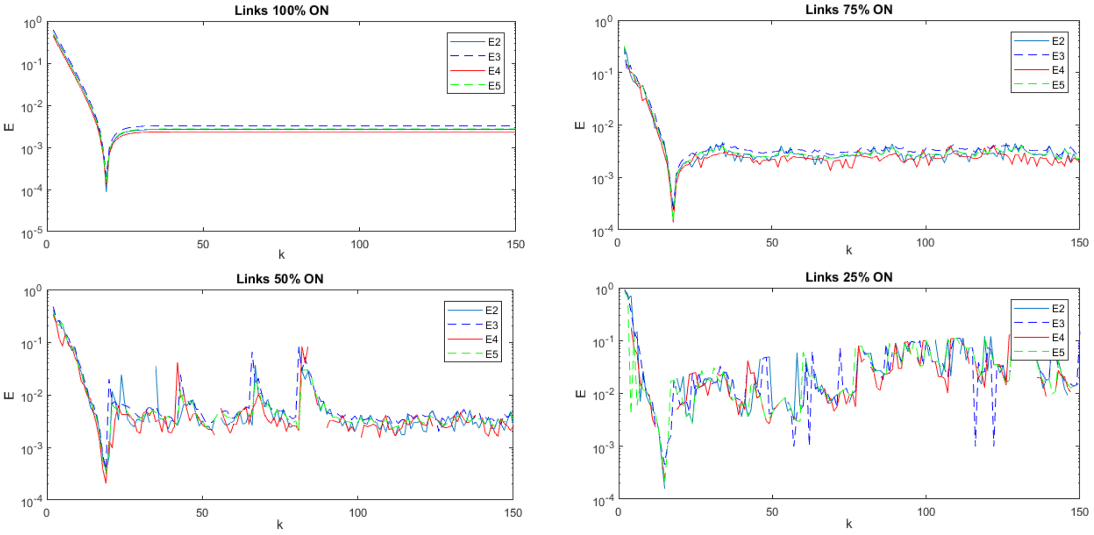


Figure 5.47: Node Error profile for the Test Network for Different Probabilities of Channel Availability for TSAU

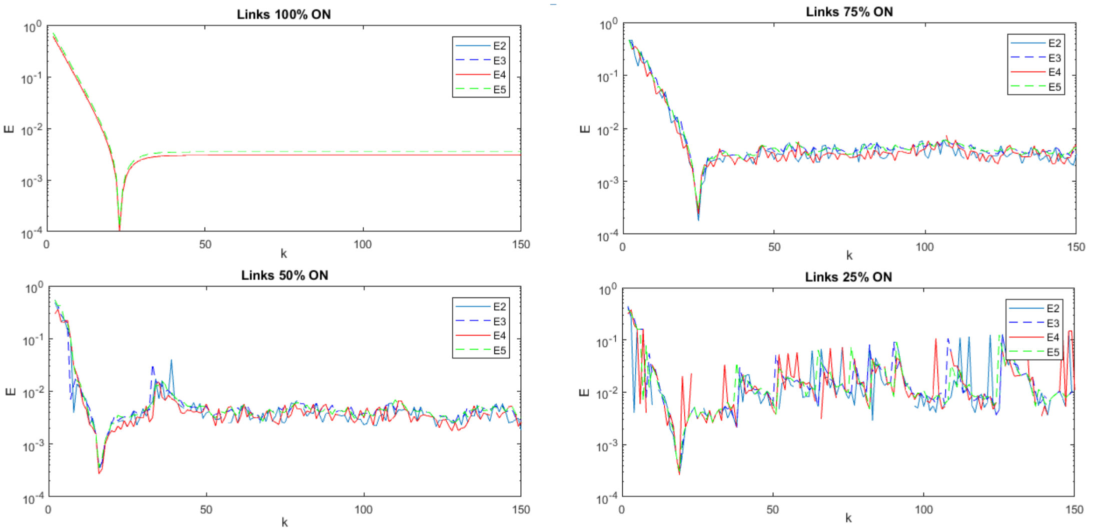


Figure 5.48: Node Error profile for the Test Network for Different Probabilities of Channel Availability for UAF

in UAF as compared to TSAU with BAF registering lowest errors for all nodes but the nodes are more tightly synchronized in UAF and TSAU as compared to BAF. Also for all methods, it is observed that, the number of iterations required to reach the dip error remains nearly the same irrespective of link availability.

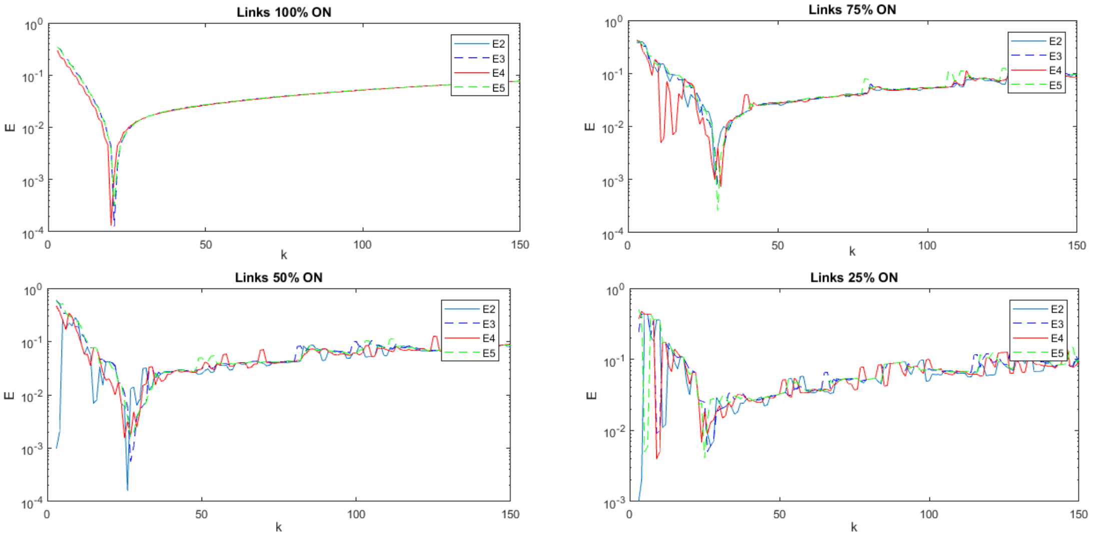


Figure 5.49: Node Error profile for the Test Network for Different Probabilities of Channel Availability for BAF

5.6 Summary

In this chapter, three methods based on asynchronous consensus were proposed for local node time update. The first method named *Timed Sequential Asynchronous Update (TSAU)* carries out time updates asynchronously through sequential activation of nodes one at a time using proximity to the gateway node. The second method, *Unidirectional Asynchronous Flooding (UAF)* improves the design of TSAU by allowing nodes at approximately same proximity to the gateway node to update at the same communication instant. Node activation in UAF is coordinated by the gateway node. The third method dubbed *Bidirectional Asynchronous Flooding (BAF)* removes the need for the gateway coordination of activation through an adaptive bidirectional wakeup cycle.

Simulation and practical experiments of each method was then conducted for 4, 9 and 16 networks configured in the grid, random and hexagonal topologies.

The results for all the suggested methods and the *Synchronous Time Update (STU)* method are compared using the three key parameters of evaluation: the average minimum error in dip region, $\mathbb{E}_{min}^{dip}$, the number of communication cycles to reach dip region, k_{min}^{dip} and the variance in the number of communication cycles required to reach minimum error in the dip region, $\mathbb{V}_{k_{min}^{dip}}$. For the simulation results, we observed that, in general, UAF showed the best accuracy in terms of synchronization in the dip region followed by STU then TSAU. BAF was observed to have the least synchronization accuracy especially for the large networks. A similar performance pattern was observed for the practical results.

We also inferred from our simulations that, for the number of iterations needed to reach the minimum error in the dip region, BAF generally performs best, followed by TSAU then UAF. For the practical results, STU was observed performs best, followed by BAF then TSAU and UAF had the least performance. For the last parameter, $\mathbb{V}_{k_{min}^{dip}}$, UAF outperformed all the other methods for all networks. TSAP has the next best performance for the 9 and 16 node networks, followed by STU. The performance of BAF on the large networks was the least for this metric. Similar observations were made for the practical experimental results.

The comparative analyses carried out in this chapter are done between the developed local time update methods for the suggested time synchronization protocol and the parameters considered are specific to it. To ascertain the performance of our suggested schemes on real time networks, an extensive experimental evaluation is required to compare these schemes to some reported relevant synchronization protocols in the literature. This task is carried out in the next chapter, where we adopt two distributed and two centralized reported protocols for the comparative evaluation.

Simulation sensitivity test was carried out for all three methods on a network with stochastic connectivity. The dip feature of the protocol was observed to persist for all methods with link failures up to 75%.

CHAPTER 6

EXPERIMENTAL PERFORMANCE EVALUATION: COMPARISON WITH OTHER PROTOCOLS

6.1 Introduction

This chapter focuses on a comparative experimental evaluation of the suggested asynchronous local time update schemes: TSAU, UAF and BAF with two fully distributed protocols: Energy Efficient Gradient Time Synchronization protocol (EGTSP) and Average Proportional-Integral Synchronization protocol (AvgPISync) and two centralized flooding-based protocols: Flooding Time Synchronization Protocol(FTSP) and Flooding Proportional-Integral Synchronization protocol (FloodPISync) [1]. The experimental framework and methodology adopted for TSAU, UAF and BAF are in section 4.3.2. FTSP and FloodPISync are *flooding-based* protocols where time synchronization is done by synchronizing all network nodes to a reference node. In FTSP [58] the estimation of the reference node clock is done using linear regression. In FloodPISync [20], clock synchronization is done by adjusting the drifts and offsets of each network node using the proportional-integral controller analogy so as to estimate the clock speed and offset of a reference or root node. AvPISync [20] is a distributed protocol where the speed of node logical clocks are adjusted using PI controllers. In this protocol, neighborhood and hence global synchronization is achieved through average consensus, which is also used in our proposed methods. EGTSP [45] is a localized algorithm that achieves a global time consensus and gradient time property using effective drift compensation and incremental averaging estimation. Detailed review of these protocols are given in chapter 2. To compare the proposed methods with these reported protocols in the literature, the main parameters considered for evaluation are the global and local synchronization errors [45, 48, 58], the memory required to efficiently execute and store the core heuristics and parameters for each protocol [32, 65], and the energy consumed [40, 56] by a node to fully run the protocol.

---

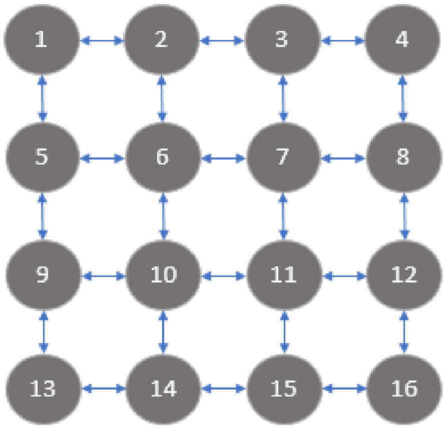
[1] The source codes for AvgPISync, EGTSP, FTSP and FloodPISync are publicly available in the GitHub for TinyOS applications repositories

6.1.1 Test Networks

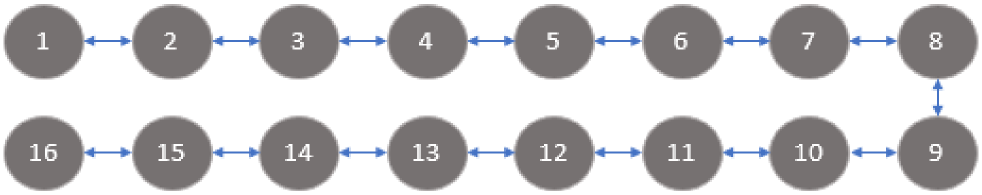
In our experiments, we utilize a line and grid topology of 16 nodes shown in Figure 6.1. Experiments on line topologies allows for the scalability of protocols to be tested since the performance of flooding based time synchronization protocols degrade with increase in network diameter [38] whereas experiments on the grid topology gives an indication of the performance of time synchronization protocols when subjected to some adverse network conditions like contention, congestion and increased packet collisions [32, 66]. It should be noted here that, in order to realize these network configurations for our experiments, network nodes are forced to accept packets from only some nodes. The testbed setup of the nodes which entails the way each network operates for our experiments for TSAU, UAF and BAF is discussed in section 4.3.2.1.

6.1.2 Experiment Parameters

In our experiments, a beacon period, B of 30 seconds was used for all protocols. The least-squares regression table used for FTSP and EGTSP is kept at a capacity of 8 elements as used for the comparative work in [39, 58]. The experimental parameters for FloodPISync and AvgPISync, β, α_{max} and e_{max} are set respectively at 1, 3.33×10^{-8} and 6000 *ticks* as used for comparisons in [20]. Since it takes a number of pooling cycles for FTSP to elect the root node, the protocol is modified to have a fixed root node. Node '1' is used as the reference node for the FTSP and FloodPISync experiments and used as the gateway node for TSAU, UAF



(a) Grid Topology



(b) Line Topology

Figure 6.1: Distribution of Sensor Nodes for Practical Experiments

and BAF. A TMicro timer ticking periodically at an interval of 1000 [2] is used for the gateway node in the experiments for TSAU, UAF and BAF. When each experiment commences, network nodes are switched on randomly within 2 minutes of operation with each experiment taking about 3.8 hours.

6.1.3 Energy Consumption and Memory Requirements

In this section we present an evaluation of the performances of EGTSP, FTSP, AvgPISync, FloodPISync, TSAU, UAF and BAF in terms energy consumption and memory requirements. For a time synchronization protocol to operate efficiently, it's operation must conserve resources which are deemed scarce for WSN

---

[2] i.e., Δ of 1 ms

nodes. The Random Access Memory (RAM) is the memory used to store node information obtained from the network during the execution of the protocol and therefore determines the main memory requirements of the protocol [3]. This resource must therefore be conserved especially in distributed protocols where the retention of neighborhood clock information is necessary for effective operation. For each node to store the main protocol application source code and the auxiliary codes and interfaces required to carryout synchronization, the Read-Only Memory is needed, which is also a limited resource and must be conserved. The core heuristics and main applications of the protocol must be effective to synchronize all network nodes but simple enough to fit on the ROM[4]. Since most wireless sensor nodes perform other tasks, like routing, clustering, congestion control and traffic management, the time synchronization protocol should occupy only a small fraction of the available ROM space. Apart from conserving the RAM and ROM, each protocol must also conserve the energy consumption of network nodes. Crucial for energy consumption is the number of communication cycles, the CPU overhead [5] and the length of synchronization messages. For the number of communication cycles, the higher the frequency at which nodes communicate, the higher the amount of information needed to be exchanged to achieve synchronization and the more energy is expended in the transmission and reception of synchronization messages. Also the higher the CPU overhead, the higher the

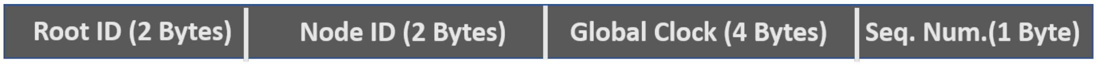
---

[3] The Memsic MicaZ nodes are equipped with 4kB RAM

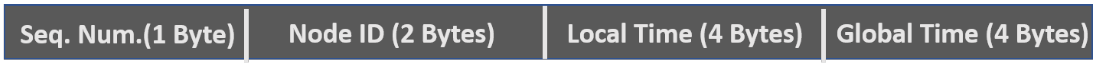
[4] Each Memsic MicaZ node is equipped with 128kB program flash memory

[5] The CPU overhead is the time in *ticks* taken for a recently received synchronization message to be processed and used to update the clock of a node. It should be noted that 1 $tick = 1\ \mu s$.

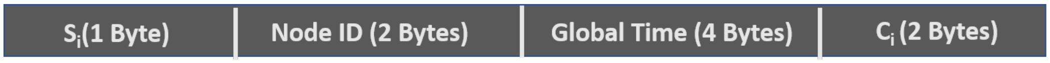
energy needed to process and update the time synchronization information. The length of the message is also a crucial determinant in the time taken to transmit, receive and process synchronization messages. In Table 6.1, we present the CPU overhead, the message length and the main memory (RAM) and flash momory (ROM) requirements for EGTSP, FTSP, AvgPISync, FloodPISync, TSAU, UAF and BAF.



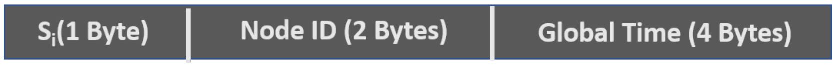
(a) FTSP and FloodPISync Payload Fields



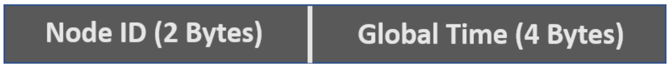
(b) EGTSP Payload Fields



(c) BAF Payload Fields



(d) UAF Payload Fields



(e) AvgPISync and TSAU Payload Fields

Figure 6.2: TimeSync Packet Payload Field Descriptions for FTSP, FloodPISync, EGTSP, AvgPISync, TSAU, UAF and BAF

In the operation of FTSP, when a new synchronization packet is received, FTSP is designed to store the current clock information in an 8 element regression table and conduct least square regression computations involving several float point divisions and multiplications [37], hence is observed to have the high CPU overhead of 5440 *ticks*. EGTSP carries out all these tasks and but also computes the averages of the clock rate multiplies and offsets. This involves the storage of neighborhood time values and the averaging involves arithmetic operations and floating points. EGTSP has the highest CPU overhead of approximately 5610 *ticks*. For AvgPISync and FloodPISync, the protocols are lightweight and carry out few additions and multiplications with almost no float point divisions and are therefore observed to each consume a significantly reduced computation time or CPU overhead of 145 *ticks*. TSAU carries out an averaging calculation in each cycles and also maintains the parameter, $UpdateTime$ which regulates the sequential activation. TSAU has a CPU overhead of 133. For UAF, the mathematical computations involves just the additions and floating point division used in calculating the average and hence has the lowest CPU overhead of 133 *ticks*, whereas in BAF the CPU overhead increases by 29 *ticks* due to the added logical operations involving the counter variable, c_i [6].

Table 6.1: Memory Requirements, CPU Overhead, and Synchronization Message Length of FTSP, FloodPISync, EGTSP, AvgPISync, TSAU, UAF and BAF

Protocols	**Centralized**		**Distributed**		**Proposed**		
	FTSP	FloodPISync	EGTSP	AvgPISync	TSAU	UAF	BAF
CPU Overhead (ticks)	$\simeq$5440	$\simeq$145	$\simeq$5610	$\simeq$145	$\simeq$141	$\simeq$133	$\simeq$162
Message Length (bytes)	9	9	11	6	6	7	9
Main Memory (bytes)	52	16	$64\vert\mathcal{N}\vert + 12$	16	12	12	14
Flash Memory (bytes)	17974	17974	20738	16166	13510	13320	12282

As shown in Figure 6.2 (a), for FTSP and FloodPISync, each synchronization payload is composed of 2 *bytes* reference or root node ID, 2 *bytes* node ID of the particular node sending the packet, 4 *bytes* current clock information and 1 *byte*

---

[6] c_i is the variable included to ensure automatic regulation of the asynchronous bidirectional cycling (Refer section 5.2.3, Algorithm 5, lines 9 and 11)

sequence number, making a total message length of 9 *bytes* as shown in Table 6.1. For EGTSP, each payload is composed of 1 *byte* sequence number, 2 *bytes* node ID of the particular node sending the packet, 4 *bytes* local time and 4 *bytes* local time, making the total message length of 11 *bytes* as shown in Figure 6.2 (b). For TSAU and AvgPISync each payload has a length of 6 *bytes* composed of node ID of the particular node sending the packet and 4 *bytes* global time as shown in Figure 6.2 (e). For UAF, each synchronization payload shown in Figure 6.2 (d) is composed of 2 *bytes* node ID of the particular node sending the packet, 4 *bytes* current clock information and 1 *bytes* for the binary-status bit variable, s_i, making a total message length of 7 *bytes*. The added counter variable, c_i is the only difference between BAF and UAF packets and hence BAF has a total message length of 9 *bytes* similar to FTSP and FloodPISync. Based on the clock model presented by Equation 2.2 in [20] which is employed in the source code of all these protocols, each protocol is required to store a hardware clock of 4 *bytes*, a logical clock of 4 *bytes* and a rate multiplier of 4 *bytes*. In addition to this, FTSP allocates 40 *bytes* for its regression table [37] whereas AvgPISync and FloodPISync allocates 4 *bytes* for storing an error parameter [20]. EGTSP also requires an extra 64 bytes of memory allocated to keep track of each neighbouring node, which is an extremely big memory overhead. Hence EGTSP requires a total of $(64|\mathcal{N}|+12) bytes$, where $|\mathcal{N}|$ denotes the maximum neighborhood cardinality. For TSAU, UAF and BAF, no memory allocation is required for the storage of neighboring node(s) or gateway node clocks, but 2 *bytes* is allocated for the counter variable, c_i in BAF. Therefore as shown in Table 6.1, the main memory (RAM) required for the execution and operations of FTSP, FloodPISync, EGTSP, AvgPISync, TSAU, UAF and BAF are 52 *bytes*, 16 *bytes*, $(64|\mathcal{N}|+12)bytes$, 16 *bytes*,12 *bytes*, 12 *bytes* and 14 *bytes* respectively. Hence both TSAU, UAF and BAF outperforms all the other reported protocols in terms of memory requirements. For the flash memory (ROM), FTSP requires 17947 *bytes*, FloodPISync requires 16352 *bytes*, EGTSP requires 20738 *bytes*,AvgPISync requires 16166 *bytes*, TSAU requires 13510 *bytes*, UAF requires 13320 *bytes* and BAF requires 12282 *bytes*.

Table 6.2: Current Consumption by MicaZ Nodes [1]

Operation	**MicaZ**
Minimum Voltage	2.7 V
Standby	27.0 μA
MCU Idle	3.2 mA
MCU Active	8.0 mA
MCU + Radio RX	23.3 mA
MCU + Radio TX (0 dBm)	21.0 mA

In order to calculate the total energy consumed, we need to combine the total required energy for transmission, reception and processing. For the Memsic MicaZ nodes employed in this work, the minimum voltage, v_{min} required, the current consumed in transmission, i_{TX}, the current consumed in reception, i_{RX} and the current consumed by the microcontroller in active mode, i_{MCU} are given in Table 6.2. Given a default 11 *bytes* header and 7 *bytes* footer for TinyOS packets, the total packet length, L of FTSP, FloodPISync and BAF is 27 *bytes* each, that of AvgPISync and TSAU is 24 *bytes* each, that of EGTSP is 29 *bytes* and that of

UAF, 25 *bytes*. Given a CPU overhead, c and a data rate, R of 250 kbps [7], the total energy, E_T require for each protocol can given as;

$$E_T = c \times i_{MCU} \times v_{min} + \frac{L}{R} \times i_{TX} \times v_{min} + \frac{L}{R} \times i_{RX} \times v_{min} \qquad (6.1)$$

Based on Equation 6.1, the energy required for FTSP, FloodPISync, EGTSP, AvgPISync, TSAU, UAF and BAF are given respectively as 130.4 μJ, 16.1 μJ, 135 μJ, 14.6 μJ, 14.53 μJ, 14.8 μJ and 16.4 μJ. Hence we observe that TSAU consumes least power but has nearly a similar consumption with UAF followed by the PISync protocols then BAF. EGTSP consumes the highest power among all the protocols. We see here that our proposed protocols consume about 9 times less than FTSP and EGTSP but approximately similar consumption as the PISync protocols.

6.1.4 Local and Global Synchronization Errors

To evaluate the synchronization accuracy for each protocol, the differences in clock values of network nodes has to be observed. We accomplish this by collecting all nodes' clock values at each communication instant, k. Network-wide synchronization error at each communication instant is then calculated using the *maximum global synchronization error*, E^G_{max} and the *average global synchronization error*, E^G_{avg} which gives a measure of the global synchronization accuracy for all nodes given by Equations 6.2 and 6.4 respectively [20]. Further, we observe the syn-

---

[7] Chipcon CC2420 radio chip provides a 250 kbps data rate at 2.4 GHz frequency.

chronization error of nodes with respect to their neighbors by using the *maximum local synchronization error*, E^L_{max} and the *average local synchronization error*, E^L_{avg} which given respectively by Equations 6.3 and 6.5 [39]. An important note here is that, since our suggested synchronization ceases update at the dip region, it is fair to compare the above described parameters with the global and local errors in the dip region for our developed method. The time taken for the network to synchronize to a global time, i.e. convergence time is also compare. For TSAU, UAF and BAF, the convergence time is the same as the number of communication cycles needed to reach the dip region depicted by the parameter, k^{dip}_{min}.

$$E^G_{max} = \max_{i,j \in V}(t_i(k) - t_j(k)) \tag{6.2}$$

$$E^L_{max} = \max_{i \in V, j \in E}(t_i(k) - t_j(k)) \tag{6.3}$$

$$E^G_{avg} = \frac{1}{N} \sum_{i \in V} \max_{i,j \in V}(t_i(k) - t_j(k)) \tag{6.4}$$

$$E^L_{avg} = \frac{1}{N} \sum_{i \in V} \max_{i \in V, j \in E}(t_i(k) - t_j(k)) \tag{6.5}$$

Where V and E are the global network vertex set and the neighborhood graph vertex set respectively and N is the number of nodes in the network.

6.1.4.1 Comparison: Synchronization Error for Grid Topology

In this section we compare the performance of each protocol in terms of local and global errors for the grid topology. The maximum and average, local and global errors are presented for FTSP, FloodPISync, EGTSP, AvgPISync, TSAU, UAF and BAF by Figures 6.8, 6.5, 6.9, 6.6, 6.4, 6.3 and 6.7 respectively. The critical values for these metrics including the convergence time are summarized in Table 6.3. First we observe that, the global and local errors curve for TSAU, UAF and BAF show a more smooth characteristic as compared to AvgPISync, EGTSP, FTSP and FloodPISync.

This might stem from the fact that in our methods of time update, clock estimation is not done by an independent estimation of clock skew rate and offset like in these other protocols but is done by directly using the global logical time values. Although individual estimation of skew rate and offset is widely reported in a myriad of synchronization protocol, for our method of synchronization, the skew rate and offset would still converge to a consensus value so far as for any network node, i, there exist a spanning path from it to the gateway node.

Secondly, we observe the *dip* characteristics [8] of the algorithm in all the global and local errors. With a well defined minimum error our proposed method are expected to locate the transient and stop update. This is a huge advantage for our proposed synchronization protocol because unlike the other protocols where nodes are continuously, communicating and estimating the global time through drift and offset compensation, nodes running on TSAU, UAF and BAF only need to identify the transient dip, stop updates and sleep to conserve a lot of energy.

From the results we observe that, the global and local errors for BAF far out perform all the other protocols as shown in Table 6.3. This is followed by TSAU then UAF. We further observe from Table 6.3 that, FTSP outperforms

---

[8] Refer section 3.2.3

FloodPISync on the grid topology in terms all the error registering an average local error on 4.1 μs as compared to 6.2μs for FloodPISync. We also observed that, EGTSP performed worst on the grid network with a maximum global error as high as 42 μs as compared to the highest for our method, maximum global error for UAF at 4 μs. Hence given the same network, after convergence is achieve in the dip region, at worst it will take about 10 time the same time for our proposed method to drift to the same level as EGTSP on a grid network. The protocol with a near performance to our schemes is AvgPISync which has error values about 1.5 time that of UAF.

Table 6.3: Global and Local Error Comparison for Grid Topology

Global and Local Error Comparison for Grid Topology							
Protocols	**Centralized**		**Distributed**		**Proposed**		
	FTSP	FloodPISync	EGTSP	AvgPISync	TSAU	UAF	BAF
Max. Global(μs)	8.0	13.0	42.0	6.0	0.73	4.00	0.42
Avg. Global(μs)	6.9	11.6	37.8	5.0	0.47	3.80	0.20
Max. Local(μs)	8.0	13.0	42.0	5.0	0.53	1.50	0.23
Avg. Local(μs)	4.1	6.2	18.4	2.9	0.35	0.47	0.14
Conv. Time(s)	900	750	1000	1200	400	155	230

On the convergence time, TSAU, UAF and BAF also out performed all the other protocols. We observe that, the centralized protocols outperformed the distributed protocols on the grid networks. This might be due to the time take to exchange neighborhood information and estimate global time, whereas for the centralized protocols nodes just pass on the clock information of the root node for estimation. Our proposed methods are all decentralized and communication is done only with 1-hop neighbors therefore a lower convergence time is observed.

Since for TSAU only one node updates at a time, we observe that it takes about twice more time for the network to converge as compared to UAF and BAF as shown in Table 6.3.

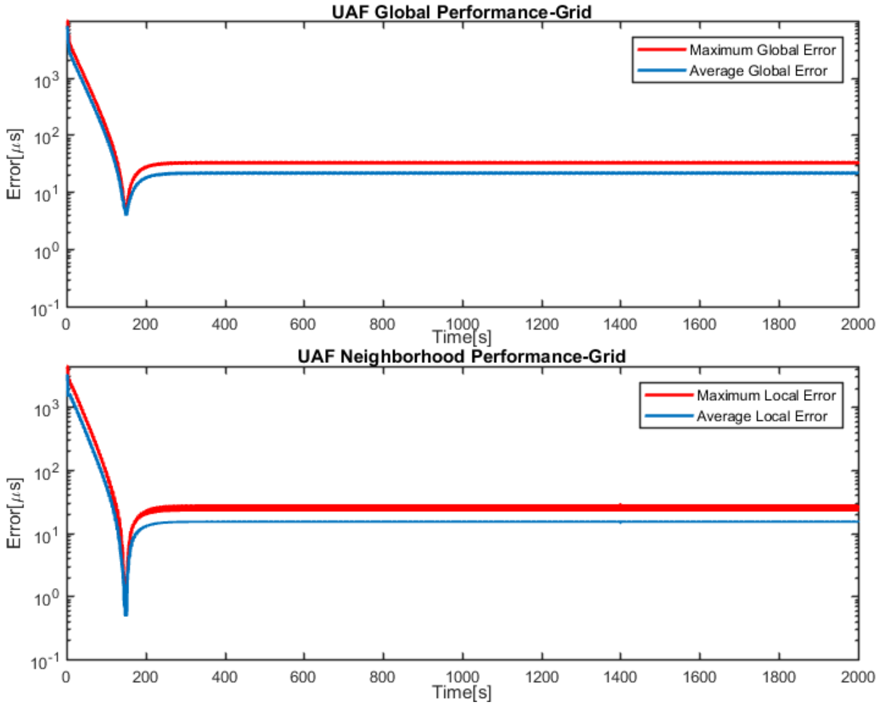


Figure 6.3: UAF Neighborhood and Global Synchronization Error for Grid Topology

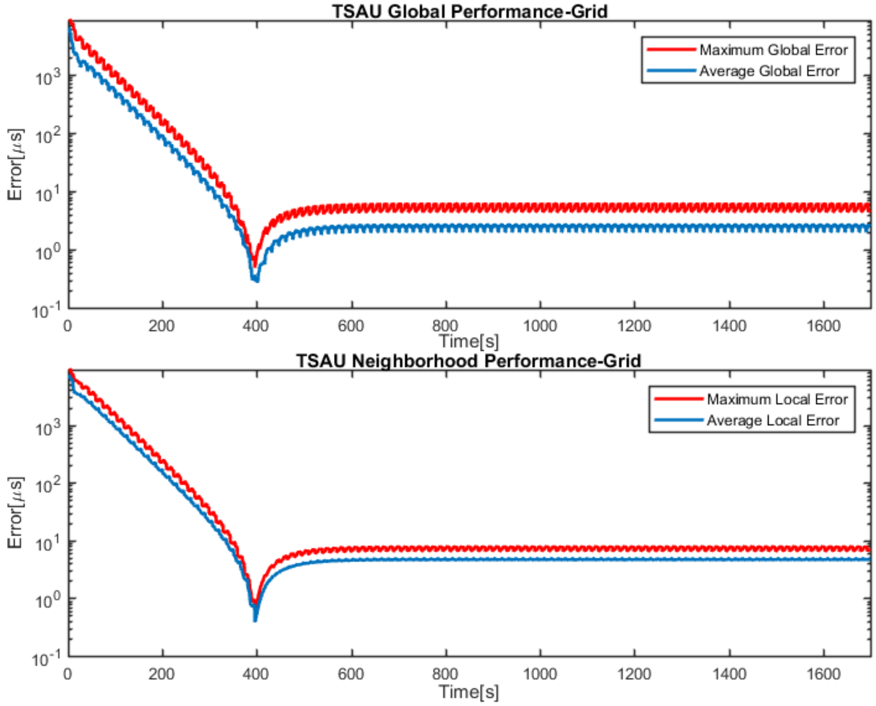


Figure 6.4: TSAU Neighborhood and Global Synchronization Error for Grid Topology

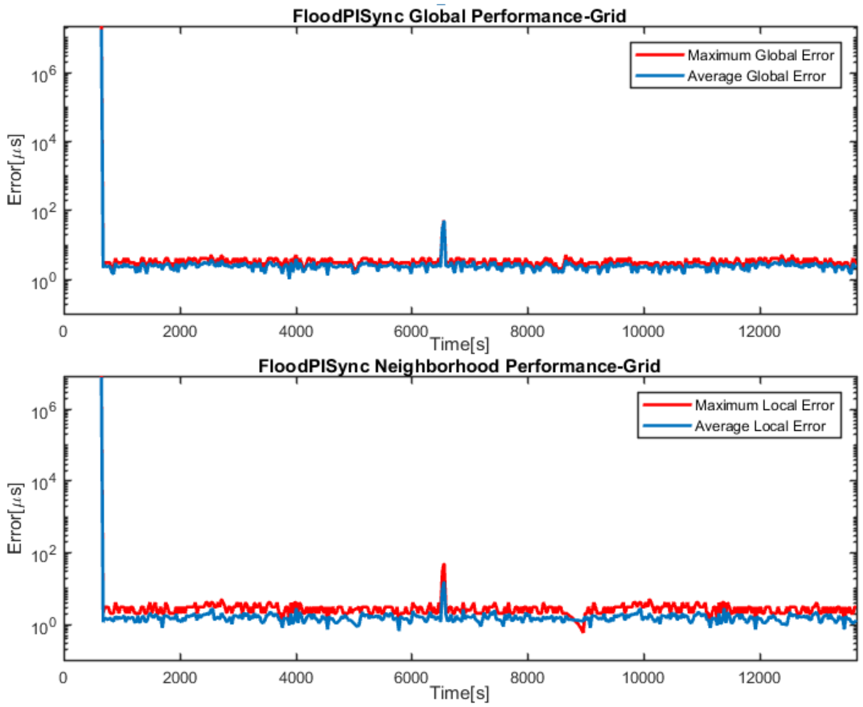


Figure 6.5: FloodPISync Neighborhood and Global Synchronization Error for Grid Topology

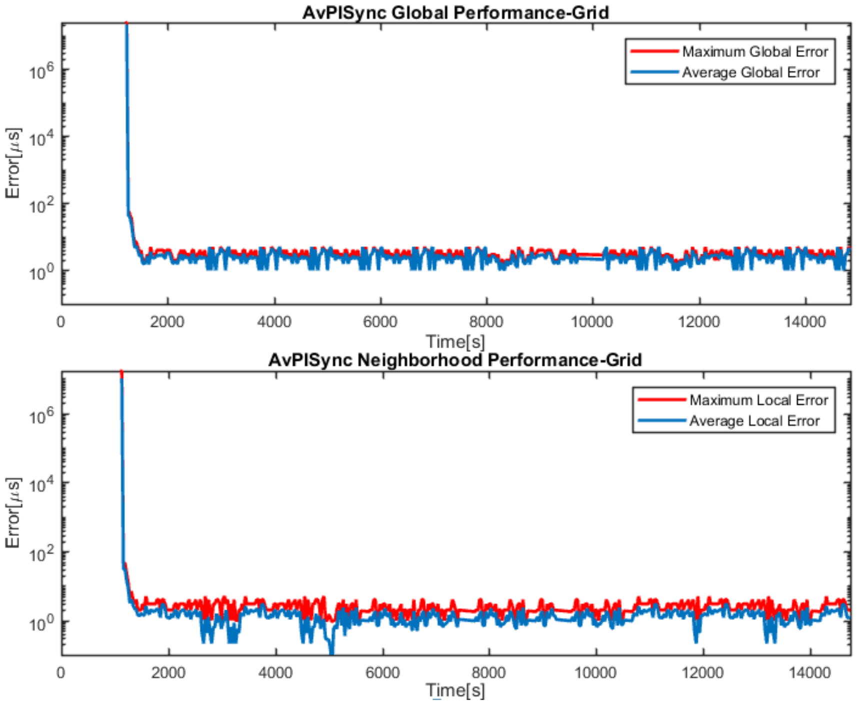


Figure 6.6: AvgPISync Neighborhood and Global Synchronization Error for Grid Topology

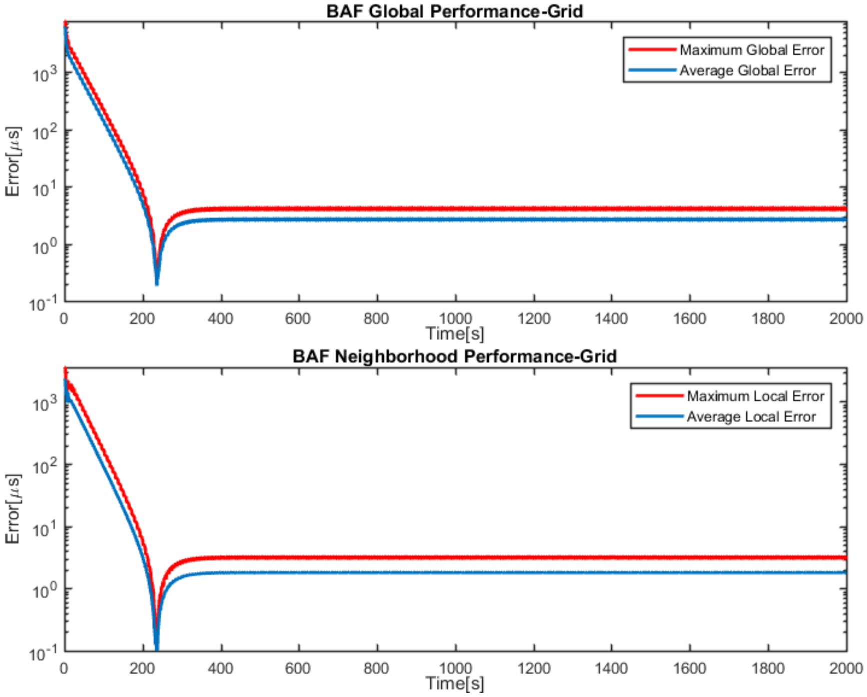


Figure 6.7: BAF Neighborhood and Global Synchronization Error for Grid Topology

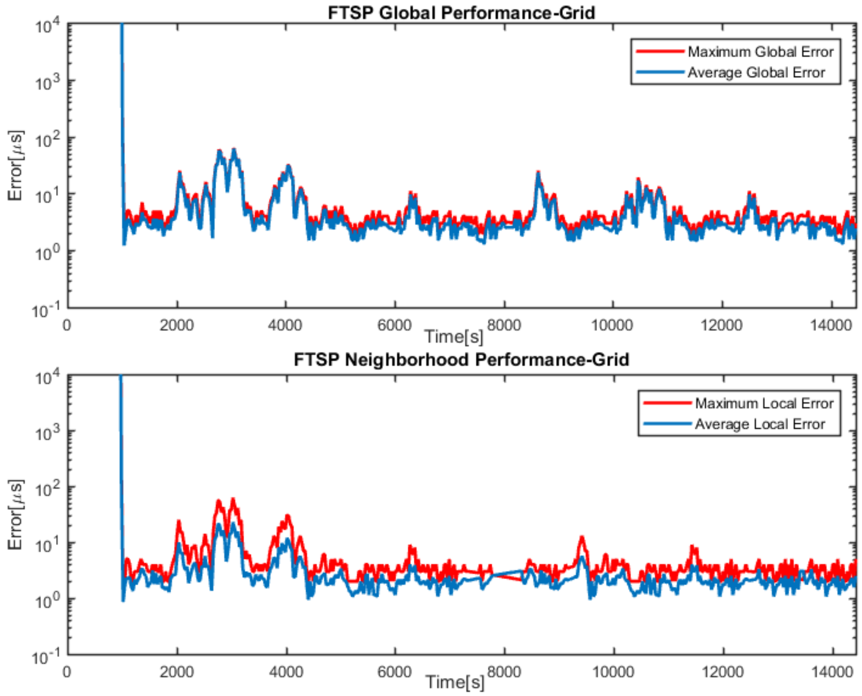


Figure 6.8: FTSP Neighborhood and Global Synchronization Error for Grid Topology

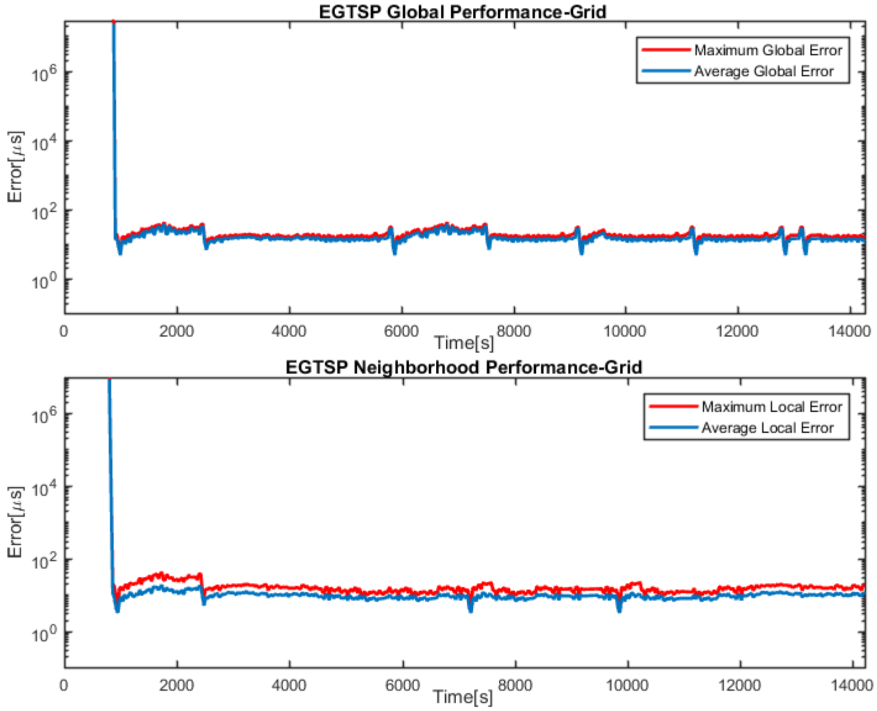


Figure 6.9: EGTSP Neighborhood and Global Synchronization Error for Grid Topology

6.1.4.2 Comparison: Synchronization Error for Line Topology

The results for global and local performances of the protocols in a line topology are presented in this section. The maximum and average, local and global errors are presented for FTSP, FloodPISync, EGTSP, AvgPISync, TSAU, UAF and BAF by Figures 6.16, 6.12, 6.14, 6.13, 6.10, 6.11 and 6.15 respectively and Table 6.4 summarizes these results including the convergence time.

For TSAU, UAF and BAF, the characteristics of the error curves are observed to be similar to those of the grid topology but almost all the global and local errors are have surprisingly lower values as compared to those in the grid topology. For TSAU we observed all global and local error values to be below 0.7 μs as compared the grid topology where values go as high as $4 \mu s$. This is understandable because TSAU operate one node at a time so a structured line networks fit its operation as compared to the grid topology. But TSAU still recorded the higher error values as compared to UAF and BAF. BAF still performed best in terms of local and global error followed by UAF. This pattern is also identical to the comparative results in chapter 5.

For FTSP, we observe a graph with high overshoots and undershoots as compared to the grid error curves.Also we note that, PBAF has the lowest local and global skews in the steady state region. But as opposed to the results in the grid topology, UAF out performs both FloodPISync and FTSP for the grid topology. EGTSP has the worst performance with error values in the hundreds. All the other protocols also registers in the tens compared to values less than one in TSAU, UAF and PBAF. Hence, once the stopping criterion is able to stop node updates in the dip region, our proposed protocol will outperformed all these protocols by a huge margin.

Table 6.4: Global and Local Error Comparison for Line Topology

Global and Local Error Comparison for Line Topology							
Protocols	**Centralized**		**Distributed**		**Proposed**		
	FTSP	FloodPISync	EGTSP	AvgPISync	TSAU	UAF	BAF
Max. Global(μs)	63.0	51.0	519.0	50.0	0.66	1.01	0.34
Avg. Global(μs)	61.5	50.0	418.8	50.0	0.30	2.80	0.18
Max. Local(μs)	62.0	50.0	225.0	50.9	0.57	0.77	0.21
Avg. Local(μs)	22.7	15.4	195.3	29.8	0.38	0.36	0.10
Conv. Time(s)	≃2150	≃550	≃1700	≃1750	≃400	≃150	≃250

In the line topology, we see an even larger convergence time for all protocols except FloodPISync. Also TSAU, UAF and BAF have nearly similar convergence times as in the grid topology but we observe that BAF has an increased convergence time from 230μs in the grid topology to 250 μs in the line network. It is also observed here that, FTSP has about more than 5 time convergence time as compared to our developed methods. FloodPISync still registered the lowest convergence time for the reported protocols whereas EGTSP and AvgPISync have a fairly high convergence time. Hence we can infer that amongst our protocols UAF performs best in terms of convergence time with is matches the results in chapter 5.

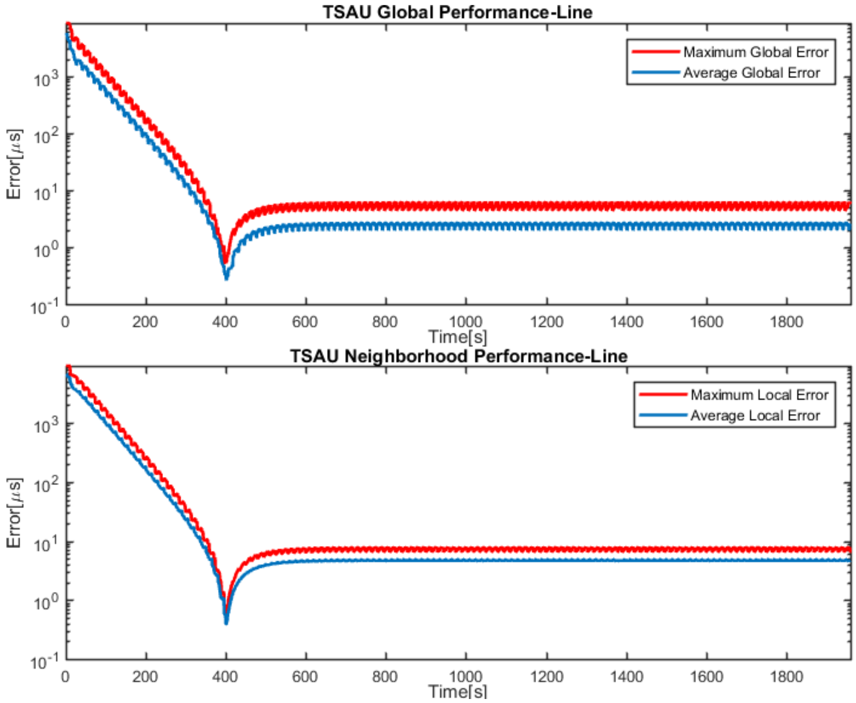


Figure 6.10: TSAU Neighborhood and Global Synchronization Error for Line Topology

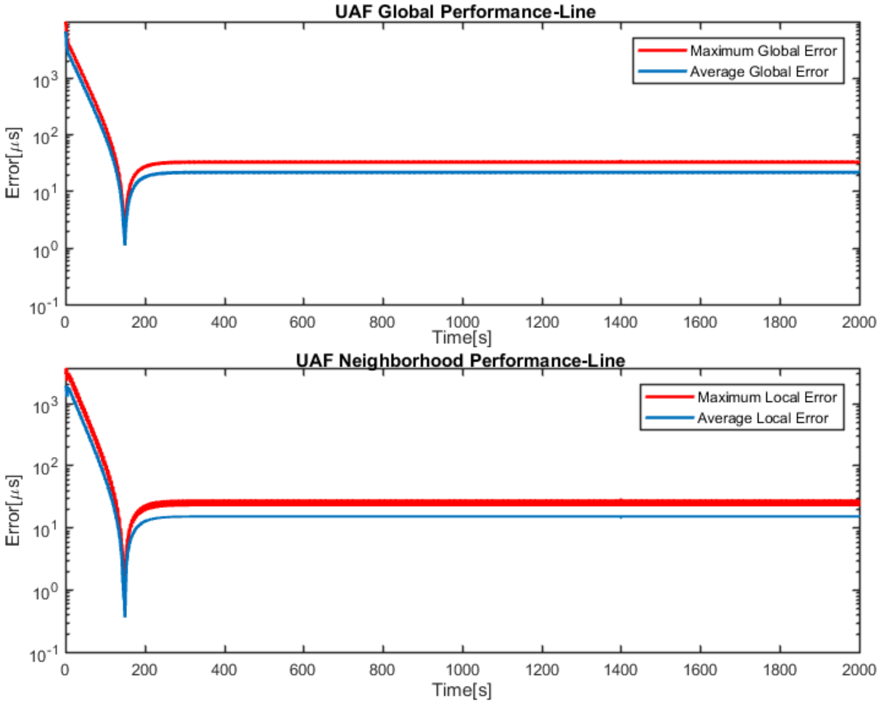


Figure 6.11: UAF Neighborhood and Global Synchronization Error for Line Topology

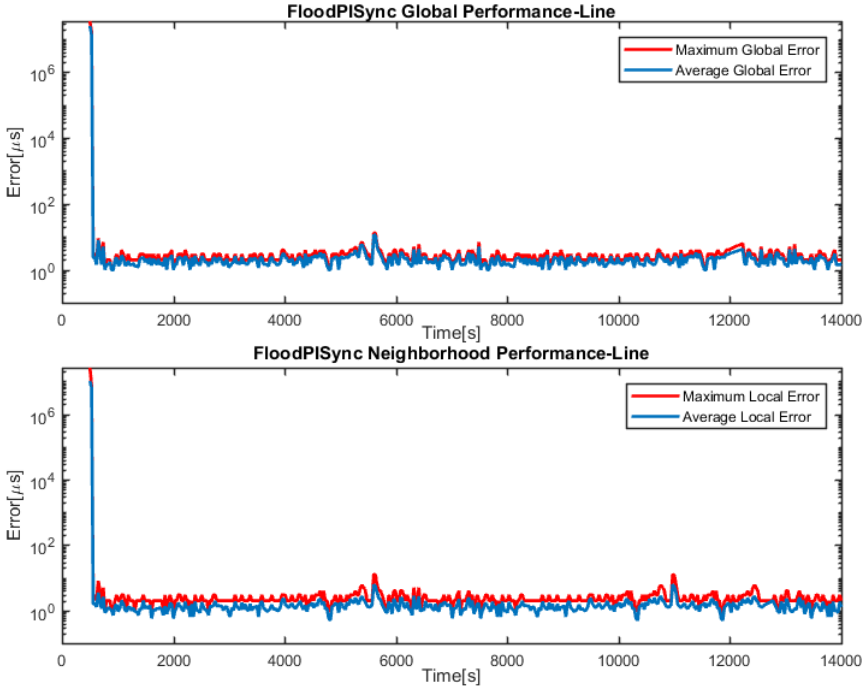


Figure 6.12: FloodPISync Neighborhood and Global Synchronization Error for Line Topology

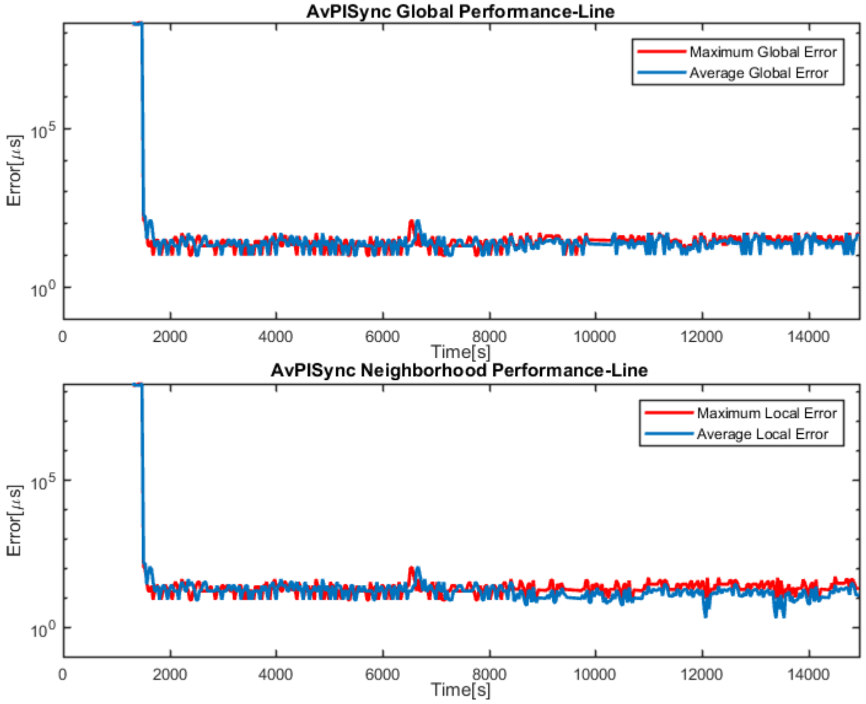


Figure 6.13: AvgPISync Neighborhood and Global Synchronization Error for Line Topology

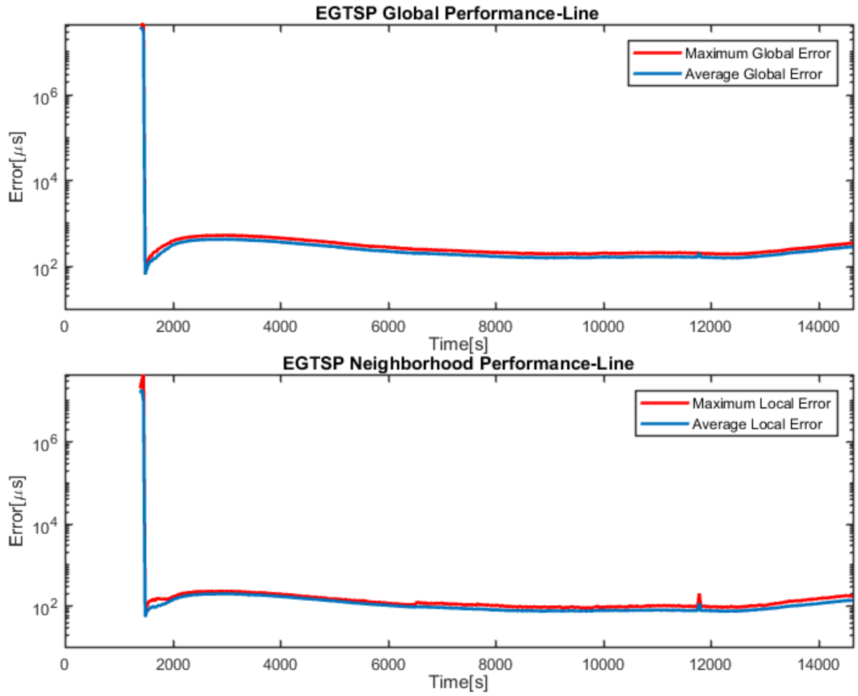


Figure 6.14: EGTSP Neighborhood and Global Synchronization Error for Line Topology

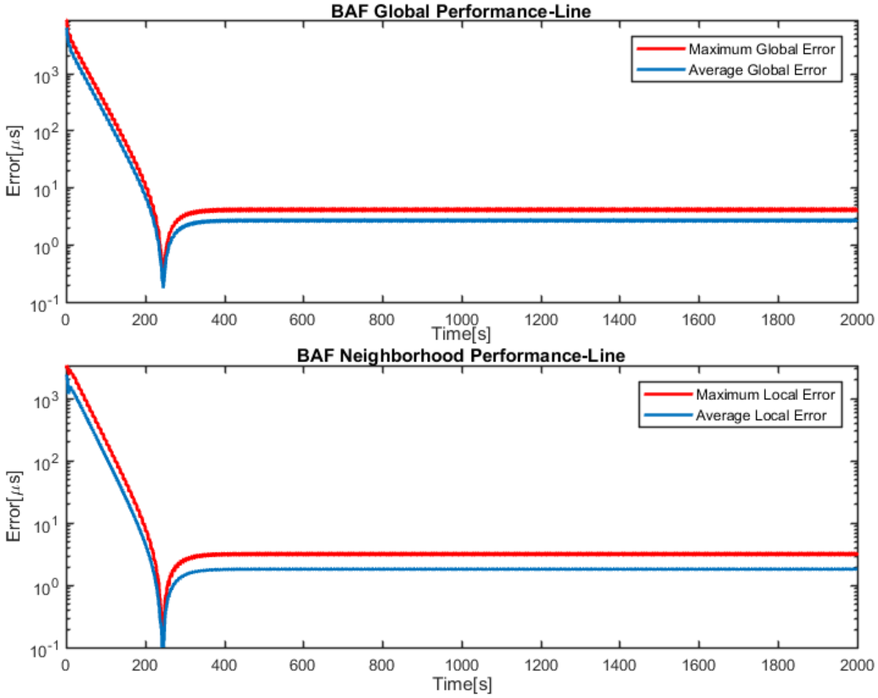


Figure 6.15: BAF Neighborhood and Global Synchronization Error for Line Topology

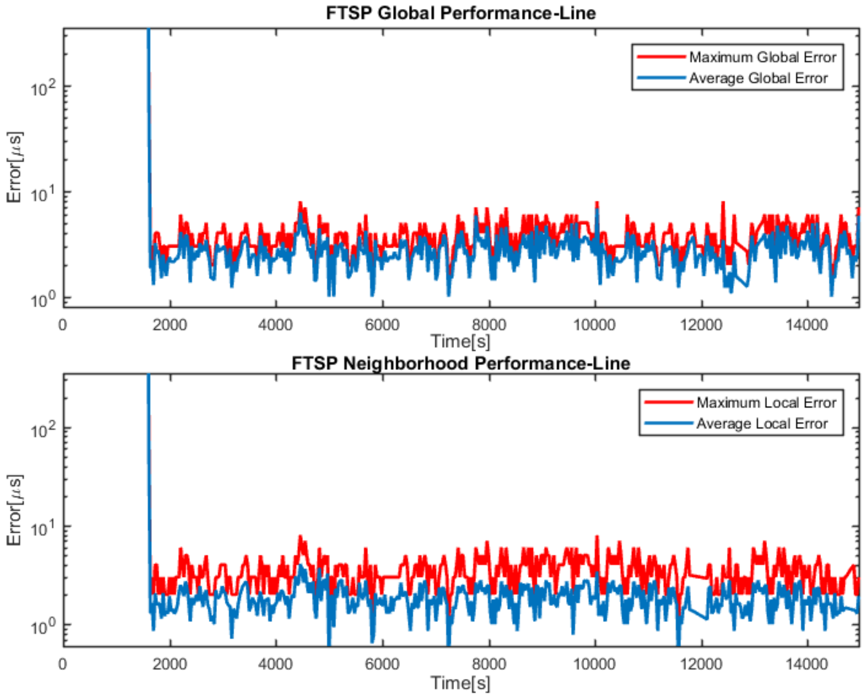


Figure 6.16: FTSP Neighborhood and Global Synchronization Error for Line Topology

6.2 Summary

In this chapter, a comprehensive comparative experimental evaluation was carried out for our developed local time update methods: TSAU, UAF and BAF as against FTSP, EGTSP, AvgPISync and FloodPISync. The experimental parameters adopted for the evaluation of all the protocols were discussed in detail. The test networks employed for the evaluation were 16 node networks of grid and line topologies.

For each protocol, we evaluated how much access and flash memory it requires to effectively synchronize a network. We observed that TSAU, UAF and BAF had less access and flash memory requirements as compared to the other protocols.

For our developed methods, TSAU had the least memory requirements followed by UAF and then BAF. We also compared the packet length of each protocol and observed that, FTSP, FloodPISync and BAF need 27 *bytes* each, AvgPISync and TSAU is require 24 *bytes* each, EGTSP requires 29 *bytes* and for UAF, 25 *bytes* is needed for each synchronization message. The number of CPU cycles a protocol requires to fully process and update the logical clock was also compared for all protocols. EGTSP and FTSP were observed to require more than 30 times the number of cycles needed for all our developed methods. However, the PISync protocols require similar values of CPU cycles as our developed methods. All these parameters were used to compute the total energy consumed by each network node for one cycle of update. We observed that, FTSP, FloodPISync, EGTSP, AvgPISync, TSAU, UAF and BAF respectively require 130.4 μJ, 16.1 μJ, 135 μJ, 14.6 μJ, 14.53 μJ, 14.8 μJ and 16.4 μJ.

To further compare TSAU, UAF and BAF against FTSP, EGTSP, AvgPISync and FloodPISync, we evaluated the synchronization accuracy by comparing the global and local synchronization errors for each protocol for the 16 node gird and line networks. We observed the *dip-before-steady state* phenomenon in all the global and local error curves for TSAU, UAF and BAF. With a well defined minimum error, once a node locates the transient and stops updates, unlike the other protocols where nodes are continuously communicating and estimating the global time through drift and offset compensation, nodes running on TSAU, UAF and BAF only need to identify the transient dip, stop updates and sleep, hence each node spends a short time consuming the energy presented above and is therefore expected to survive for a long time with respect to power if deployed in a wireless sensor network operating in a harsh environment where nodes are prone to failures and devoid of resources.

For the convergence time, since our methods are all decentralized and communication is done only with 1-hop neighbors a lower convergence time is observed as against the other protocols for both grid and line networks. We also observed that, the global and local errors for BAF far out perform all the other protocols. The performance of TSAU was next best followed by UAF. For the proposed methods, their performance in terms of global and local synchronization errors were lower in the line topology as compared to the grid topology.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

Sensor network applications can greatly benefit from synchronized clocks to perform various tasks such as data fusion and energy-efficient communication. A perfect time synchronization algorithm should fulfill a handful of different properties simultaneously: precise global and local time synchronization, fast convergence, fault-tolerance, low memory requirements and energy-efficiency. In this thesis, we presented a time synchronization scheme, based on consensus control and suited for operation in harsh environments that uses a single hop communication scheme to synchronize state parameters representing the virtual clock of each node to the clock of the gateway node of the wireless sensor network. The proposed synchronizer, inspired by [51], was designed with three main components namely; local node time update where nodes establish consensus on a global time, followed by a stopping criterion which halts the update process once a minimum error with respect to the gateway time is achieved. The last component involves putting network nodes to the *idle/sleep* mode for a certain period of drift and switching them back on to resynchronize. This work focused on the first component of the proposed synchronizer. First, this work presented a way of introducing flexibility in the proposed synchronizer by modifying the local node time update in a way that gives a trade-off between the convergence time and synchronization accuracy. This method was inspired by the realization that, the state equation representing the synchronization error with respect to the gateway time has a term, whose effect can be nullified by introducing a new parameter in the local time update of each node. We then carry out simulations and practical experiments on 4, 9, and 16 node networks configured in the grid topology. In our simulations we observed that, generally, with this addition, the synchronization error can be reduced up to a factor of about 5×10^{-4} at the expense of twice the number of iterations for the networks considered. From the practical results we also observed a reduction in synchronization error of up to a factor of about 60×10^{-3} at the expense of twice the number of iterations therefore offering flexibility in the design of the synchronizer. The original local node time update presented for the synchronizer utilized synchronous averaging to achieve consensus in on a global network time which is not practical since synchronous consensus requires synchronization. We extended this observation by adopting the asynchronous approach for local node time update. The main difference being, in the asynchronous time update, a subset of the nodes are activated to update their times based on proximity to the gateway node whiles all other nodes are inhibited. Based on this framework, we designed three different schemes of practically carrying out the activation so as to realize the asynchronous update. The three methods are the Timed Sequential Asynchronous Update (TSAU) where only one node updates in each communication instant, the Unidirectional Asynchronous Update (UAF) where nodes within the same proximity to the gateway node update at the same instant but activation is regulated by the gateway and Bidirectional Asynchronous Flooding (BAF) which removes the need of the activation regulation by the gateway node. These methods and evaluated using simulations and practical experiments on 4, 9 and 16 node networks configured in grid, random and hexagonal topologies. All the suggested methods exhibited the the dip phenomenon even in a network with random connectivity. BAF was generally observed to perform best in terms of synchronization accuracy and UAF and TSAU were observed to perform best in terms of the number of communication cycles needed to converge and the variance in this parameter. To show the feasibility of our suggested methods in practice, we performed experiments on a testbed of 16 grid and line node networks on the TinyOS platform. We also conducted experiment on the same testbed for two fully distributed protocols: Energy Efficient Gradient Time Synchronization protocol (EGTSP) and Average Proportional-Integral Synchronization protocol (AvgPISync) and two centralized flooding-based protocols: Flooding Time Synchronization Protocol(FTSP) and Flooding Proportional-Integral Synchronization protocol (FloodPISync). The suggested methods were observed to outperform FTSP and EGTSP in terms of memory required and energy consumption but had somewhat similar performance as the PISync protocols for these metrics. Further, it was observed that, the global and local errors for BAF far out perform all the other protocols. The performance of TSAU was next best followed by UAF. For the convergence time, a lower convergence time is was observed for our proposed schemes as against the other protocols for both grid and line networks.

7.2 Future Work

Although the dip behavior manifests in all results for our simulations and practical experiments, a mathematical analysis needs to be carried out to ascertain and explain this behavior in future research. Future work may also include testing synchronization performance of the protocol on node and link failures, changing network topology and mobile networks. Research could further be conducted to optimize the algorithm to achieve minimum error at the dip region by designing an optimal filter capable of stopping the update of each node when minimum error is reached. Further studies could also be carried out to test the sensitivity of the protocol to communication impairments like delays, jitters and noise. Also, the period for network resynchronization could be investigated by estimating the time at which each nodes reaches a certain threshold of synchronization error or drift.

REFERENCES

[1] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *Proceedings of the 4th international symposium on Information processing in sensor networks*, p. 48, IEEE Press, 2005.

[2] Memsic, *MICAz, WIRELESS MEASUREMENT SYSTEM*. Memsic Inc, One Tech Drive Suite 325 Andover, MA 01810, 2017.

[3] Memsic, *MTS/MDA, SENSOR, DATA ACQUISITION BOARDS*. Memsic Inc, One Tech Drive Suite 325 Andover, MA 01810, 2017.

[4] K. Sohraby, D. Minoli, and T. Znati, *Wireless Sensor Networks: Technology, Protocols, and Applications*. John Wiley & Sons, Apr. 2007.

[5] G. Kaur and R. M. Garg, "Energy efficient topologies for wireless sensor networks," *International Journal of Distributed and Parallel Systems*, vol. 3, no. 5, p. 179, 2012.

[6] A. Giridhar and P. Kumar, "Toward a theory of in-network computation in wireless sensor networks," *IEEE Communications Magazine*, vol. 44, pp. 98–107, Apr. 2006.

[7] S. Kar and J. Moura, “Sensor Networks With Random Links: Topology Design for Distributed Consensus,” *IEEE Transactions on Signal Processing*, vol. 56, pp. 3315–3326, July 2008.

[8] O. Chughtai, N. Badruddin, and A. Awang, “Distributed on-demand multi-optional routing protocol in multi-hop wireless networks,” in *TENCON 2014 - 2014 IEEE Region 10 Conference*, pp. 1–6, Oct. 2014.

[9] *Protocols and Architectures for Wireless Sensor Networks*. Dec. 2015.

[10] E. Mallada and T. Ao, “Distributed clock synchronization: Joint frequency and phase consensus,” in *in Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference*, 2011.

[11] K. R. Jeremy Elson, “Wireless Sensor Networks: A New Regime for Time Synchronization,” *Computer Communication Review*, vol. 33, no. 1, pp. 149–154, 2003.

[12] H. Karl and A. Willig, *Protocols and architectures for wireless sensor networks*. John Wiley & Sons, 2007.

[13] S. B., U. Buy, and D. Kshemkalyani A, “Clock synchronization for wireless sensor networks: A survey,” *University of Illinois at Chicago, Tech. Rep*, March 2005.

[14] C. Lindsey W, F. Ghazvinian, C. Hagmann W, and K. Dessouky, “Network synchronization,” in *Proceedings of IEEE*, vol. 73, pp. 1445–1467, October 1985.

[15] H. Rentel C, *Network time synchronization and code-based scheduling for wireless ad hoc networks.* PhD thesis, Carleton University, January 2006.

[16] Y. Quan and G. Liu, "Drifting clock model for network simulation in time synchronization," in *Proc. IEEE International Conf. on Innovative Computing Information and Control,*, pp. 385–388, June 2008.

[17] S. Sun, G. Strom E, F. Branstom, and D. Sen, "Long-term clock synchronization in wireless sensor networks with arbitrary delay distributions," in *Proc. IEEE Global Communications Conf. (GLOBECOM*, December 2012.

[18] M. Leng and Y. Wu, "On joint synchronization of clock offset and skew for wireless sensor networks under exponential delay," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, no. 461464, June 2010.

[19] M. Chaudhari Q, E. Serpedin, and K. Qaraqe, "On minimum variance unbiased estimation of clock offset in a two-way message exchange mechanism," *IEEE Trans. Inf. Theory*, vol. 56, p. 28932904, June 2010.

[20] S. L. Yldrm K.S., Carli R., "Adaptive control-based clock synchronization in wireless sensor networks," (Linz, Austria,), IEEE, European Control Conference ECC15, jul 2015.

[21] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks.* John Wiley & Sons, Oct. 2007.

[22] L. Schenato and F. Fiorentin, “Average TimeSynch: A consensus-based protocol for clock synchronization in wireless sensor networks,” *Automatica*, vol. 47, no. 9, pp. 1878–1886, 2011.

[23] S. Lasassmeh and J. Conrad, “Time synchronization in wireless sensor networks: A survey,” in *IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the*, pp. 242–245, Mar. 2010.

[24] D. Mills, “Internet time synchronization: the network time protocol,” *IEEE Transactions on Communications*, vol. 39, pp. 1482–1493, Oct. 1991.

[25] J. Elson, L. Girod, and D. Estrin, “Fine-grained Network Time Synchronization Using Reference Broadcasts,” *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 147–163, Dec. 2002.

[26] S. Ganeriwal, R. Kumar, and M. B. Srivastava, “Timing-sync Protocol for Sensor Networks,” in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, (New York, NY, USA), pp. 138–149, ACM, 2003.

[27] H. Dai and R. Han, “TSync: A Lightweight Bidirectional Time Synchronization Service for Wireless Sensor Networks,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 8, pp. 125–139, Jan. 2004.

[28] J. van Greunen and J. Rabaey, “Lightweight Time Synchronization for Sensor Networks,” in *Proceedings of the 2Nd ACM International Conference on*

Wireless Sensor Networks and Applications, (New York, NY, USA), pp. 11–19, ACM, 2003.

[29] P. Sommer and R. Wattenhofer, “Gradient clock synchronization in wireless sensor networks,” in *International Conference on Information Processing in Sensor Networks, 2009. IPSN 2009*, pp. 37–48, Apr. 2009.

[30] Z. Dengchang, A. Zhulin, X. Yongjun, Z. Dengchang, A. Zhulin, and X. Yongjun, “Time Synchronization in Wireless Sensor Networks Using Max and Average Consensus Protocol, Time Synchronization in Wireless Sensor Networks Using Max and Average Consensus Protocol,” *International Journal of Distributed Sensor Networks, International Journal of Distributed Sensor Networks*, vol. 2013, 2013, p. e192128, Mar. 2013.

[31] W. Su and I. Akyildiz, “Time-diffusion synchronization protocol for wireless sensor networks,” *IEEE/ACM Transactions on Networking*, vol. 13, pp. 384–397, Apr. 2005.

[32] K. Yildirim and A. Kantarci, “External Gradient Time Synchronization in Wireless Sensor Networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, pp. 633–641, Mar. 2014.

[33] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal, “Firefly-inspired Sensor Network Synchronicity with Realistic Radio Effects,” in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, (New York, NY, USA), pp. 142–153, ACM, 2005.

[34] J. He, P. Cheng, L. Shi, J. Chen, and Y. Sun, "Time Synchronization in WSNs: A Maximum-Value-Based Consensus Approach," *IEEE Transactions on Automatic Control*, vol. 59, pp. 660–675, Mar. 2014.

[35] C. Jiming, "Feedback-based clock synchronization in wireless sensor networks: A control theoretic approach.," *Vehicular Technology, IEEE Transactions*, pp. 2963–2973, 2010.

[36] H. Dai and R. Han, "TSync: A Lightweight Bidirectional Time Synchronization Service for Wireless Sensor Networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 8, pp. 125–139, Jan. 2004.

[37] M. Marti, B. Kusy, G. Simon, and k. Ldeczi, "The Flooding Time Synchronization Protocol," in *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, (New York, NY, USA), pp. 39–49, ACM, 2004.

[38] C. Lenzen, P. Sommer, and R. Wattenhofer, "Pulsesync: An efficient and scalable clock synchronization protocol," *IEEE/ACM Transactions on Networking (TON)*, vol. 23, no. 3, pp. 717–727, 2015.

[39] K. S. Yildirim and A. Kantarci, "Time synchronization based on slow-flooding in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 244–253, 2014.

[40] J. He, "Time synchronization in wsns: A maximum-value-based consensus approach," . *ISA Transactions*, vol. 53, no. 2, pp. 347–357, 2014.

[41] Y.-R. Lee and W.-L. W. Chin, "Low-complexity time synchronization for energy-constrained wireless sensor networks: Dual-Clock delayed-message approach," *Peer-to-Peer Networking and Applications*, pp. 1–10, Feb. 2016.

[42] D. Upadhyay and P. Banerjee, "An Energy Efficient Proposed Framework for Time Synchronization Problem of Wireless Sensor Network," in *Information Systems Design and Intelligent Applications* (S. C. Satapathy, J. K. Mandal, S. K. Udgata, and V. Bhateja, eds.), no. 435 in Advances in Intelligent Systems and Computing, pp. 377–385, Springer India, 2016. DOI: 10.1007/978-81-322-2757-1_38.

[43] Q. Li and D. Rus, "Global clock synchronization in sensor networks," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 1, p. 574, Mar. 2004.

[44] R. Leidenfrost, W. Elmenreich, and C. Bettstetter, "Fault-tolerant averaging for self-organizing synchronization in wireless ad hoc networks," in *Wireless Communication Systems (ISWCS), 2010 7th International Symposium on*, pp. 721–725, IEEE, 2010.

[45] C. C. Apicharttrisorn, K. and C. Intanagonwiwat, "Energy-efficient gradient time synchronization for wireless sensor networks," in *Computational Intelligence, Communication Systems and Networks (CICSyN), 2010 Second International Conference*, 2010.

[46] M. Cremaschi, O. Simeone, and U. Spagnolini, "Distributed timing synchronization for sensor networks with coupled discrete-time oscillators," 2007.

[47] M. Maggs, S. O'Keefe, and D. Thiel, "Consensus Clock Synchronization for Wireless Sensor Networks," *IEEE Sensors Journal*, vol. 12, pp. 2269–2277, June 2012.

[48] J. He, P. Cheng, L. Shi, J. Chen, and Y. Sun, "Time Synchronization in WSNs: A Maximum-Value-Based Consensus Approach," *IEEE Transactions on Automatic Control*, vol. 59, pp. 660–675, Mar. 2014.

[49] Y. Kadowaki and H. Ishii, "Event-Based Distributed Clock Synchronization for Wireless Sensor Networks," *IEEE Transactions on Automatic Control*, vol. 60, pp. 2266–2271, Aug. 2015.

[50] J. Wu, L. Zhang, Y. Bai, and Y. Sun, "Cluster-Based Consensus Time Synchronization for Wireless Sensor Networks," *IEEE Sensors Journal*, vol. 15, pp. 1404–1413, Mar. 2015.

[51] A. Al-Shaikhi and A. Masoud, "Efficient, single hop time synchronization protocol for randomly-connected wsns," *IEEE Wireless Communications Letters*, 2017.

[52] I. Nemer, "A distributed time average synchronization protocol for wireless sensor networks," Master's thesis, King Fahd University of Petroleum and Minerals, 2015.

[53] C.-T. Chen, *Linear system theory and design.* Oxford University Press, Inc., 1995.

[54] D. Luenberger, *Introduction to Dynamic Systems: Theory, Models, and Applications.* Wiley, 1979.

[55] S. Kar and J. Moura, “Distributed Average Consensus in Sensor Networks with Random Link Failures,” in *IEEE International Conference on Acoustics, Speech and Signal Processing, 2007. ICASSP 2007*, vol. 2, pp. II–1013–II–1016, Apr. 2007.

[56] L. Schenato and F. Fiorentin, “Average TimeSynch: A consensus-based protocol for clock synchronization in wireless sensor networks,” *Automatica*, vol. 47, pp. 1878–1886, Sept. 2011.

[57] T. van den Boom, “Discrete-time systems analysis.” Internet, Oct. 2006.

[58] M. M. and S. J., *M. Maroti and J. Sallai, Packet-level time synchronization, TinyOS Core Working Group, Technical Report, May 2008. [Online]. Available: http://www.tinyos.net/tinyos-2.x/doc/pdf/tep133.pdf*, may 2008.

[59] C. Lenzen, P. Sommer, and R. Wattenhofer, “Pulsesync: An efficient and scalable clock synchronization protocol,” *IEEE/ACM Transactions on Networking (TON)*, vol. 23, no. 3, pp. 717–727, 2015.

[60] D. G. Cory Sharp, Martin Turon, “Tep-102: Timers.” Online.

[61] "Wiley: Protocols and Architectures for Wireless Sensor Networks - Holger Karl, Andreas Willig," Dec. 2015.

[62] S. Kar and J. Moura, "Distributed Consensus Algorithms in Sensor Networks With Imperfect Communication: Link Failures and Channel Noise," *IEEE Transactions on Signal Processing*, vol. 57, pp. 355–369, Jan. 2009.

[63] P. Denantes, F. Benezit, P. Thiran, and M. Vetterli, "Which Distributed Averaging Algorithm Should I Choose for my Sensor Network?," in *IEEE INFOCOM 2008. The 27th Conference on Computer Communications*, Apr. 2008.

[64] A. Akl, T. Gayraud, and P. Berthou, "An investigation of self-organization in ad-hoc networks," in *2011 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, pp. 1–6, Apr. 2011.

[65] K. S. Yildirim, "Gradient Descent Algorithm Inspired Adaptive Time Synchronization in Wireless Sensor Networks," *arXiv:1512.02977 [cs]*, Dec. 2015. arXiv: 1512.02977.

[66] S. L. Yldrm K.S., Carli R., "Proportional-integral clock synchronization in wireless sensor networks," *IEEE/ACM TRANSACTIONS ON NETWORKING*.

Vitae

- Name: Ramadan Abdul-Rashid
- Nationality: Ghanaian
- Date of Birth: 01-04-1990
- Email: *g201409740@kfupm.edu.sa, ram.rashid.rr@gmail.com*
- Permenant Address: University Blvd, King Fahd University of Petroleum and Minerals, Dhahran 34463, KSA

- Bachelor Degree of Science in Electrical and Electronic Engineering at the University of Mines and Technology (UMaT), Tarkwa, W/R, Ghana, West-Africa in June, 2013.
- Master Degree of Science in Telecommunication Engineering at the King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, KSA from January 2015 until now.