REAL-TIME PERFORMANCE EVALUATION OF FLOODING & RECURSIVE TIME SYNCHRONIZATION PROTOCOLS OVER ARDUINO & XBEE

BY

DANISH SATTAR

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

COMPUTER NETWORKS

DECEMBER, 2013

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

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **DANISH SATTAR** 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 COMPUTER NETWORKS**.

Thesis Committee

Dr. Tarek R. Sheltami (Adviser)

Dr. Ashraf Mahmoud (Member)

Dr. Yahya Osais (Member)

Dr. Basem Madani
Department Chairman

Dr. Salam A. Zummo
Dean of Graduate Studies

1/1/14
Date

عمادة الدراسات العليا
DEANSHIP OF GRADUATE STUDIES

Dedicated to my Beloved Parents

ACKNOWLEDGMENTS

Thanks to Allah for the wisdom and perseverance that he has been bestowed upon me during this research project, and indeed, throughout my life.

Acknowledgment is due to the King Fahd University of Petroleum & Minerals for supporting this research. I am also thankful to King Abdulaziz City for Science and Technology (KACST) for funding this research under project # AR-29-71.

Foremost, I would like to express my sincere gratitude to my advisor Dr. Tarek R. Sheltami for the continuous support of my master's thesis, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my master's thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Ashraf S. Hasan Mahmoud and Dr. Yahya E. Osais, for their encouragement, insightful comments, and hard questions.

I must thank my family, and parents, for their unconditional support; financially, spiritually and emotionally throughout my degree and life. I would also like to thank my grandparents, sisters and brothers for their love and support.

Last but not least, I would like to thank all my friends and colleagues who made my time enjoyable at KFUPM: especially, Ahmad Bilal Numan, Ghazanfar

Latif, Adil Humayun Khan, Khaqan Majeed, Waqas Waseem Ahmed, Fahd Zia, Umar Khan, Hussain Ali, Omair Butt, Furqan Tahir, Ahmad Tariq, Muhammad Adnan Ashraf, Muzammal Naseer for stimulating discussions on various interesting topics, and Mohammad Mohsin Butt for the sleepless nights, when we were working together before deadlines, and for all the fun we have had in the last two and half years.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii

LIST OF TABLES vii

LIST OF FIGURES viii

LIST OF ABBREVIATIONS xii

ABSTRACT (ENGLISH) xiv

ABSTRACT (ARABIC) xvi

CHAPTER 1 INTRODUCTION 1

1.1 Wireless Sensor Networks 1

1.2 Time Synchronization in Sensor Networks 4

1.3 Thesis Objectives 6

1.4 Contribution 7

CHAPTER 2 RELATED WORK 9

2.1 Network Time Protocol 9

2.2 Reference Broadcast Synchronization 10

2.3 Continuous clock synchronization in wireless real-time applications 12

2.4 Network-wide time synchronization in sensor networks 13

2.5 Delay measurement time synchronization protocol 14

2.6 Probabilistic clock synchronization service in sensor networks . . . 15

2.7 Time-diffusion synchronization protocol 16
2.8 Flooding Time Synchronization Protocol 17
2.9 Recursive Time Synchronization Protocol 18

CHAPTER 3 HARDWARE **21**

3.1 Arduino Mega 21
3.2 Zigbee Network 23
3.3 XBee Pro 24
3.4 XBee Shield 25
3.5 XBee Explorer 26
3.6 Hardware Limitation 29

CHAPTER 4 IMPLEMENTATION **32**

4.1 Flooding Time Synchronization Protocol 35
4.2 Recursive Time Synchronization Protocol 37

CHAPTER 5 RESULTS AND DISCUSSION **40**

5.1 Bus Topology 41
5.2 Grid Topology 53
5.3 Mesh Topology 60
5.4 Tree Topology 64

CHAPTER 6 CONCLUSION AND FUTURE WORK **77**

REFERENCES **81**

VITAE **91**

LIST OF TABLES

3.1 Hardware Modules 21
3.2 Arduino Mega 2650 specification summary 22

4.1 Structure of a FTSP Message 36
4.2 Structure of a RTSP Message 38

5.1 Message Bytes Table 41

LIST OF FIGURES

2.1 Message request and reply in RTSP [1] 19

3.1 Arduino Mega 2650 22
3.2 Zigbee Topologies 24
3.3 XBee Pro 900 24
3.4 XBee Shield 25
3.5 XBee Explorer 26
3.6 XBee Explorer with XBee RF module 26
3.7 Screenshot of X-CTU Software 28
3.8 Screenshot of X-CTU Components 29

5.1 Bus Topology 42
5.2 FTSP Bus Topology - Number of Messages per Node for each experiment 43
5.3 RTSP Bus Topology - Number of Messages per Node for each experiment 45
5.4 Bus Topology - Average Number of Messages per node for five experiments with 95% confidence interval 47
5.5 Bus Topology - Average Number of Messages for each experiment with 95% confidence interval 48
5.6 Bus Topology - Average Bandwidth per node for five experiments with 95% confidence interval 48
5.7 Bus Topology - Average Bandwidth for each experiment with 95% confidence interval 49

5.8 FTSP Bus Topology - Convergence Time per Node for each experiment 50
5.9 RTSP Bus Topology - Convergence Time per Node for each experiment 50
5.10 Bus Topology - Average Convergence Time per node for five experiments with 95% confidence interval 51
5.11 Bus Topology - Average Convergence Time of each experiment with 95% confidence interval 52
5.12 Grid Topology 53
5.13 FTSP Grid Topology - Number of Messages per Node for each experiment 54
5.14 RTSP Grid Topology - Number of Messages per Node for each experiment 55
5.15 Grid Topology - Average Number of Messages per node for five experiments with 95% confidence interval 55
5.16 Grid Topology - Average Number of Messages for each experiment with 95% confidence interval 56
5.17 Grid Topology - Average Bandwidth per node for five experiments with 95% confidence interval 57
5.18 Grid Topology - Average Bandwidth for each experiment with 95% confidence interval 57
5.19 FTSP Grid Topology - Convergence Time per Node for each experiment 58
5.20 RTSP Grid Topology - Convergence Time per Node for each experiment 59
5.21 Grid Topology - Average Convergence Time per node for five experiments with 95% confidence interval 59
5.22 Grid Topology - Average Convergence Time of each experiment with 95% confidence interval 60

5.23 FTSP Mesh Topology - Number of Messages per Node for each experiment 61
5.24 RTSP Mesh Topology - Number of Messages per Node for each experiment 62
5.25 Mesh Topology - Average Number of Messages per node for five experiments with 95% confidence interval 62
5.26 Mesh Topology - Average Number of Messages for each experiment with 95% confidence interval 63
5.27 Mesh Topology - Average Bandwidth per node for five experiments with 95% confidence interval 64
5.28 Mesh Topology - Average Bandwidth for each experiment with 95% confidence interval 64
5.29 Tree Topology 65
5.30 FTSP Tree Topology - Number of Messages per Node for each experiment 66
5.31 RTSP Tree Topology - Number of Messages per Node for each experiment 67
5.32 Tree Topology - Average Number of Messages per node for five experiments with 95% confidence interval 67
5.33 Tree Topology - Average Number of Messages for each experiment with 95% confidence interval 68
5.34 Tree Topology - Average Bandwidth per node for five experiments with 95% confidence interval 69
5.35 Tree Topology - Average Bandwidth for each experiment with 95% confidence interval 70
5.36 FTSP Tree Topology - Convergence Time per Node for each experiment 70
5.37 RTSP Tree Topology - Convergence Time per Node for each experiment 71

5.38 Tree Topology - Average Convergence Time per node for five experiments with 95% confidence interval 72
5.39 Tree Topology - Average Convergence Time of each experiment with 95% confidence interval 73
5.40 Comparison for all four topologies for Number of Messages metric 73
5.41 Comparison for all four topologies for Bandwidth metric 74
5.42 Comparison for all four topologies for Convergence Time metric . 75

LIST OF ABBREVIATIONS

AODV	Ad hoc OnDemand Distance Vector
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CPU	Central Processing Unit
DIN	Digital Input
DOUT	Digital Output
EEPROM	Electrically Erasable Programmable Read-Only Memory
FTSP	Flooding Time Synchronization Protocol
ICSP	In Circuit Serial Programming
IEEE	Institute of Electrical and Electronics Engineers
LSLR	Least Square Linear Regression
MAC	Media Access Control
NTP	Network Time Protocol
PAN	Personal Area Network

PWM	Pulse Width Modulation
RBS	Reference Broadcast Synchronization
RF	Radio Frequency
RSSI	Receive Signal Strength Indicator
RTSP	Flooding Time Synchronization Protocol
SFD	Start Frame Delimiter
SPDT	Single Pole, Double Throw
TDMA	Time Division Multiple Access
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
UTC	Coordinated Universal Time
WSN	Wireless Sensor Network

THESIS ABSTRACT

NAME:	Danish Sattar
TITLE OF STUDY:	Real-Time Performance Evaluation of Flooding & Recursive Time Synchronization Protocols over Arduino & XBee
MAJOR FIELD:	Computer Networks
DATE OF DEGREE:	December, 2013

Time synchronization plays an important role in distributed systems. Distributed wireless sensor networks (WSNs) often require accurate time synchronization for coordination and data reliability. The wireless sensor networks have three major goals: time synchronization, low bandwidth operation, and energy efficiency. Different time synchronization algorithms aim to achieve these objectives using various methods.

In this thesis, performance evaluation of two state-of-the-art time synchronization protocols is presented, namely; Flooding Time Synchronization Protocol and Recursive Time Synchronization Protocol. To achieve time synchronization in WSNs, these two protocols make use of different mechanisms: broadcast mecha-

nism is used by Flooding Time Synchronization while peer-to-peer communication is used by Recursive Time Synchronization Protocol. As this is a performance evaluation, three performance parameters were set: the synchronization message count per cycle, the bandwidth and convergence time. Both have been verified using Arduino and XBee using various topologies including bus, grid, mesh, and tree. Each protocol performs differently based on the topology.

خلاصة أطروحة

تلعب مزامنة الوقت دورا هاما في النظم الموزعة. توزيع شبكات الاستشعار اللاسلكية (WSNs) غالبا ما تتطلب تزامن وقت دقيق للتنسيق وموثوقية البيانات. شبكات الاستشعار اللاسلكية لديها ثلاثة أهداف رئيسية: مزامنة الوقت والعمل في نطاق ترددي منخفض، واستخدام الطاقة بكفاءة. تهدف خوارزميات التزامن المختلفة لتحقيق هذه الأهداف باستخدام أساليب مختلفة.

في هذه الأطروحة ،تم المقارنة بين أداء اثنين من البروتوكولات الحديثة وهي (RTSP) و (FTSP) حيث يستخدم كلا منهما آلية مختلفة من الفيضانات لمزامنة الوقت. تم وضع ثلاثة معاييرلاختبار أداء التزامن: عدد رسائل المزامنة لكل دورة، وعرض النطاق الترددي و التقاء الوقت. استخدم المتحكم اردوينو والمرسل (XBee) في المقارنة العملية باستخدام طبولوجيات المختلفة: (bus, grid, mesh and tree) . كل بروتوكول كان أداء مختلف حسب الطبولوجيا .

CHAPTER 1

INTRODUCTION

1.1 Wireless Sensor Networks

Significant advances in technology have occurred in recent years following the creation of low-cost sensors for processing and communicating data [2–5]. Wireless Sensor Networks (WSNs) are a kind of distributed networks wherein sensors are launched to monitor real-time occurrences. Sensor nodes can be installed into any type of environment, and can be mobile or static. Once deployed, they begin to the process of discovering an entire network for communicating the data collected individually. Because of its application in numerous fields from medical [6], environmental [7, 8], military [9], industrial, civilian, microclimate studies [10, 11], and water pollutant monitoring [12], to urban hazard avoidance [13], home networks [6, 14–17], and scientic [18–23] WSNs have earned substantial interest. Now that WSNs are an essential part of the modern age [24–29], it becomes equally essential to address the issues in devising distributed networks.

Conventional sensing architecture is not appropriate for these applications, but they work well in unobstructed environments such as sky observations via weather radars. They consist of a few sensors that are high-powered and long-range; in order for these sensors to communicate, they need a clear line of sight. However, these are not effective in harsh, disordered, and complicated environments where visual contact is limited. A lot of short-range sensors are spent decreasing the clutter effect. This pattern of sensors similarly develops signal-to-noise ratio, which is vital to sensing phenomenon. It is, however, not practical to utilize a lot of wired sensors in large areas, because they involve proper infrastructure. Manual observation is also time- and labor-intensive. Hence, using conventional sensing architecture is considered sensible for these applications.

A good alternative solution to this is the Ad-hoc wireless sensor network, a kind of network architecture that can be utilized to efficiently and effectively monitor and report phenomenon. This network works successfully where conventional network does not. Such a system will eventually be able to affect and not just observe the environment [13, 30, 31].

For this reason, a comprehensive number of vigorous researches have been concentrated on the challenging task of designing a system that works that way. This highlights the important differences between sensor networks and conventional distributed networks. For instance, sensor networks are so unique that they usually work in ways that go against assumptions guiding most conventional techniques.

Perhaps the most important difference between conventional distributed networks and sensor networks is the latters fundamental requirement for efficient energy, as each sensor node can only carry a regulated amount of energy reserves. Wireless sensor networks are used in large quantities so it is nearly impossible to replenish energy reserves. This makes system design even more challenging.

Communication will eventually be dominated by energy consumption, even as electronics becomes ever more efficient [32]. This imbalance between the long- and short-range transmission energy costs and the utilization and reuse of spatial frequency impedes communication from reaching a wider distance. Hence, researches minimized the collaboration overhead and maximized the local processing in order to save energy expenditure for communication. Through up-to-date designs, users can now order the network to do a high-level query. Instead of the node transmitting the entire data, a more efficient scenario would be if the node can first correlate the incoming data with a pattern locally and then return a feedback bearing the time and location of a match.

In order to improve the systems energy efficiency, it is important for data reduction to make use of domain knowledge, local processing, and hierarchical collaboration. Once the system is energy-efficient, it can minimize data size instead of wasting energy by transmitting the raw sensor data to the user.

Sensor networks are also dynamics another quality that makes this type of network important. Nodes will, over time, expected to run out of energy, crash due to a bug in the software, or be subjected to other harsh environmental conditions.

The communication range of a node can still be modified regardless if the topology is not mobile; some conditions that can cause this include the remaining energy in the node, the variation in Radio Frequency(RF) propagation, etc. The nature of this dynamics makes advance predictions extremely difficult, which in turn makes it completely unfeasible to create a sensor network design where every node is given attention. This highlights two unique requirements of sensor networks: first, the nodes must be self-configuring, and second, the nodes must be capable to adapt to changes.

The sensor networks distinctive and strict preconditions can be seen in virtually every facet of system design, from naming and binding, to mechanisms involving routing, addressing, and security, to application architecture and so on. Even the process of deployment the tedious method of system design, construction, and evaluation has to necessarily adapt to this routine [33].

1.2 Time Synchronization in Sensor Networks

In any distributed system, there is time synchronization, and even more so with sensor networks. However, it has always been more challenging to make time synchronization work with sensor networks than it is to get it to work in the usual distributed networks.

For sensor networks, energy efficiency is essential, and it is usually accomplished through collaboration between nodes. A usual interpretation of physical time is key condition for nodes to reason about occurrence in the physical world;

for instance, cause an acoustic beam-forming array to distribute [34], combine a time-series of proximity detections into a velocity approximation [10], withhold outmoded messages created by different sensors by identifying matching findings about similar occurrences [35], or develop a low-power Time Division Multiple Access (TDMA) radio schedule [36]. Also, as conventional distributed systems do, sensor network applications usually depend on synchronization for database queries [37, 38], authentication and cryptography systems [39–41], harmonization of future action, interface with users, requesting logged events in the course of debugging the system, calculate the time-of-light of sound using exact time [42, 43], and so on.

Utilizing time synchronization in WSNs renders it vital; however, it also causes it to be different and thus more difficult to solve. Prerequisites for applications can vary in precision, energy, lifetime, and availability. A case in point concerns global queries that necessitate global time and local collaboration that usually needs synchronization between at least two or more neighbors. Sensor tasking operates on an hourly or daily scale, while acoustic applications operate at a microseconds level, oftentimes requiring precision; meanwhile, triggers may simply need momentary synchronization, even as data logging or debugging needs a time scale that is always constant. User communication needs Coordinated Universal Time (UTC) or other external time scales, while for wholly in-network appraisals need relative time only. Nodes also differ: while some nodes are equipped with large batteries to enable them to run continuously, other nodes are limited in their

capacity, requiring them to intermittently wake up to do single sensor readings, communicate it, and return immediately to sleep mode.

It is challenging to design a time synchronization that satisfies so many requirements, especially for sensor networks wherein, apart from the usual prerequisites, they also have to exhibit energy efficiency and automatic adjustment to scalability and dynamics. For example, energy limitations disrupt the rules established by the conventional algorithms of time synchronization: that a CPU, when inactive, uses up small amounts of energy, or that random transmissions have minimum impact, or that network listening is free. Nonstop, automatic configuration is needed for node and network dynamics; this prevents a specific node from being assigned a priori as the master clock.

This leads to the paradox of sensor networks: while it places high demands on a time synchronization system, it also constrains the networks capacity to achieve time synchronization..

1.3 Thesis Objectives

Time synchronization is a significant issue when it comes to Distributed Wireless Sensor Networks (WSNs). Time synchronization in WSNs has numerous difficulties not usually present in conventional networks. For instance, sensor nodes are limited in terms of hardware, bandwidth, power, and unstable network connectivity; this poses a challenge in executing WSNs time synchronization. Two contemporary algorithms being implemented to achieve time synchronization in

WSNs include Recursive Time Synchronization Protocol (RTSP) and Flooding Time Synchronization Protocol (FTSP).

The goal of this study is to make performance evaluation of Flooding Time Synchronization Protocol and Recursive Time Synchronization Protocol in order to utilize them using XBee and Arduino. Results of this study are expected to achieve the following:

First, to investigate the feasibility of using Arduino and XBee as hardware platform of various protocols for analyzing performance; this includes identification of their constraints as hardware platform.

Second, to conduct an in-depth performance evaluation of Flooding Time Synchronization Protocol and Recursive Time Synchronization Protocol, two time synchronization protocols for wireless sensor networks; this entails testing the protocols in a real-time environment, as well as determining which of these protocols is more practical.

1.4 Contribution

The major contributions of this research fall in two categories. Our first contribution is, the investigation of Arduino and XBee as a feasible hardware platform for performance evaluation of different protocols. We also contributed their limitations as a testbed hardware platform.

Our second contribution is, the in-depth performance evaluation of two time synchronization protocols for wireless sensor networks namely: Flooding Time

Synchronization Protocol (FTSP) and Recursive Time Synchronization Protocol (RTSP) in a real-time environment as well as which protocol is more practical.

CHAPTER 2

RELATED WORK

Computer clock synchronization is a well-researched dilemma that has been studied since the first computer networks were designed [44]. This chapter will introduce selected works that are related and relevant to this research; while we cannot examine all existing related works, we will provide what we view as the best and most relevant works relative to our research.

2.1 Network Time Protocol

When it comes to synchronizing physical clocks in computer networks, there have been many protocols recommended over the years [45–48]. The following are the common characteristics of these protocols: their message exchange involves using user datagram protocol; the presence of one or more servers; the timing information are exchanged among clients; there are methods set up in case of error recovery during processing and transmission; and using a client algorithm and master server information to update the clock. Some differences have also

been noted, such as whether the network is locally synchronized or externally synchronized with a clock; whether the master server works as the intermediary of client clocks or as the only clock, among others.

The differences of these methods have been widely documented in literature. Most protocols follow the simple model of clock synchronization (detailed above) and recommend heuristic and statistical methods to improve its strength.

For computer clock synchronization, Mills proposed a Network Time Protocol (NTP) [47] that possesses scalability, sturdiness against failure and sabotage, self-configuration in large multi-hop networks, and universal placement, as well as the capacity to create a time server hierarchical structure. The NTP has radically changed over three decades; the original NTP has incorporated many new ideas over the years that has led to its present status as the de facto default time synchronization used on the internet.

2.2 Reference Broadcast Synchronization

As its name suggests, the Reference Broadcast Synchronization (RBS) protocol uses wireless medium as its broadcast mechanism [49]. Using the wireless medium as broadcast mechanism imposes that any receiver within the transmitters range of transmission will simultaneously receive any broadcasted data. To paraphrase, a group of receivers can receive the same data with only a little difference in the delay. This message can be used for clock synchronization. If each receiver logs the time stamp at which message is received and comparing it against their local clock

and up-dating their local clocks according to received message, all receivers within proximity of transmitter can synchronize with high accuracy. RBS uses multiple synchronization messages to calculate both skew & offset of the local clock relative to one another. The path of the message that contributes to non-deterministic errors in protocol is called time-critical path. RBS exploits this concept as well.

Some additional features are: no clock correction - once clocks are synchronized i.e. skew and offset are calculated, nodes local clock are not synchronized with global clock; post-facto synchronization - RBS conserves high level of energy, because it synchronized clocks when the event of interest occurs, so nodes can go to power saving mode at other times when there is no event of interest; multi-hop communication - RBS uses an intermediate node to synchronize two nodes that are situated at different ends otherwise it will lose accuracy due to delay in broadcast message reception.

RBS has several advantages e.g. skew & clock offset are calculated independent of one another, post-facto synchronization conserves energy on the expense of clock updates, multi-hop communication is supported and it can maintain both clocks relative and absolute. On other hand, RBS have some drawbacks as well e.g. it is not suitable for point-to-point networks, it requires $O(n^2)$ messages exchange for a single hop network of n nodes, convergence time is high and if reference node require synchronization than a significant number of messages will be exchanged thus waste of energy.

2.3 Continuous clock synchronization in wireless real-time applications

IEEE 802.11 standard [50] was extended by Mock et al [51]. They defined a new continuous clock synchronization protocol for WSNs. This protocol is similar to RBS [49], because it also takes advantage of the tightness characteristic of wireless medium as well as has tolerance for message loss. Main achievement of this protocol is, it uses continuous clock synchronization in contrast to instantaneous clock synchronization used by IEEE 802.11 standard.

In instantaneous clock synchronization, each node adjust its clock after calculating local clock error. The result is an abrupt change in clock time of node that can cause time disruption. Time disruption can cause severe malfunction in distributed systems e.g. node recoding same event multiple times or missing crucial events (e.g. deadlines).

Continuous clock synchronization mitigates this problem by correcting the clock over finite interval. The local clock is adjusted by moderately slowing down or speeding up the clock. However, this approach have very high processing overhead, because clock need to be corrected every tick.

Some advantages of continuous clock synchronization are; it improves clock accuracy by using continuous clock instead of instantaneous clock as in IEEE 802.11 protocol, message complexity is quite low, another advantage is protocol accounts for potential message loss. But there are some disadvantages as well. It does not have capability to communicate in multi-hop network, because it assumes

tight communication between master and slave, high energy consumption as well as high computation cost at each node.

2.4 Network-wide time synchronization in sensor networks

One of the most important requirement for any protocol is scalability. In wireless sensor network, designing a scalable time synchronization protocol is very challenging due to limited resources. In this protocol authors tries to mitigate scalability problem of clock synchronization in WSNs. The network-wide time synchronization protocol aims at high clock synchronization accuracy even if the network size keeps growing [52] [53]. The objective of network-wide time synchronization protocol is to create unique global clock by creating self-configuring hierarchical structure in wireless sensor network. A node can perform both tasks at the same time, either it could be a synchronization client or it could be a synchronization server for some nodes. Unlike other methods RBS or continuous clock synchronization that can only work in small clusters, this can work in a wide network of nodes. It works in two phases; the level discovery phase and synchronization phase.

Level discovery phase - a level is assigned to each node, root node consider itself at level 0, than it broadcasts the message to neighboring nodes. Each neighbor will assign a level one more than the received i.e. level 1 and again they will

broadcast the message with new level until all the nodes have been assigned a level. Next phase is synchronization - it uses clock offset and propagation delay with the timing information for synchronization.

Scalability is not an issue for this protocol and also it has less computational cost as compared to NTP. But there are some drawbacks, it is energy inefficient, and not suitable for mobile nodes.

2.5 Delay measurement time synchronization protocol

If uniform time is maintained among nodes within a network than it is called Time keeping [54]. A global clock as well as local clock can be used for time synchronization. Key features of this protocol are: maintenance of local clock; network time is created by synchronizing local clocks; global can be synchronized by connecting it to synchronization leader and Application Programming Interface (API) for providing services to client applications.

Berkeley motes running TinyOS kernel [4, 5] were used for implementation. Network event scheduling and event timestamps concepts were used for node synchronization. In this protocol, there is a better chance of synchronizing receivers with each other as compared to the sender.

It also supports multi-hop synchronization. Another upside of this protocol is that it provides user application interface to monitor WSNs at run-time. Energy

efficiency is high, because computational complexity is low, but this is a drawback as well. Due to low computational complexity, synchronization accuracy is low as well. So, there is a tradeoff between energy efficiency & high synchronization accuracy.

2.6 Probabilistic clock synchronization service in sensor networks

Most time synchronization algorithms rely on deterministic algorithms. The main benefit of deterministic algorithms is that they usually guarantee an upper bound on clock offset estimation error. However, downside of this approach is that, when the resources are scarcer, a guarantee on time synchronization can lead to large number of messages being exchanged between nodes during synchronization process. On the other hand, probabilistic methods can provide acceptable time synchronization accuracy with lower network overhead and computation than deterministic algorithms. PalChaudhuri et al. [55] proposed a probabilistic algorithm for time synchronization in wireless sensor networks. They extended the RBS [49] protocol further. They provided a probabilistic bound on the accuracy of clock synchronization. This protocol allows to tradeoff to dynamic synchronization accuracy for computational cost and energy resources.

Some advantages of this protocol are supports multi-hop networks. A trade of between synchronization cost and resources is allowed. But there are certain

downside as well. In critical applications (e.g. nuclear plant monitoring), this algorithm might not work well, and this protocol also is sensitive to message loss.

2.7 Time-diffusion synchronization protocol

In Time-Diffusion synchronization protocol [56], local time of sensor nodes is within a small bounded time deviation from the network global time. It has to be applied at regular intervals due to clock skew. Time-Diffusion have two phases i.e. active and passive. The protocol consists of many small algorithms.

Active phase consist of numerous cycles and each cycles duration is T. In each cycle, master nodes are selected by Election/Reelection Procedure (ERP). Diffusion of timing messages are initiated by each master independently. For each diffusion, these messages are dynamically propagated in a tree like structure. Diffusion leaders that propagates the timing messages are the non-leaf nodes of the tree. ERP is used to elect these diffusion leaders.

Some advantages of Time-Diffusion synchronization protocol are: it can be deployed in a mobile network environment. It does not require external time server for clock synchronization. It can also tolerate message loss. Although there is a hierarchical structure, which is difficult to deploy in a mobile network, but having multiple master nodes neutralizes the effect. On the downside, this protocol has high complexity due to having multiple cycles with multiple rounds in each active period. Convergence time is high.

2.8 Flooding Time Synchronization Protocol

In wireless sensor networks, Flooding Time Synchronization Protocol (FTSP) [57] is the most widely utilized protocol for time synchronization among nodes. Basically, it employees broadcast mechanism to achieve synchronization objective. The reference node broadcasts the message containing timing information, nodes in its broadcast radius listens to the message and update their local clock. Reference node is elected dynamically and it broadcast its timing information after regular interval. FTSP also creates an ad-hoc tree structure instead of fixed spanning tree. To eliminate effect of random delays, Media Access Control (MAC) layer time stamping is used at both ends. These time stamps are embedded into each message at the end of Start Frame Delimiter (SFD) byte, but after correcting error and normalization of time stamp. Least square linear regression (LSLR) is used to estimate offset & skew of timestamps. After node's local clock is synchronized with global clock, it also start broadcasting timing information in the network that way whole the network is covered. In a single hop network, FTSP has accuracy of 1.48μs.

Several enhancements for FTSP have been proposed in the literature in terms of efficiency, accuracy, energy consumption. For instance, power consumption and accuracy has been improved by D. Cox et. al. [58] for a single hop network. They have used SFD based time-stamping mechanism to improve both of these parameters. Another improvement was proposed by M. Aoun et al [59], in which they used Kalman filter for skew estimation and SFD based time stamping to

achieve accuracy at microsecond (i.e. 0.4μs) for a single hop network. With these proposed improvements, accuracy is not a problem for FTSP, but energy consumption is still a concern. Another problem with FSTP is poor synchronization at border nodes.

2.9 Recursive Time Synchronization Protocol

The Recursive Time Synchronization Protocol (RTSP) [1, 60] supports multiple topologies, i.e. flat, mesh, star, and cluster. Any node in the network can request for time synchronization. The request is forwarded in a recursive multi-hop manner to the reference node. The reply will be sent via reverse path by the reference node (A node will be elected dynamically and according to this node timing information, whole network will be synchronized. This node is also called root node), or any intermediate node, which is synchronized. Each node along the reverse path will compensate for time drift (propagation delay). It also uses MAC layer time stamping at both ends. Time stamps are embedded at the end of SFD byte. There are many other algorithms proposed in RTSP like selection of reference node, clusterheads, and working under different topologies, but this is a brief overview of the algorithm. In the following, we will explain the RTSP protocol using an example. RTSP message consists of message type (Enquiry/Election (ERN), Request (REQ), and Reply (REP)), message ID, original source ID, intermediate source ID, destination ID, T1 (When the request was sent from source), T2 (when the request was received at destination), T3 (when the

reply was sent), T4 (when the reply was received at the destination node) and Tr (Reference Time). Time drift is calculated and adjusted at each node along the reply path, using following formula:

$$d = ((T_2 - T_1) + (T_4 - T_3))/2,$$

$$T_r = T_r + d.$$

Where, "d" is the time drift. This time drift will compensate for the propagation delay. When the message will reach edge nodes, reference time will be the same as on reference node.

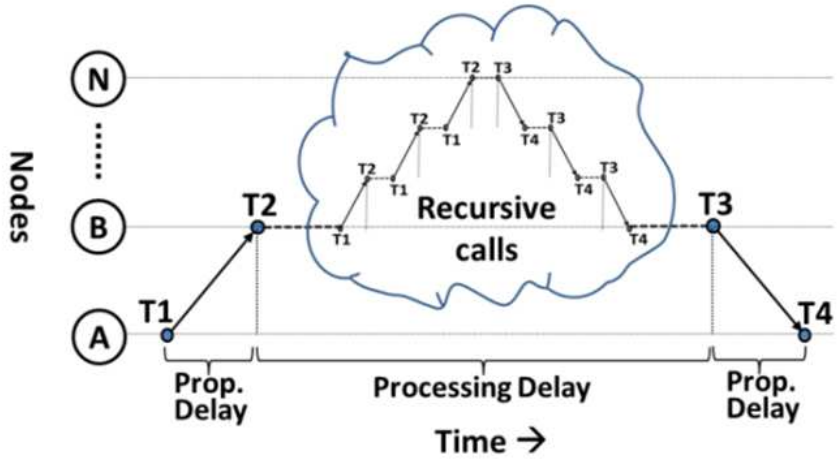


Figure 2.1: Message request and reply in RTSP [1]

Figure 2.1 illustrates the request and reply message propagation in RTSP. Node A send a time synchronization request at time T1 to the synchronized node or reference node via an intermediate node B that will receive the request message

from A at time T2. If node B is not synchronized, it will forward the request message to another node in recursive manner until it reaches a fully synchronized node or the reference node. Now, the synchronized node or reference node reply with timing information at time T3 with reference time Tr. The intermediate node receives the reply message at time T4. The intermediate node will calculate the time drift using the above formulas and adjusts the Tr. This reply message follows the reverse path (of request message) until it reaches node A. Time synchronization is insensitive to processing delay, so each node would only have to compensate for propagation delay.

CHAPTER 3

HARDWARE

In this chapter we will discuss hardware modules namely: Arduino Mega, XBee Pro, XBee Shield, and XBee Explorer. Following table shows complete list of hardware used for experimentation.

Name of Module	**Quantity**
XBee Pro Module	16
XBee Shield	16
Arduino Mega Board	15
Arduino Uno	1
XBee Explorer	1
9V Battery	30

Table 3.1: Hardware Modules

3.1 Arduino Mega

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 chipset [61]. It has 54 digital input/output pins (of which 15 can be used as Pulse Width Modulation (PWM) outputs), 16 analog inputs, 4 Universal Asynchronous

Receivers/Transmitters (UARTs), a 16 MHz crystal oscillator, a USB connection, a power jack, an In Circuit Serial Programming (ICSP) header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a Universal Serial Bus (USB) cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

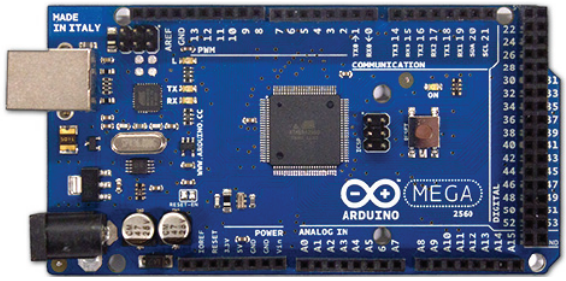


Figure 3.1: Arduino Mega 2650

Parameters	**Value**
Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Table 3.2: Arduino Mega 2650 specification summary

Figure 3.1 shows Arduino Mega 2650 and summary of its specifications is shown in table 3.2.

3.2 Zigbee Network

Zigbee and 802.15.4 both provide infrastructure for wireless sensor networks. Zigbee defines application and network layer whereas 802.15.4 provide data link and physical layer specifications. Zigbee always makes a Personal Area Network (PAN). Each device can only join one PAN network maintained by the coordinator device. Zigbee defines three device types: coordinator, router and end devices.

Coordinator is the most powerful device in the Zigbee network. It selects a channel and PAN id to start the network, has capability to allow routers and end devices to join the network, assist in routing of data and can never go to sleep mode. Each PAN can have only one coordinator.

Router is a less powerful devices as compared to the coordinator. It allows routers and end devices to join the network. As name suggest, it is used for routing of information. It also can never go to sleep.

End device is the least powerful device. It must join PAN network to send or receive information. It does not allow devices to join the network. End devices cannot communicate with each other directly, but they can go to sleep mode.

Zigbee supports multiple network topologies i.e. star, cluster tree and mesh topology. In star topology, coordinator is at the center of network and will relay all information from one node to another. In cluster tree topology, multiple cluster star topologies are connected together to make a cluster tree topology and in mesh topology, all devices are connected to each other in their communication range. These topologies can be seen in Figure 3.2 [62].

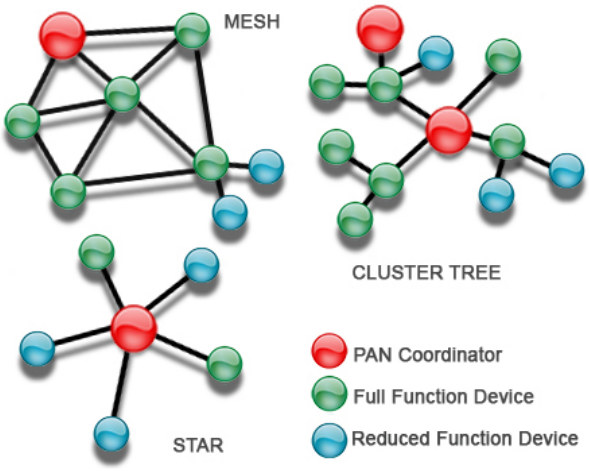


Figure 3.2: Zigbee Topologies

3.3 XBee Pro

Figure 3.3: XBee Pro 900

XBee Radio Frequency (RF) modules are cost effective wireless solutions. These devices are not difficult to deploy. By default, mesh topology is formed. XBee can be used for home automation, temperature sensing, smart grids and etc. Very minimal configurations are required. It has self-healing and discovery

capabilities to maintain network connectivity [63]. The RF module used in our experimentation works at 2.4GHz.

3.4 XBee Shield

XBee radios are an excellent way to add wireless capability to Arduino. XBee shield is used as intermediate layer between Arduino and XBee RF module [64].

The serial pins (DIN and DOUT) of the XBee are connected through a Single Pole, Double Throw (SPDT) switch, which allows you to select a connection to either the UART pins (D0, D1) or any digital pins on the Arduino (D2 and D3 default). Power is taken from the 5V pin of the Arduino and regulated on-board to 3.3V DC before being supplied to the XBee. The shield also takes care of level shifting on the DIN and DOUT pins of the XBee. Figure 3.4 shows an image of the XBee shield.

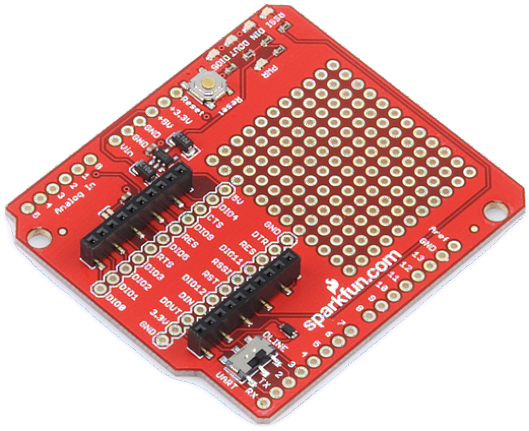


Figure 3.4: XBee Shield

3.5 XBee Explorer

XBee explorer is a simple to use, USB-to-serial base unit for the XBee RF modules [65]. XBee explorer and XBee explorer with XBee RF module is shown in Figures 3.5 and 3.6, respectively.

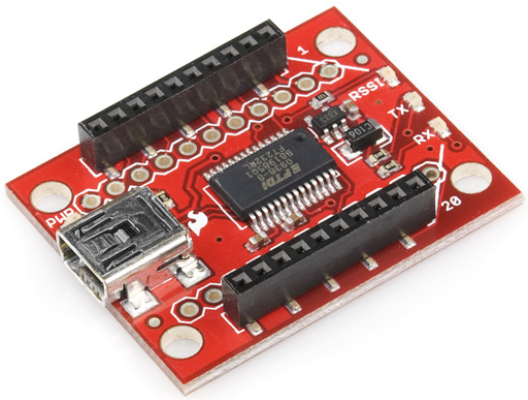


Figure 3.5: XBee Explorer

Figure 3.6: XBee Explorer with XBee RF module

A software from Digi International called X-CTU [66] is used for testing and configurations of XBee RF modules. Features of the software are:

- Integrated terminal window.

- Display of Receive Signal Strength Indicator (RSSI).
- Display both American Standard Code for Information Interchange (ASCII) and hexadecimal characters in terminal window.
- Compose test packets in either ASCII or hexadecimal for transmitting in terminal interface.
- Save and retrieve commonly used module configurations (known as profiles).
- Automatically detect module type.
- Restore factory default parameters.
- Display help about each of the radio parameters.
- Program radio profiles in a production environment using command line interface.

The software is easy to use and allows to test the radio modems in the actual environment with just a computer and the items included with the radio modems [66]. X-CTU can be seen in Figure 3.7.

Minimum configuration that is required XBee to work in API mode as is shown in Figure 3.8. The first step is to select an appropriate modem type from the dropdown menu, as shown in Figure 3.8a or another way is to read configuration of the modem after attaching it to the XBee explorer, but second option is only available for the XBee Pro modules. Next step is to decide whether this device is going to be a Zigbee coordinator, router or an end device (Figure 3.8b). Although,

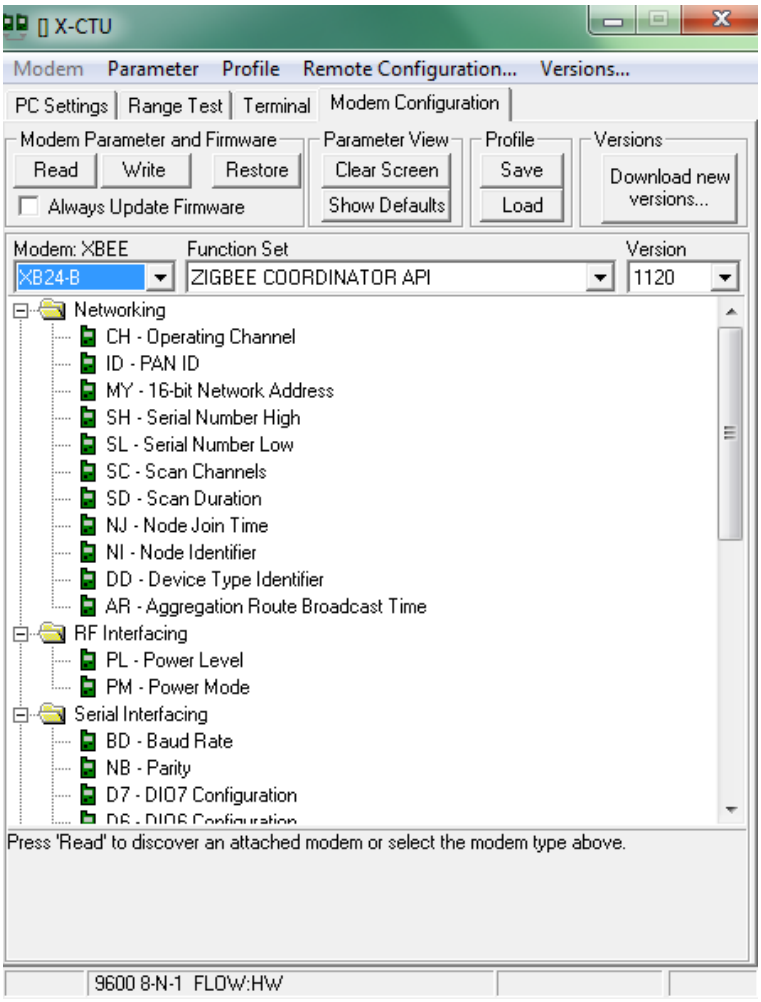
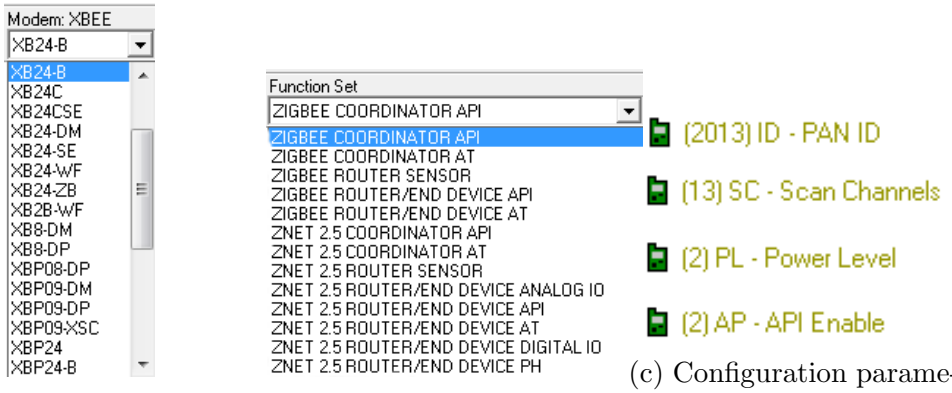


Figure 3.7: Screenshot of X-CTU Software

router API and end device API have no difference, both have functionality of a router. If this device is a coordinator than set the minimum parameters required as shown in Figure 3.8c. Coordinator will set PAN id ranging from 0-3FFF, a scan channel (0-0xFFFF), power level and API mode. Power level is optional, but it is s a good practice to set power level. In our example, modem is XB24-B and coordinator in API mode as shown in Figure 3.8. Rest of the parameters are: PAN id “2013”, scan channel “13”, power level “2-medium” and API mode is “2”.



(a) XBee Modems (b) Function set (c) Configuration parameters

Figure 3.8: Screenshot of X-CTU Components

After setting these parameters, coordinator is ready to make a network. Assign same parameters for routers and end devices. Turn on the power, these devices will automatically make a ready to use network.

For AT or serial mode same configurations are required except we need to choose coordinator/router/end device in AT mode instead of API mode from function set menu as shown in Figure 3.8b.

3.6 Hardware Limitation

There are a few limitations with the aforementioned hardware. We start with XBee Pro. This device has proprietary implementation, which means we are constrained with the functions provided by the vendor. We cannot implement custom routing protocol and embed it into its network layer. For example, clustering routing protocol cannot be implemented at network layer. It has to be at application layer, which increases the complexity and processing. MAC frame cannot be modified due to proprietary implementation, so there is no embedding of time stamps

after SFD byte. Another very important problem with the XBee Pro module is that it implements devices as a software API i.e. for coordinator, router and end device, we have a built-in API that can be loaded onto the device as explained in section 3.5. If these devices truly implemented Zigbee than we should be able to make aforementioned(section 3.2) network topologies easily, but this is not the case. Zigbee dictates, if two devices are configured as end devices than they should not be able to communicate with each other directly. But in case of XBee pro, even if devices are configured as end devices API, they are still able to communicate with each other directly that complicate things further. There are also some implementation issues when using Arduino and XBee. If several packets are coming, XBee might be able to receive all messages and transfer it to Arduino, but due to complex programming at application layer, Arduino might not be ready to receive packets all the time. It might be busy in processing previous packet. Arduino can transfer 6 packets every second to XBee, but XBee cannot handle the packets at that speed. So, to make it compatible with each other, we have to find an appropriate packet transfer rate between Arduino and XBee. This also puts an extra overhead in implementation. XBee allows maximum packet size of 72 bytes, but it is not possible in our case to use all 72 bytes, because it cannot exchange 72 bytes packet at the rate we are transmitting in our experiment. To fully utilize packet size, there should be some delay between sending two packets. Packet size must be 50 bytes or less for efficient working of network.

One very crucial problem relating to energy is that routers and coordinators

cannot go to sleep, and their role cannot be changed dynamically, so they require a constant energy source. Thus, energy efficiency aspect of both protocols cannot be analyzed using current hardware. Arduino also makes it hard to calculate energy utilization, because it does not provide functionality to calculate energy consumption.

Arduino micro-controller is 16-bit, it can only store microseconds for approximately 70 minutes after which it automatically resets the variable that holds the microseconds information, which is a disadvantage and limits the broadcasting duration of timing information (i.e. we have to broadcast timing information every 70 minutes or less). Another limitation of Arduino is that we cannot dynamically reply to the incoming message. We can read the network address from incoming packet, but to reply dynamically, all the network address need to be pre-defined in the software program.

CHAPTER 4

IMPLEMENTATION

In this chapter implementation details are discussed. Both protocols are simplified due to several reasons. One of the reason is hardware limitations as explained in previous chapter. Another reason is some sections of the proposed algorithm are not relevant in this context. We use time resolution in seconds due to the fact that it is not possible with the current hardware to use μs resolution for experimentation.

Due to hardware limitations, both algorithms are implemented at application layer. All traffic will be coming, all the way up to application layer for processing. We cannot utilize built-in functionality of lower layers, to simplify our implementation. For example, we have to implement duplicate message checking mechanism at application layer rather than at MAC layer, which already does that. These little function adds complexity and overhead at application layer. There are also some implementation issues when using Arduino and XBee. If many packets are coming, XBee might be able to receive all messages and transfer it to Arduino, but

due to complex programming at application layer, Arduino might not be ready to receive packets all the time. It might be busy in processing previous packet. This also puts an extra overhead in implementation. Another scenario for Arduino to not receive a packet is, if we add delay in Arduino side implementation. It has an adverse effect on network communication. Even if we add few millisecond delay to avoid congestion, all nodes get out of synchronization and Arduino is not able to receive several packets, which creates a disorder between nodes. There is another technical issue with the implementation. All the nodes have to power up manually, which creates an initial clock difference. Some clocks are ahead of others, this can lead to false root node selection. For example, if there are two nodes with id's 5 and 10. If we power up the node 10 before the node 5, than it would get ahead start and there is a strong possibility that it can become a false root node. With two nodes it can be solved easily, but when there are 15 nodes, it becomes difficult to handle manually. In addition to that, it also contributes to human error. To avoid these technical issues and errors, the last node that power up, will immediately broadcast a reset message and each node will reset their local clocks to zero state and then the process of root selection and time synchronization will begin, thus eliminating any human errors.

There are two kind of messages that are present in the network i.e. system messages and application messages. System messages are those, which are used by XBee to communicate. For example, routing messages or Ad-hoc On-Demand Distance Vector (AODV) messages [67]. As we know, AODV is also a broadcasting

protocol used by XBee for routing, so it will create a broadcast path whenever it requires, but these messages are hidden from the user. User cannot interfere with these messages or modify them in any way. MAC layer will be checking for duplicate messages, but only for system messages not for application messages. On the other hand, application messages will be filtered at application layer, like duplicate message checking etc.

Metrics used for performance evaluation are message count, the bandwidth, and convergence time. Total number of message (incoming and outgoing) per synchronization cycle, required for clock synchronization by each node is called message count. Now for message count, any message that comes to application layer is counted in message count metric. As mentioned before, protocols are implemented at application layer, so even duplicate message will come to application layer, although there will be no processing on these duplicate messages except they will be discarded, but they will be counted into message count metric. Second metric is the bandwidth. It is similar to message count, because we will just multiply message size with message count, but it is separated from message count, because in some topologies message count is lower for one protocol, but its bandwidth is high due to large message size. So, we need to distinguish between message count and bandwidth. Last metric for evaluation is the convergence time. It is the time required by an individual node for synchronization with root node.

We have implemented four topologies i.e. bus, grid, mesh, and tree. Zigbee can make most of topologies without any problem, but using XBee it is difficult

to make these topologies. Therefore, we had to make these topologies manually or in other words, nodes were arranged in shape of required topology e.g. For tree topology, nodes were arranged in a tree like structure. Detail implementation of each topology will be explained in next chapter.

Keeping in mind hardware limitations and above explained restrictions, we will discuss each protocol implementation in detail in following sections.

4.1 Flooding Time Synchronization Protocol

Flooding Time Synchronization Protocol (FTSP) is a very simple and elegant protocol. It uses simple broadcasting mechanism for root selection and time synchronization. According to authors, each Mica2 mote's clock drift 40μs each second [57]. To compensate for this, they have implement a clock drift mechanism. More details of protocol are described in section 2.8.

In our FTSP implementation, we have done some simplifications. There is no clock drift mechanism, because Arduino clock is accurate for approximately 70 minutes and our experiment run for only two and half minute, so there is no need for clock drift management. Because FTSP is a very simple protocol, there was no other thing to simply. Now, we will explain full FTSP implementation with an example that will include root selection, root failure and time synchronization.

Message structure for proposed FTSP and our FTSP is same. We have a "*nodeID*", "*timestamp*", and "*messageID*"". One assumption is that network is connected. To make sure this assumption is true, we wait for some fixed amount

of time before broadcasting any message. In our implementation, root node is a function of time and node id. Whichever node has smallest node id, it will become root node. In case of FTSP, there is no peer-to-peer communication, so nodeIDs can be assigned either dynamically or statically. After each node is assigned a nodeID. Each node waits for some amount of time equal to its nodeID. Now, the node that have smallest ID will start broadcasting timing information first, thus becoming the root node. When root node broadcast first synchronization message, it will contain root nodeID, timestamp and a unique messageID. Root node also adds this message to its message table for future duplicate message checking. This broadcast message will be heard by all nodes in its radio range. Those nodes will update their timing information, adds this message to their message table and rebroadcast the message without any change. Message table contains all three parameters of the message. This message will propagate to the end of the network. But for certain, each node will hear the message many times, if we do not discard the duplicate messages, these messages will remain in network till the end. To avoid infinite message loop, when the message is again heard by the node, each node will check in its message table, if the same message exists, it will just increase the message count and discard the message thus breaking the infinite message loop. Table 4.1 shows message structure of FTSP.

Field	Description
nodeID	Root Node ID.
messageID	ID of the message.
timestamp	Reference time.

Table 4.1: Structure of a FTSP Message

In case of root node failure, if nodes does not receive any messages for "T" time, than they will clear their root node and message table or in other words they will reset everything and again same process will be repeated. There is a very minor chance of having two root nodes in the network, which is ignorable.

As mentioned before, this is a very simple broadcasting protocol with a simple message structure and reliable time synchronization method, but there is a problem with this approach. As messages travel further away from the root node, there is no compensation for propagation delay, edge nodes will not be exactly synchronized with the root node. To compensate for this problem, RTSP utilizes request and reply mechanism, as explained in section 2.9 and we will further analyze in the following section.

4.2 Recursive Time Synchronization Protocol

Recursive Time Synchronization Protocol (RTSP) is complex and difficult to implement protocol. Backbone of the algorithm is request-reply mechanism for time synchronization. Request-reply mechanism guarantees propagation delay compensation at destination node. RTSP have three type of messages i.e. ERN, REQ and REP. ERN messages are for discovery and root selection, REQ are time request messages and REP are reply messages. There are many other mechanisms employed by RTSP. For example, root selection, failure recovery, energy compensation, clock drift management, etc. As explained earlier, we have simplified these protocols due to hardware limitations. Now, we will explain our implementation

of RTSP.

RTSP message [68] consists of ten elements as shown in table 4.2.

Field	Description
msgType	Type of the message: ERN, REQ, and REP.
msgID	ID of the message.
originID	ID of the node that originated the message.
imSrcID	ID of immediate source node that forwarded the message.
imDestID	ID of immediate destination for message forwarding.
refNodeID	ID of the reference node known (-1 for ERN enquiry).
T1	Local time when the message was sent or forwarded.
T2	Local time when the message was received.
T3	Local time when reply was sent or forwarded.
Tr	Reference time when the reply was sent or forwarded.

Table 4.2: Structure of a RTSP Message

RTSP is implemented at application as well, and it also requires peer-to-peer communication to accurately synchronize. But it is not possible to dynamically determine the address of incoming message node and reply to that address using Arduino XBee. In our RTSP implementation, root selection is a function of time and node id. So, in this case we cannot assign node id's dynamically, because we need to reply each node directly rather than via broadcast method and also reply message need to follow the same path as of request message. So, each node have a complete table of all the node ids and their corresponding network addresses.

Root selection is a static process in our case, thus eliminating some complexity, but this creates another problem. In the original protocol, after root selection whole network knows, which node is the root node. But in our case, although process is static, but nodes do not know which node is the root node, because only after their timers are expired, root node will start transmitting messages

and network will know this node is the root node. For this protocol to work, we need a request and a reply message without these it will not work. There could be multiple ways to solve this problem. But we have used a very simple method. When each node power up, they will broadcast a single request message. Now unlike FTSP, there will not be any rebroadcasts, because any node which hears this message will do the following: It will check weather it has sent its own request message or not, if not than it will store this message in a queue and send its own request message, otherwise it will just store the message and wait for a root reply. There is a duplicate message checking before any of that happens. This way there will be very few messages in the network. Now, nodes will wait for their local timers to expire. Whichever node's timer expire first will start replying to other nodes. Each node would have at least one node in the queue for clock synchronization. Node will obtain destination address from local network address table and reply to that node. Destination node will check weather it has already received a reply from some other node or not, if the reply is already been received, it will just ignore the message. But there could be a case, both nodes would have received request message from each other. In that case, recipient node never reply to replying node or source node.

There are no mechanism for energy compensation or clock drift due to hardware limitations unlike the originally proposed protocol. In case of node failure, it follows the same mechanism as FTSP. If nodes does not receive a message for "T" duration, they reset all parameters and starts timer again for root selection.

CHAPTER 5

RESULTS AND DISCUSSION

In this chapter, we will discuss the results of experimentation. We have made four topologies i.e. bus, grid, mesh and tree to compare both protocols. The parameters used for performance evaluation are, number of messages required by each node for one synchronization cycle, bandwidth utilization and convergence time. For each topology, experiment is repeated five times for both protocols. Network is fully connected and there exist a path from each node to every node. Path could be direct or indirect. Root selection is a function of time and node id. Whichever node has smallest id, it will become root and start transmitting timing information. For example, if there are three nodes with node IDs 5, 10, and 15, each node would have to wait for amount of time equal to their node id i.e. node with node id 5 would have to wait for 5 seconds before it could declare itself a root node than rest of the nodes will hear timing information and update their clocks according to reference time received from root node as well as their root node status. Node IDs are not random they are assigned manually. By using

combination of time and smallest id for root selection, we are almost free of having two root nodes in the network. There are no regular update message, because we are more interested in first time full network synchronization rather than later updates of each protocol. Table 5.1 shows the message length of each protocol.

Field	RTSP	FTSP
msgType	1	-
msgID	1	1
originID	1	1
imSrcID	1	-
imDestID	1	-
refNodeID	1	-
T1	10	-
T2	10	-
T3	10	-
Tr	10	10
Total (Bytes)	46	12

Table 5.1: Message Bytes Table

5.1 Bus Topology

Bus topology is implemented for both protocols. Eleven nodes were placed in a row with node 1 (root node) at the top and node 11 at the bottom end. Each experiment lasted for 150 seconds after which each node reset its local clock and repeated same process of time synchronization for five more times. All information regarding root node and number of messages is transmitted to a single sink node, which is not a part of the experiment, a separate node for data collection only. As a secondary measure, data is also saved on the Electrically Erasable Programmable Read-Only Memory (EEPROM) of each Arduino Mega. In case, if some data

packets to sink node (collector node) are lost, as a backup measure, we would have that information saved onto the EEPROM of Arduino. Figure 5.1 shows physical bus topology configuration for both protocols.

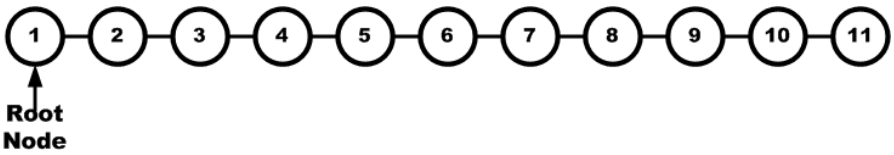


Figure 5.1: Bus Topology

As mentioned above, these protocols have been implemented at application layer. Any message that comes to application layer, whether it is a broadcast/re-broadcast (echo message) message or request/reply message, it will be counted in the number of messages metric.

In Figure 5.2 and 5.3, Horizontal axis show the number of nodes and on the vertical axis number of messages. As mentioned earlier, for each topology experiment was repeated five times. Average of all five experiments for each node is also shown. Ideally, number of messages should be liner for node 2-10, because they are connected with exactly 2 nodes. For example, node 3 will be connected with node 2 and node 4, but in reality this is not the case, it could be connected with more than two nodes. There are many factors in play. This is a wireless medium, so there could be a collision of messages, XBee might not be in the state of receiving messages or multiple messages arrive at the same time. Another very important factor is the state of 9V battery powering Arduino Mega and XBee. Now each node have an independent 9V battery, so each battery would have different power

level remaining in it. If one battery have less power remaining it would reduce its reception and transmission range and vice versa.

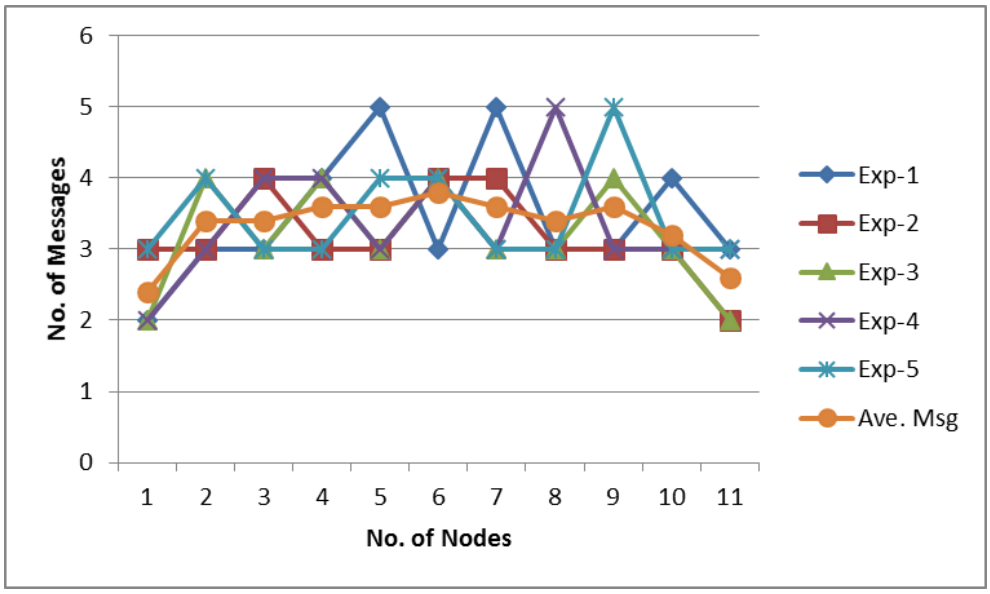


Figure 5.2: FTSP Bus Topology - Number of Messages per Node for each experiment

Figure 5.2 shows bus topology for the Flooding Time Synchronization Protocol. For each node, number of messages varies from 2 to 5, where in FTSP at least two messages are required by each node for synchronization. But if we take average of five experiments for each node, it almost becomes linear. As it can be seen in Average Message ("Ave.Msg") line, it's almost linear except for node 1 and 11, because they are directly connected with only one node i.e. node 2 and 10, respectively.

Consider an example with only two nodes in the network. Node 1 with id 5 and Node 2 with id 10. Both nodes will turn on their local clock timers and wait for them to expire. Root node is a function of time and node id. So, Node 1 timer

will expire first, because it has node id 5. It will broadcast a message containing node id, message id and reference time. Node 2 will hear the message and checks if received message node id is smaller than its own, it will assume remote node as root node and update its local clock according to received time as well as store a copy of message in its memory for future duplicate message checking. Node 2 will rebroadcast the received message without changing anything in it. As mentioned earlier, this broadcast message will be again heard by the root node, because protocols are implemented at application layer, so all broadcast messages always comes to application layer. But root node would already have a copy of this message in its memory (because message was stored when it broadcasted first time), thus it will discard the message, but will include in the message count. From this example we can see that two message send/received by each node, one broadcast message and other echo message (rebroadcasted message). FTSP with two or more nodes requires at least two messages for synchronization.

Figure 5.3 shows bus topology for the Recursive Time Synchronization Protocol. As we can see, number of messages varies from as low as 3 and as high as 9, but average behaviour of nodes over five runs is linear except for node 1 and 11 due to the reasons explained above. RTSP have many fluctuations as we can see in Figure 5.3. Few reasons are explained in the FTSP bus topology section, but there is another very important reason. The working of RTSP is very different from FTSP. FTSP is a very simple protocol as compared to RTSP. Most prominently difference is RTSP is suitable for only clustered/Tree topology. On

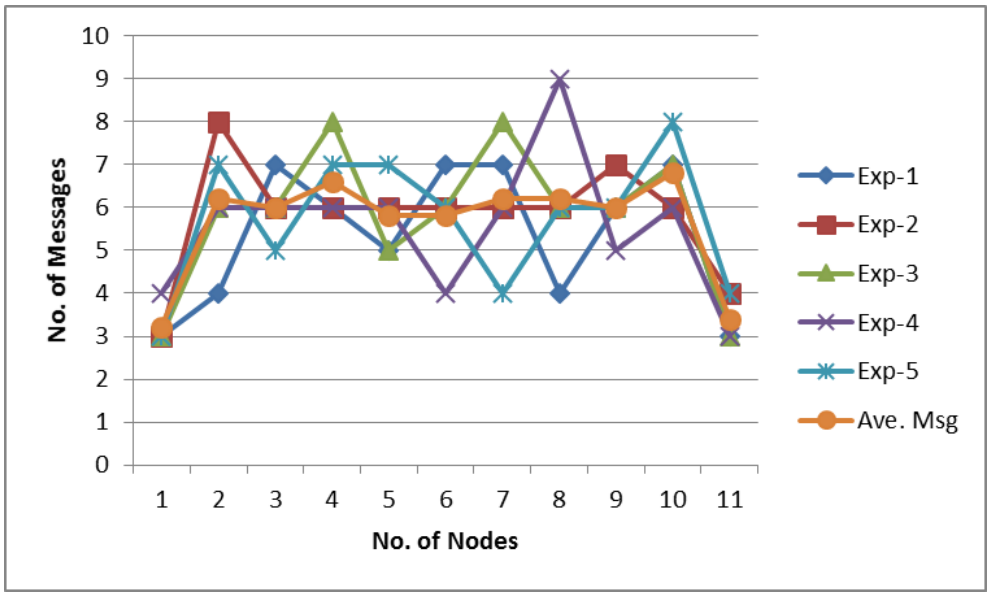


Figure 5.3: RTSP Bus Topology - Number of Messages per Node for each experiment

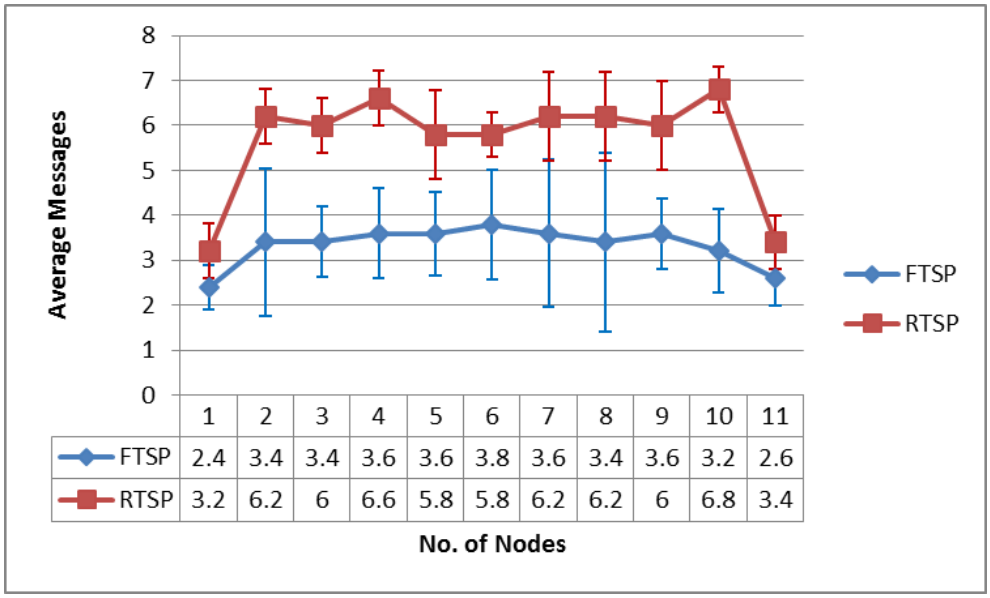
the other hand, FTSP can work in almost all types of topologies for a small to medium size network.

As aforementioned, we have done few simplifications in both algorithms, one of the simplification for RTSP is elimination of root election, because now root selection is a function of time and node id, so there is no need for root elections. But this creates another problem, RTSP heavily rely on request and reply mechanism for time synchronization, now there is no unique message for just root announcement instead whichever node becomes root will reply to other nodes. To solve this problem, we just simply broadcast request message as soon as a node power up. Now, each node is a clusterhead, there is no echo broadcasts, because if any other node listens the message, it will simply check if i am root or i am synchronized than it will reply to the requesting node with timing information or if i am not

synchronized and i have sent a request message for time synchronization, it will simply store the request message and wait for its own synchronization reply than to the requesting node. Consider the following example.

Suppose that we have a fully connected network of 3 nodes with node 5, node 10 and node 15. As soon as nodes power up, they will turn on their local clocks and send a time request message (it will be a broadcast message). Node 5 will hear request messages from node 10 and 5, node 10 will hear messages from node 5 and 15, and node 15 will hear messages from node 5 and 10. So each node would send/receive 3 messages. Now, they wait whichever node's timer expire first it will become root and will reply to rest of the connected nodes. In this example, node 5 becomes root and replies to node 10 and 15. Although node 10 and 15 received request message from node 5, but they will not reply back, because they have received reply message from same node instead they will reply to each other (Nodes never reply to the node from which they received reply). Now, root node has message count of 5 and other two nodes have message count of 6. If network have only two nodes than message count would be 3, so it require at least 3 messages/node for synchronizing two node network. Hence, this fluctuation is due to request and reply mechanism. May be some nodes received more request messages as compared to other or some node have replied to more node as compared to other nodes. But in the long run this factor also does not matter, because average line for five run is almost linear with the exception of node 1 and node 11, because they are connected with one node only, and rest of

the nodes are connected with two or more nodes.



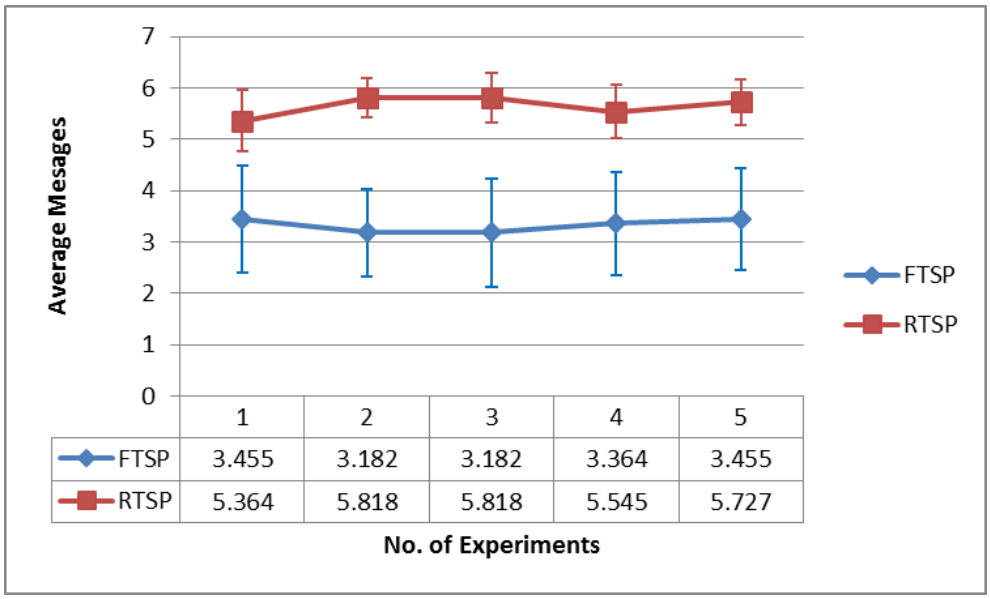
	1	2	3	4	5	6	7	8	9	10	11
FTSP	2.4	3.4	3.4	3.6	3.6	3.8	3.6	3.4	3.6	3.2	2.6
RTSP	3.2	6.2	6	6.6	5.8	5.8	6.2	6.2	6	6.8	3.4

Figure 5.4: Bus Topology - Average Number of Messages per node for five experiments with 95% confidence interval

In Figure 5.4, a comparison of both protocols for bus topology have been presented. In this figure, average of five experiments for each node has been shown. As expected RTSP have much higher message count as compared to FTSP, because inherently, it requires more number of messages/node to synchronize.

Each node's behaviour varies in each experiment, but it can be seen in Figure 5.5, its almost linear over the course of five experiments for each protocols and there is not much variation between both protocols for experiments average messages.

Second metric for comparison is bandwidth. Bandwidth is different from number of messages. But, it is also the function of number of messages, because as explained earlier, we cannot truly calculate the bandwidth used by these devices,

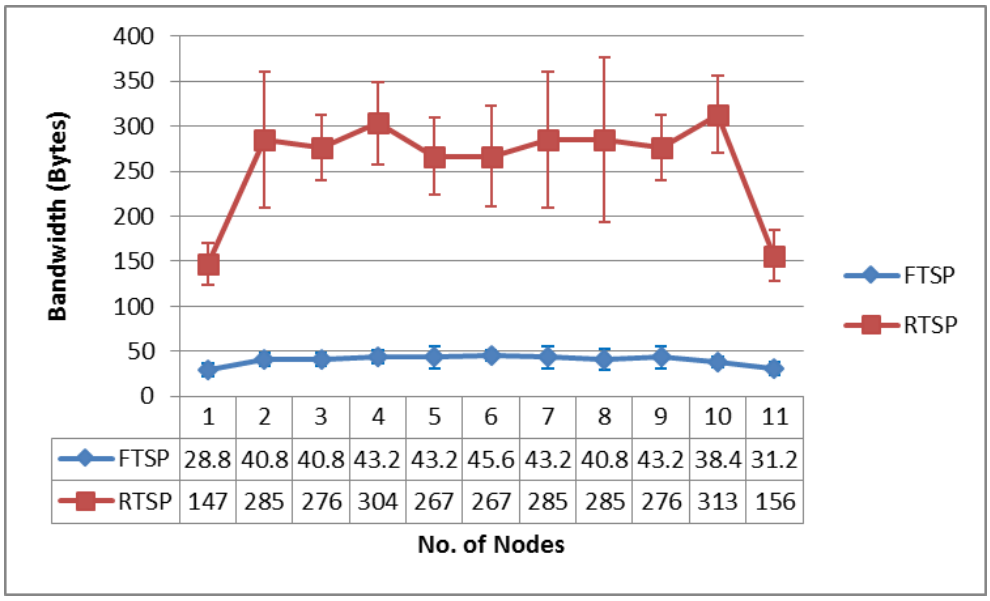


	1	2	3	4	5
FTSP	3.455	3.182	3.182	3.364	3.455
RTSP	5.364	5.818	5.818	5.545	5.727

Figure 5.5: Bus Topology - Average Number of Messages for each experiment with 95% confidence interval

so we have assumed that their data packet size equal to bandwidth. As we know,

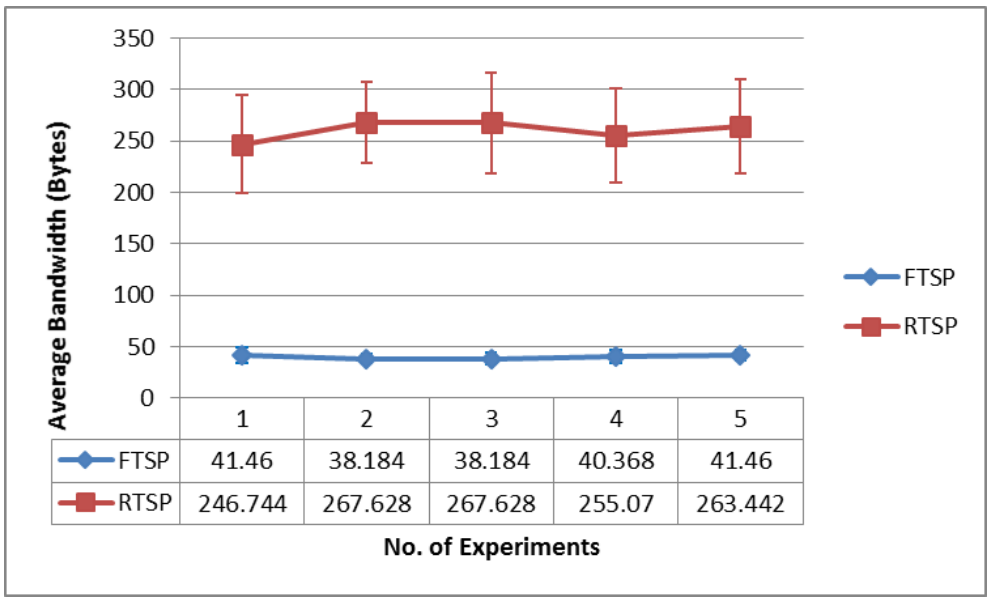
FTSP packet consists of 12 bytes and RTSP packet consists of 46 bytes.



	1	2	3	4	5	6	7	8	9	10	11
FTSP	28.8	40.8	40.8	43.2	43.2	45.6	43.2	40.8	43.2	38.4	31.2
RTSP	147	285	276	304	267	267	285	285	276	313	156

Figure 5.6: Bus Topology - Average Bandwidth per node for five experiments with 95% confidence interval

We multiply average message and experiments average messages with FTSP and RTSP packets sizes accordingly and we get the average bandwidth for each node and each experiment accordingly. As shown in Figure 5.6 and 5.7.



	1	2	3	4	5
FTSP	41.46	38.184	38.184	40.368	41.46
RTSP	246.744	267.628	267.628	255.07	263.442

Figure 5.7: Bus Topology - Average Bandwidth for each experiment with 95% confidence interval

Now the difference of bandwidth between both protocols is more than 4 times. The main reason is inherent larger packet size of RTSP and also it requires more number of messages for synchronization. In RTSP, node 1 and 11 have the lowest bandwidth consumed, because they are edge nodes and have much less interference as compared to rest of the nodes, which are directly connected with at least 2 nodes as compared to 1 node.

Now, if we take a general view of both metrics, FTSP clearly performs much better than RTSP. We can also deduce from these results that FTSP will consume much less energy (power) as compared to RTSP, because transmission/reception

energy will be conserved in case of FTSP due to lower bandwidth.

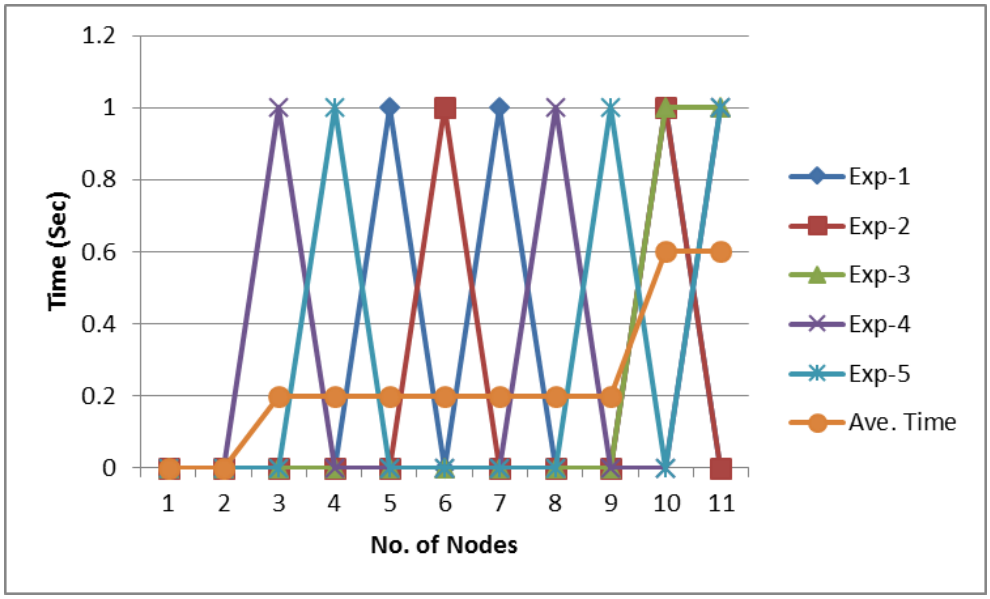


Figure 5.8: FTSP Bus Topology - Convergence Time per Node for each experiment

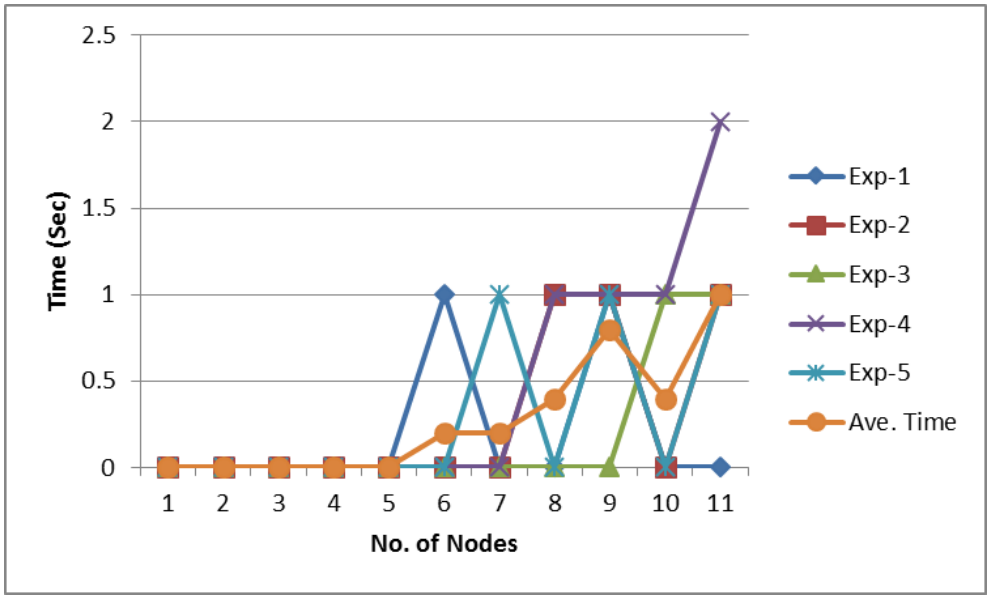
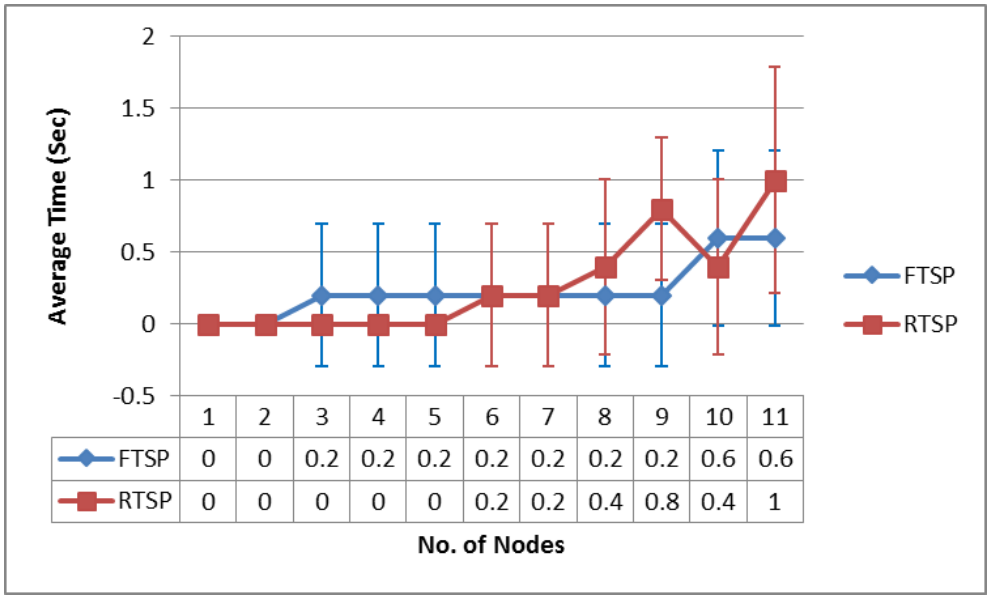


Figure 5.9: RTSP Bus Topology - Convergence Time per Node for each experiment

Third metric used for comparative analysis is the convergence time. Figure 5.8

and 5.9 shows convergence time for FTSP and RTSP, respectively. As we can see, convergence time for FTSP is very low. At maximum it goes to 1 second for some nodes. On the other hand, RTSP has maximum of 2 second convergence time for node 11. In a best case scenario, convergence time effect should be commutative for the all the nodes in bus topology. For example, if node 3 has a 1 second delay, than this effect should reach till the node 11. So convergence time for the rest of the nodes would be 1 second or greater. But this is not the case here, because there is some interference between nodes. Node 3 message can be heard by node 1 and node 5 (most probably). Due to this reason, convergence time is not consistent among the nodes.

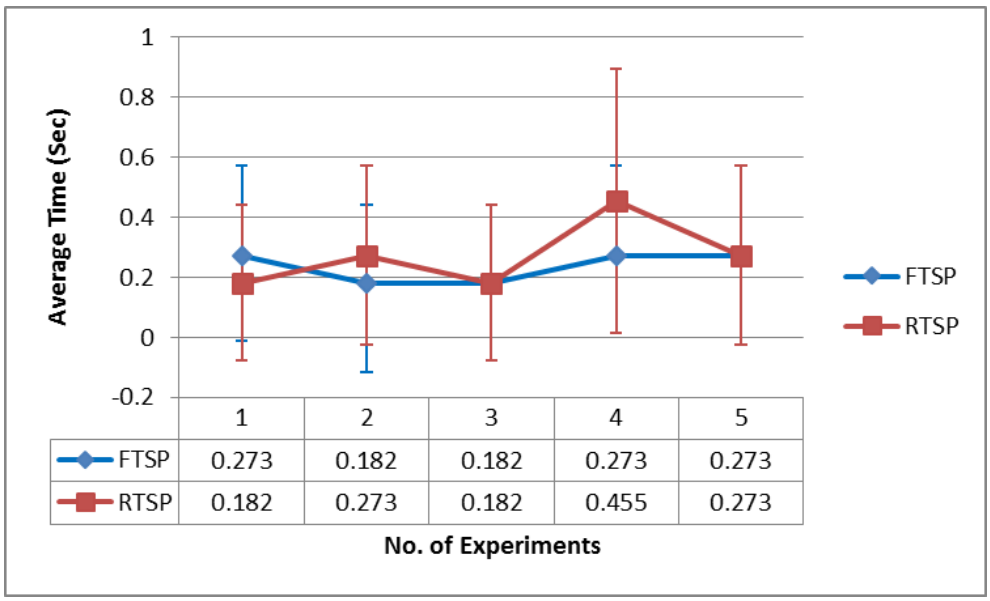


	1	2	3	4	5	6	7	8	9	10	11
FTSP	0	0	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.6	0.6
RTSP	0	0	0	0	0	0.2	0.2	0.4	0.8	0.4	1

Figure 5.10: Bus Topology - Average Convergence Time per node for five experiments with 95% confidence interval

In Figure 5.10 average convergence time of each node over five experiments is shown. As we can see, convergence time for FTSP is low as well as consistent for

most of nodes. But in case of RTSP, convergence time is zero for initial nodes but goes up to 1 second for tail nodes. The same phenomenon happens in case of average convergence time of each experiment. FTSP performs better in this case as well, but RTSP is not bad either. Both protocol have convergence time less than 500 milliseconds.



	1	2	3	4	5
FTSP	0.273	0.182	0.182	0.273	0.273
RTSP	0.182	0.273	0.182	0.455	0.273

Figure 5.11: Bus Topology - Average Convergence Time of each experiment with 95% confidence interval

These millisecond differences could be due to various reasons. For example, when the clock resets, there will be some millisecond difference between nodes due to propagation delay. Hardware clock of individual nodes also play some part in it. In case of convergence time, both protocols perform comparative to each other.

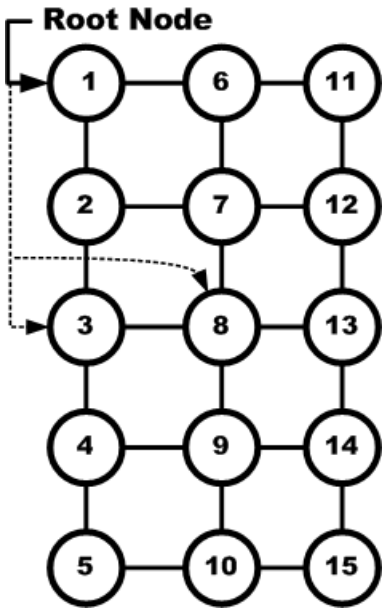


Figure 5.12: Grid Topology

5.2 Grid Topology

Grid configuration makes an interesting topology, because root node could have three different locations in the grid as shown in Figure 5.12. Fifteen (15) nodes were used for grid topology in a 5x3 configuration (5 rows, 3 columns). In the first configuration, root node was placed at location (1,1), i.e node 1. In this configuration, root node is directly connected with 2 nodes i.e. node 2 and 6. In the second configuration, root node was placed at (3,1), i.e node 3 and it was directly connected to nodes 2, 4 and 8. In the last configuration, root node was placed at (3,2) i.e node 8, where it was directly connected with 4 nodes i.e. nodes 3,7,9 and 13. The most definite change would be in number of messages, but most interesting would be in the convergence time. That was our assumption before experimentation, once we started the experimentation, it became clear that

former is true, but latter is not. So, for this topology only one configuration result is shown i.e. configuration (1,1). It does not make much difference in either of the configurations for number of messages metric, because let's suppose root node is at (3,2) location, number of messages for node 7, 8 and 9 would be approximately same, because all of the nodes are connected with 4 nodes on either sides, same is true for rest of the configurations. The only difference, as mentioned before would have been convergence time, because when node is in the center, synchronization messages would take less amount of time, but that may be the case for a much larger area. Speed of communication is also fast enough for our experimentation that it makes no difference where we place root node. Figure 5.13, 5.14 shows FTSP and RTSP grid topology results for number of messages metric for five experiments, respectively.

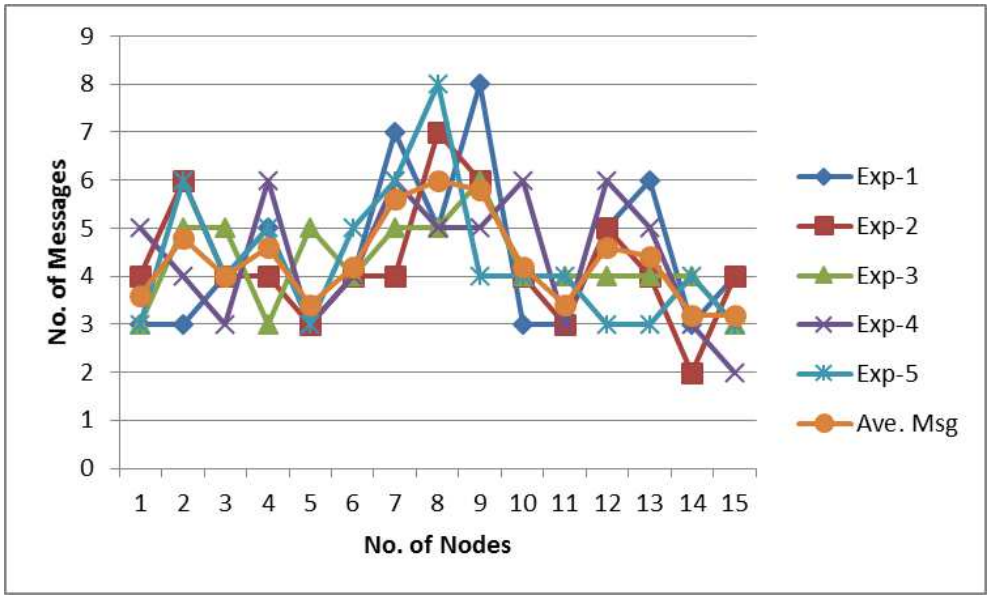


Figure 5.13: FTSP Grid Topology - Number of Messages per Node for each experiment

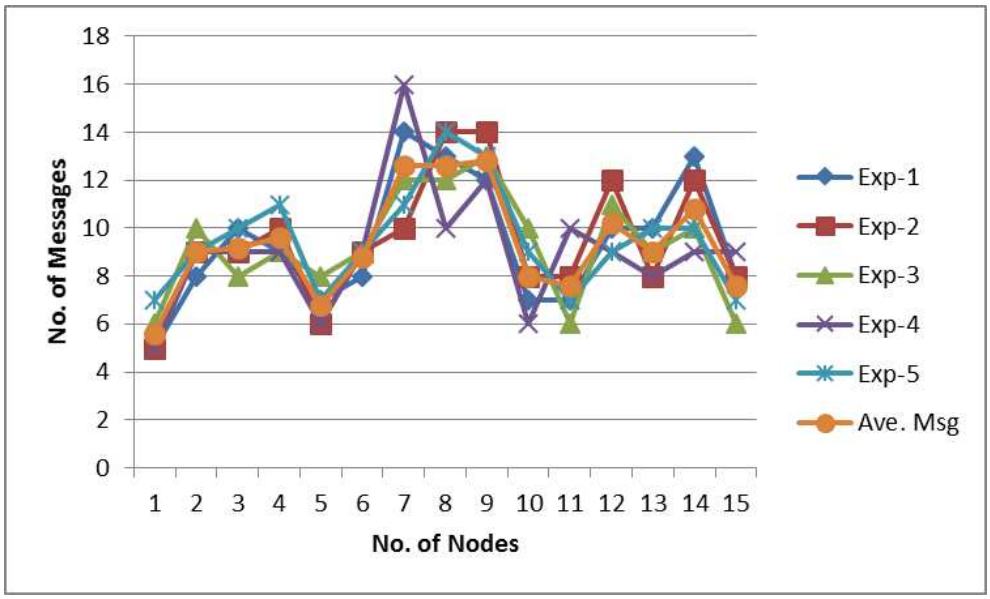
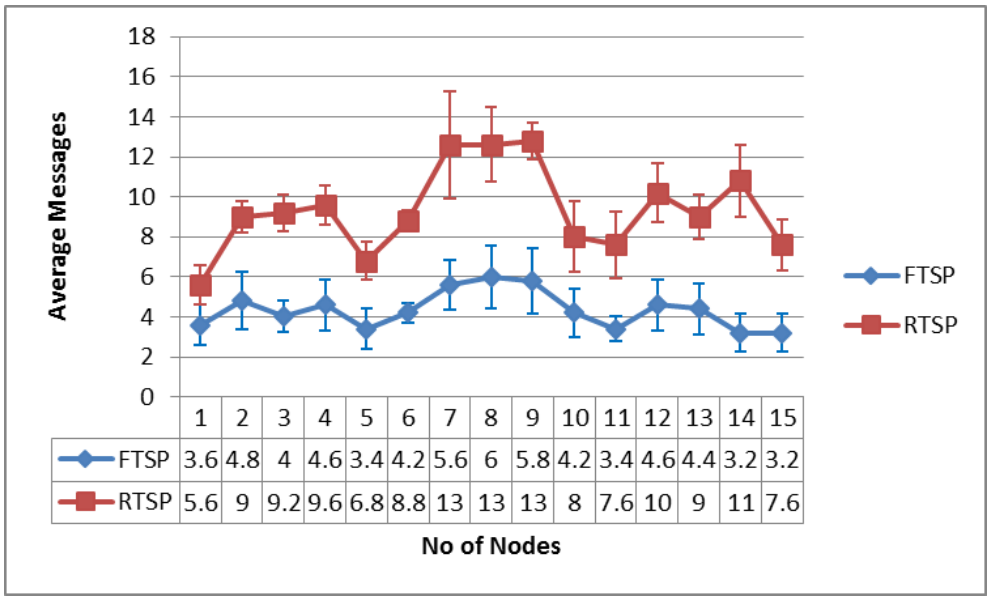


Figure 5.14: RTSP Grid Topology - Number of Messages per Node for each experiment

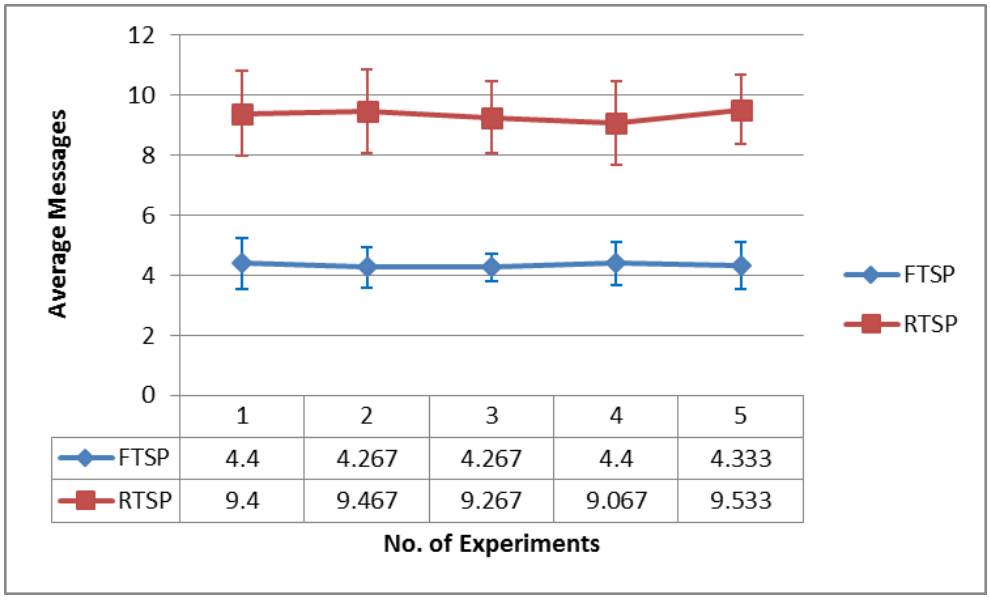


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FTSP	3.6	4.8	4	4.6	3.4	4.2	5.6	6	5.8	4.2	3.4	4.6	4.4	3.2	3.2
RTSP	5.6	9	9.2	9.6	6.8	8.8	13	13	13	8	7.6	10	9	11	7.6

Figure 5.15: Grid Topology - Average Number of Messages per node for five experiments with 95% confidence interval

Figure 5.15 shows the average messages for each nodes for five experiments. We can see the behaviour of the nodes here. Nodes that are directly connected to

more nodes have higher average number of messages i.e. nodes directly connected with 4, 3 and 2 nodes. Most obvious observation would be FTSP performs better than RTSP in this topology for very simple reason, FTSP requires less number of messages for synchronization as compared to RTSP.

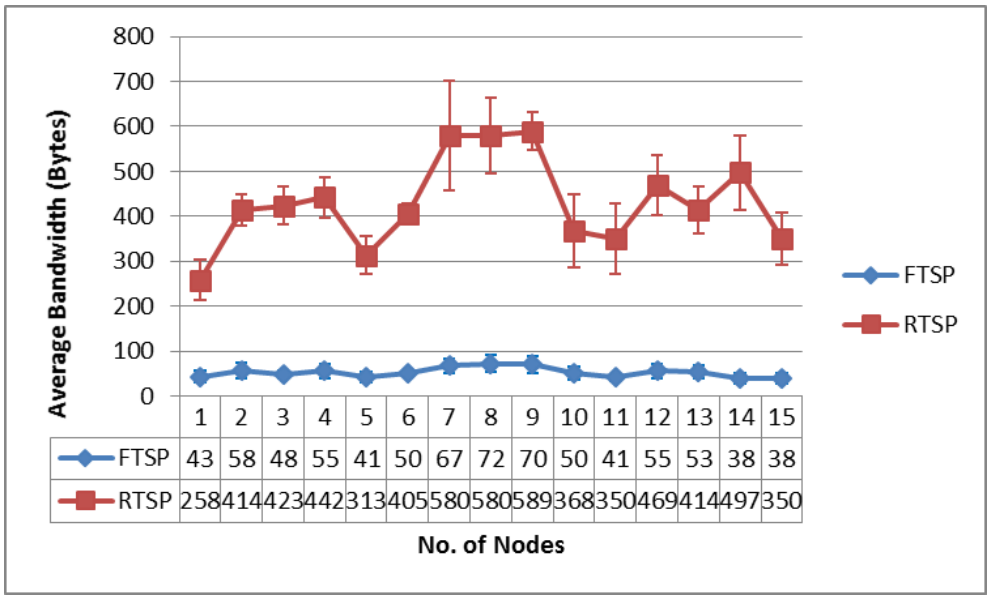


	1	2	3	4	5
FTSP	4.4	4.267	4.267	4.4	4.333
RTSP	9.4	9.467	9.267	9.067	9.533

Figure 5.16: Grid Topology - Average Number of Messages for each experiment with 95% confidence interval

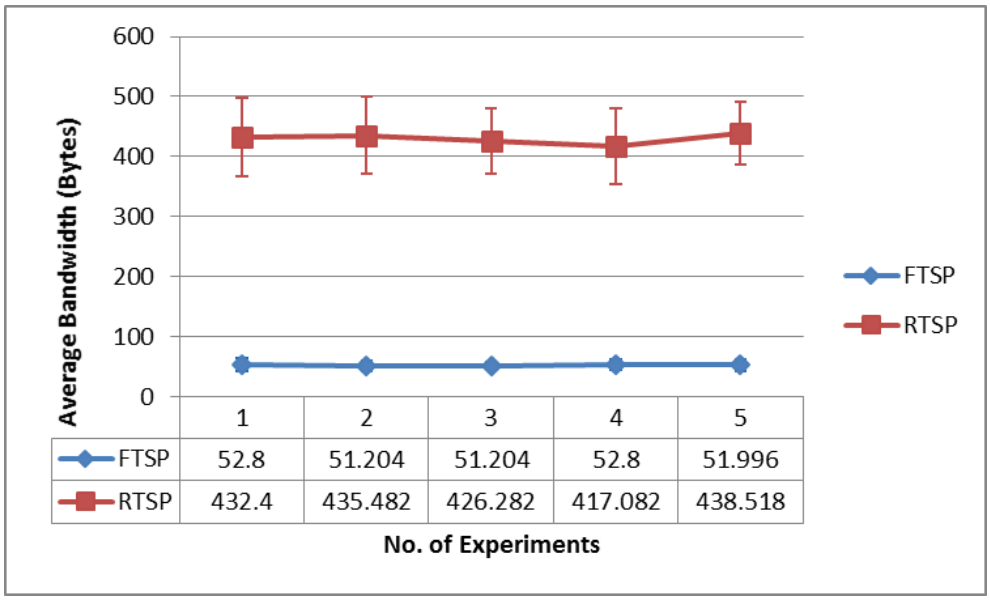
Both protocols behave linearly for average number of messages for each experiment as shown in Figure 5.17. But the difference between both protocols is almost double. RTSP perform much worst in this case. This behaviour also suggests that RTSP does not perform efficiently, when connected to large number of nodes. On the other hand, FTSP performs much better in this case.

Figures 5.17, 5.18 shows average bandwidth per node for five experiments and average bandwidth of each experiment, respectively. Behavior of the Bandwidth is also similar, because bandwidth is a function of number of messages, but it does



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FTSP	43	58	48	55	41	50	67	72	70	50	41	55	53	38	38
RTSP	258	414	423	442	313	405	580	580	589	368	350	469	414	497	350

Figure 5.17: Grid Topology - Average Bandwidth per node for five experiments with 95% confidence interval



	1	2	3	4	5
FTSP	52.8	51.204	51.204	52.8	51.996
RTSP	432.4	435.482	426.282	417.082	438.518

Figure 5.18: Grid Topology - Average Bandwidth for each experiment with 95% confidence interval

show how big the difference between both protocols for bandwidth consumption is. Clearly, RTSP is not very economical solution in this case. Actually, it is an

expensive solution for grid topology and it remains consistent for all experiments. There is not a single case, where we can say that RTSP is a better or comparable solution for this particular topology.

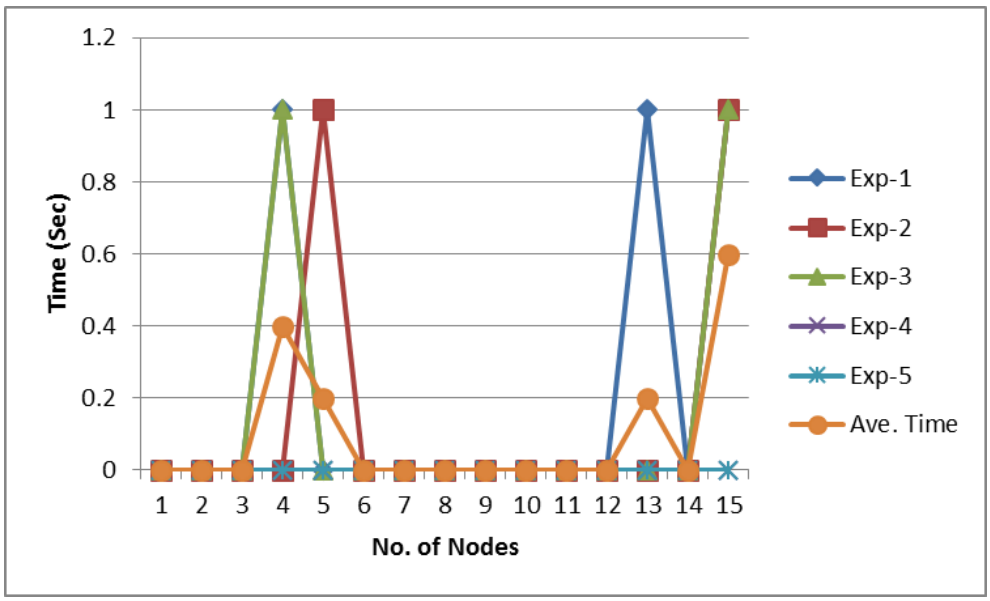


Figure 5.19: FTSP Grid Topology - Convergence Time per Node for each experiment

Last metric is the convergence time. Figures 5.19 and 5.20 shows convergence time for Flooding Time Synchronization Protocol and Recursive Time Synchronization Protocol, respectively. As we can see, there is great amount of fluctuations in RTSP convergence time. There could be several reasons e.g. protocol is not well equipped for this configurations, hardware clocks, processing delays, etc. On the other hand, FTSP has much less fluctuations and much more suited for this kind of environment. As aforementioned, only one configuration results are shown here i.e. (1,1). The results for rest of the grid configurations were more or less similar. Due to this reason, those results are not shown here.

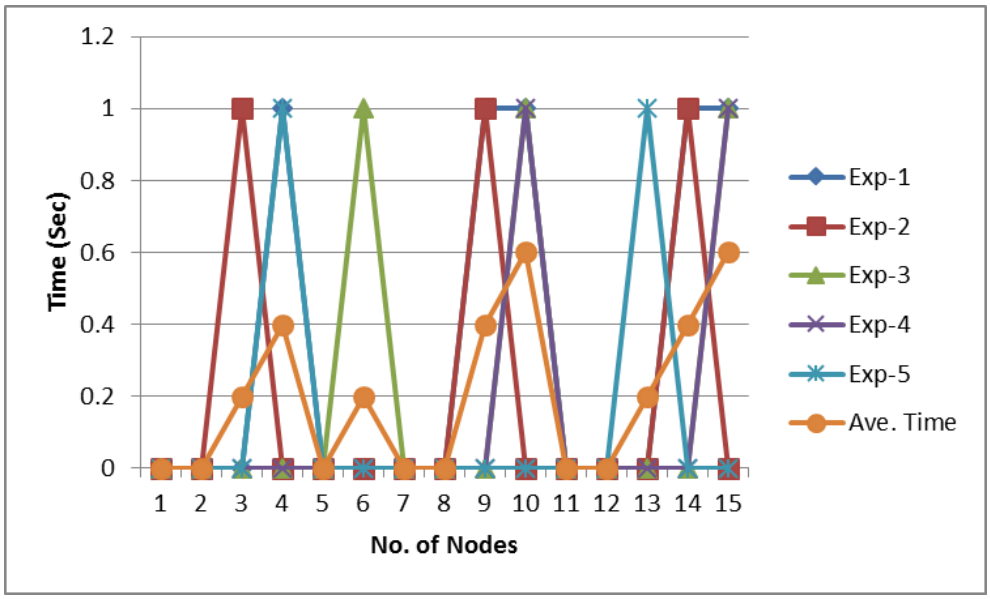
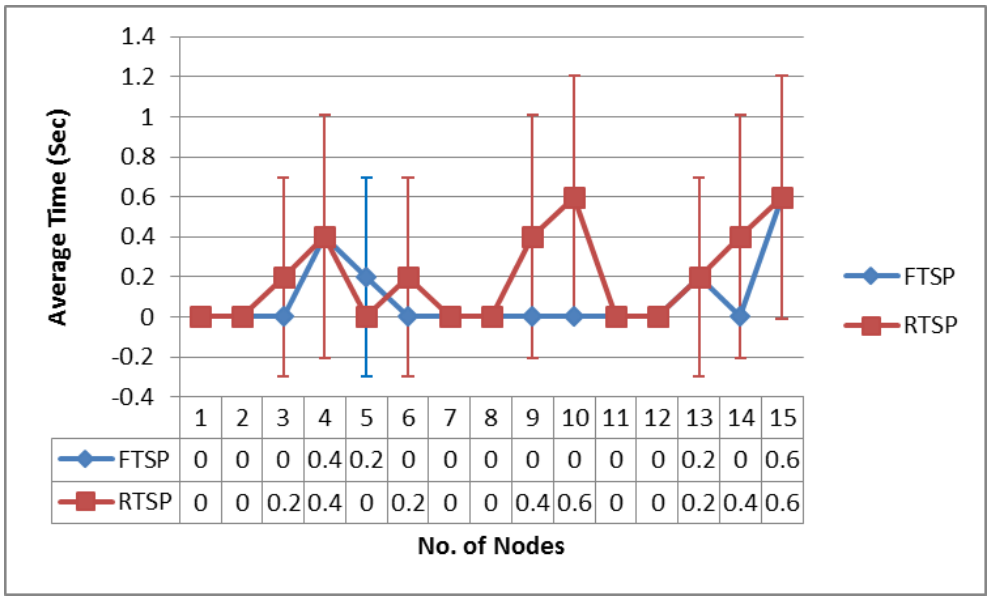


Figure 5.20: RTSP Grid Topology - Convergence Time per Node for each experiment



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FTSP	0	0	0	0.4	0.2	0	0	0	0	0	0	0	0.2	0	0.6
RTSP	0	0	0.2	0.4	0	0.2	0	0	0.4	0.6	0	0	0.2	0.4	0.6

Figure 5.21: Grid Topology - Average Convergence Time per node for five experiments with 95% confidence interval

Figures 5.21, 5.22 shows average convergence time of each node and each experiment, respectively. Average convergence time of each for RTSP is consistent for

five experiments i.e. remains worst. On the other hand, FTSP consistently performs better throughout the experimentation. Average convergence time of each experiment shows interesting results. Convergence time starts high, but then they start to reduce for both protocols. May be this is due to better connectivity of network for last experiments or there could be several other factors contributing to that.

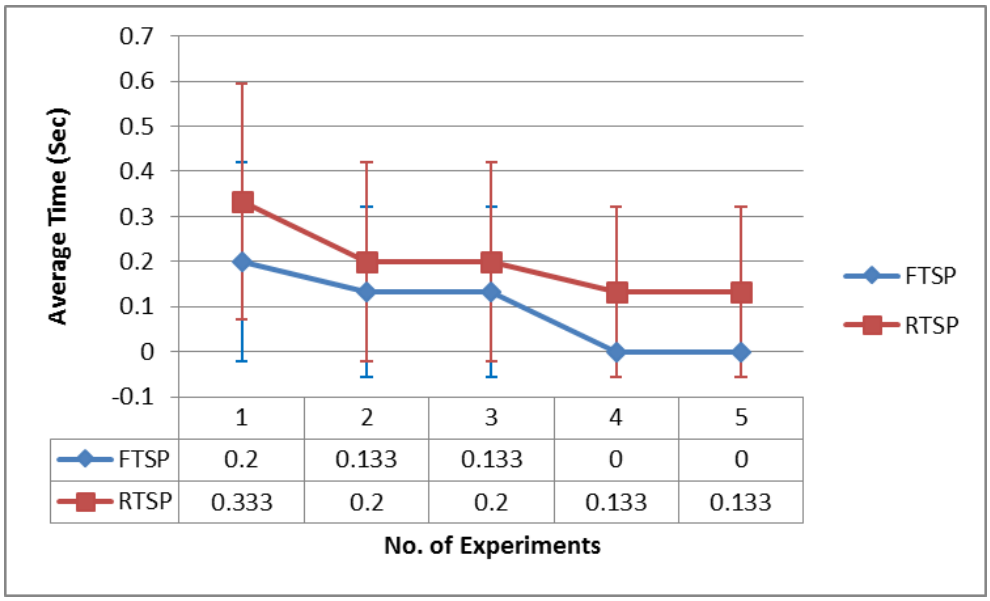


Figure 5.22: Grid Topology - Average Convergence Time of each experiment with 95% confidence interval

5.3 Mesh Topology

Third topology used for experimentation is mesh. It's a fully connected mesh topology, where each node is directly connected with all the other nodes. Nodes were placed in a small area, within the antenna range of each node. Due to

fully connected mesh topology, large number of messages are exchanged between nodes. Highest number of messages among four topologies for both protocols. Figures 5.23 and 5.24 shows number of messages/node for five experiments for FTSP and RTSP, respectively. As we can see, RTSP is at its worse in the mesh topology as well as FTSP, but FTSP still performs much better than RTSP. RTSP is going at a peek of almost 43 messages for a single synchronization cycle. It is clearly not very feasible from any aspect.

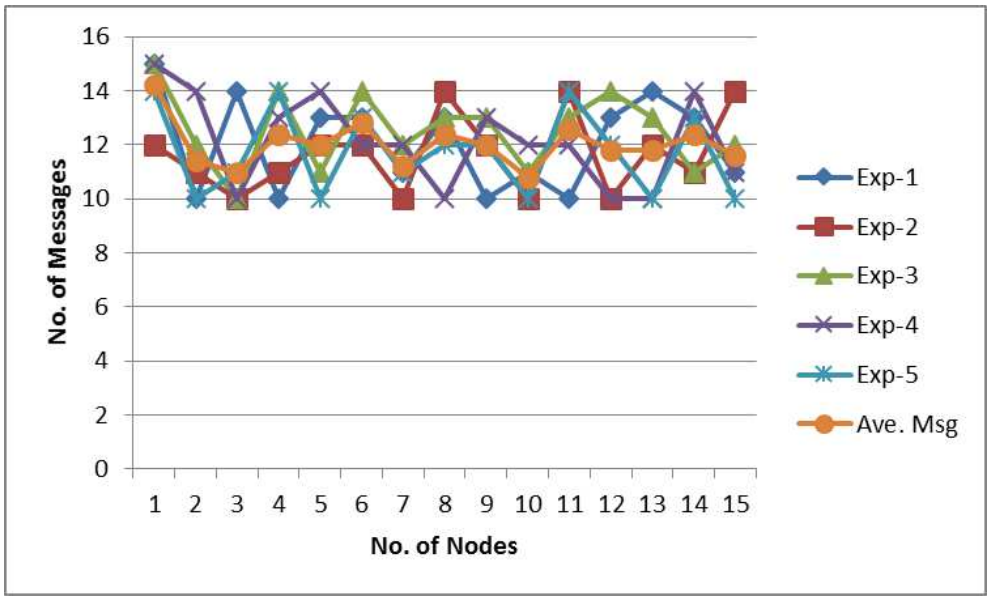


Figure 5.23: FTSP Mesh Topology - Number of Messages per Node for each experiment

Figure 5.25 shows average messages per node for five experiments. The difference between both protocols is more than three folds. RTSP requires 3 times more messages than FTSP for one synchronization cycle. On the other hand, FTSP performs significantly better. The main reason for both protocols performing so poor in comparison to other topologies is that they have to deal with more number

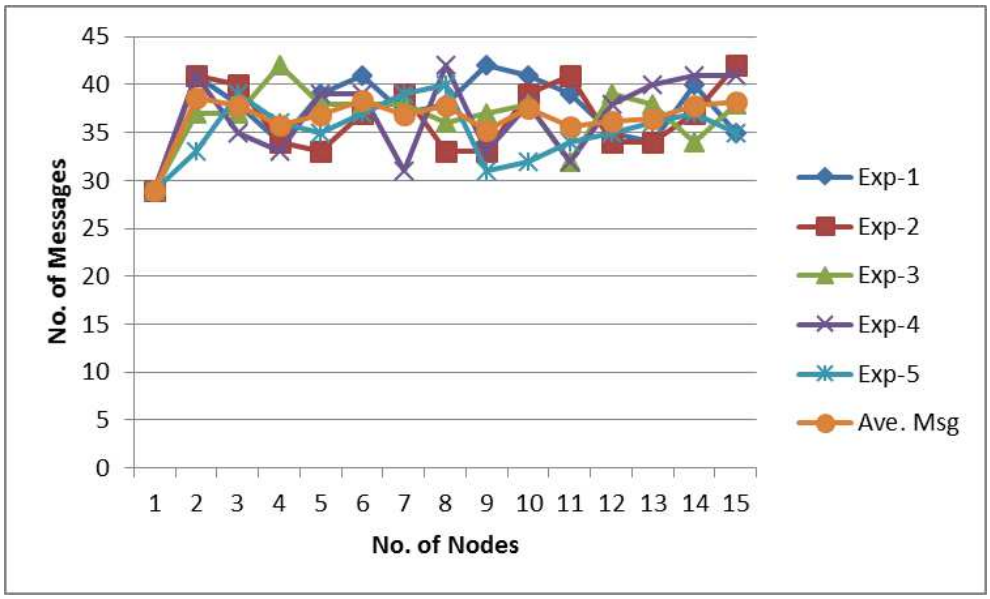
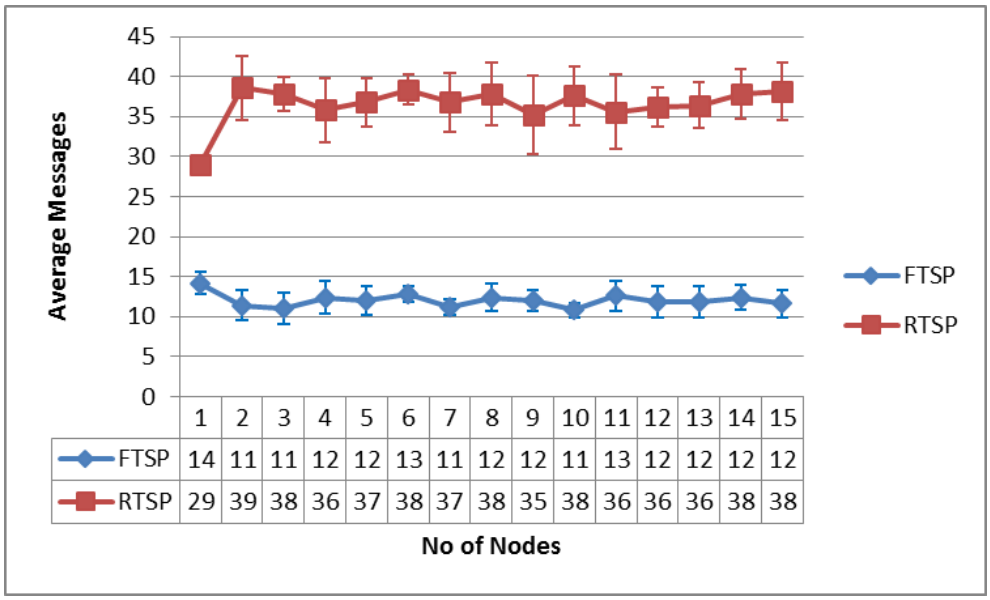


Figure 5.24: RTSP Mesh Topology - Number of Messages per Node for each experiment



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FTSP	14	11	11	12	12	13	11	12	12	11	13	12	12	12	12
RTSP	29	39	38	36	37	38	37	38	35	38	36	36	36	38	38

Figure 5.25: Mesh Topology - Average Number of Messages per node for five experiments with 95% confidence interval

of nodes per cycle as compared to other topologies, where maximum number of directly nodes were only 4.

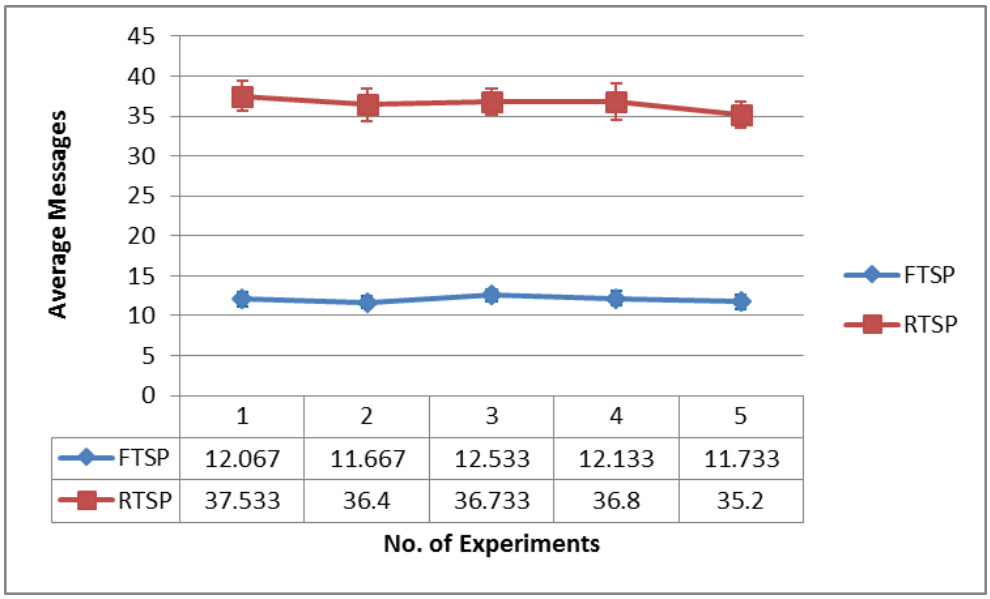
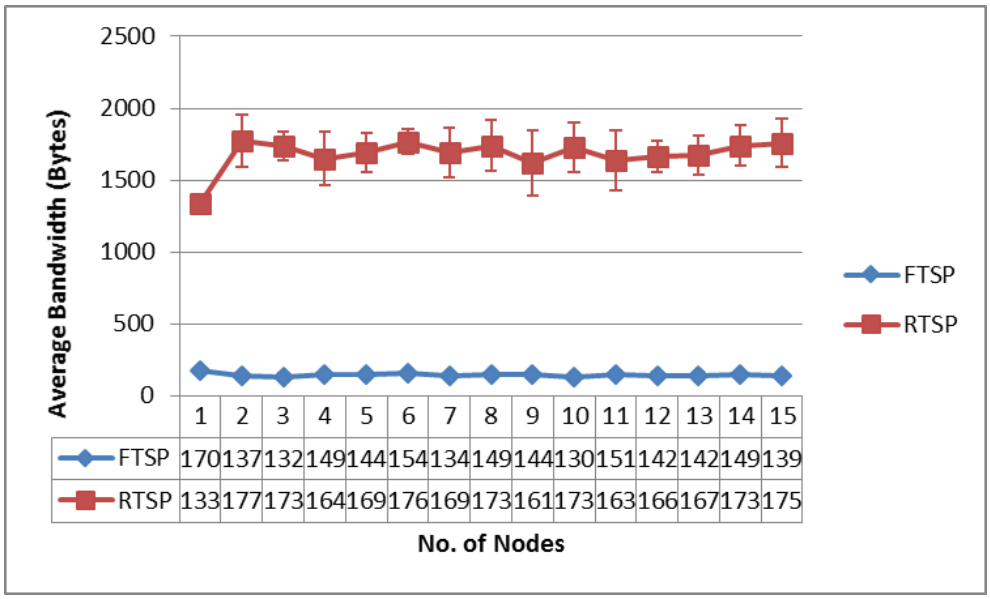


Figure 5.26: Mesh Topology - Average Number of Messages for each experiment with 95% confidence interval

As it can be seen in Figure 5.26 that both protocols have linear behavior over the course of five experiments. On average, FTSP uses 12 and RTSP uses 36 messages/node for a single synchronization cycle. In essence, both protocols are not very feasible for mesh topology.

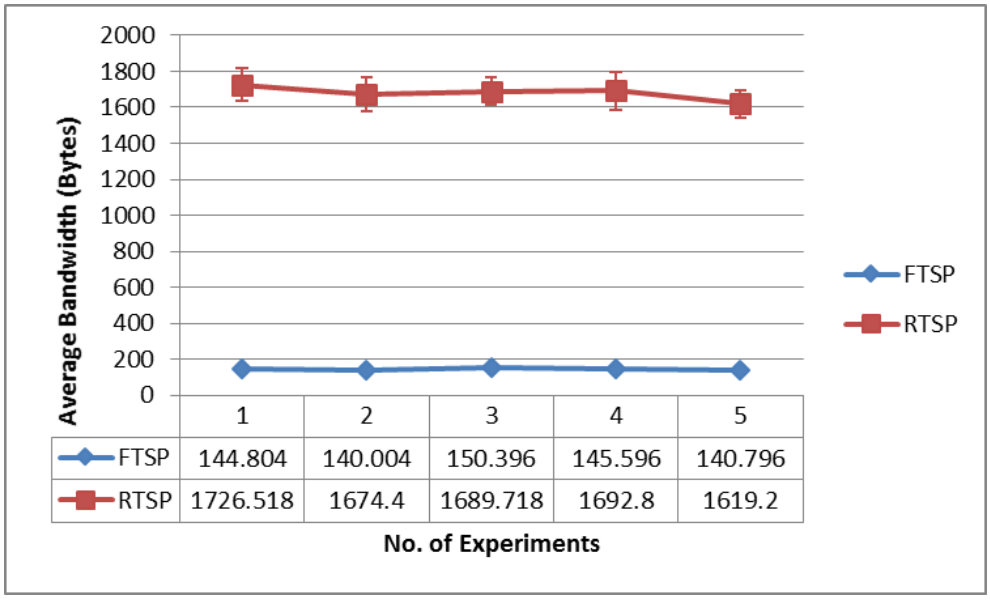
The bandwidth situation is worse. For RTSP bandwidth consumption is always in few hundred bytes, less than 1 Kilobyte. But in this case, it went all the way up to 1.7KB, which is very significant for a single synchronization cycle. FTSP also performed worse here as compared to its performance in rest of topologies, but still much better than RTSP as it can be seen in figures 5.26 and 5.27.

Convergence time metric is not shown for mesh topology, because it is almost ignorable. Almost all of the nodes have convergence time equal to zero.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FTSP	170	137	132	149	144	154	134	149	144	130	151	142	142	149	139
RTSP	133	177	173	164	169	176	169	173	161	173	163	166	167	173	175

Figure 5.27: Mesh Topology - Average Bandwidth per node for five experiments with 95% confidence interval



	1	2	3	4	5
FTSP	144.804	140.004	150.396	145.596	140.796
RTSP	1726.518	1674.4	1689.718	1692.8	1619.2

Figure 5.28: Mesh Topology - Average Bandwidth for each experiment with 95% confidence interval

5.4 Tree Topology

Figure 5.29 shows physical tree topology configurations for both protocols. Node 1 is the root node. Rest of the configuration for both protocols is same as described

in previous sections.

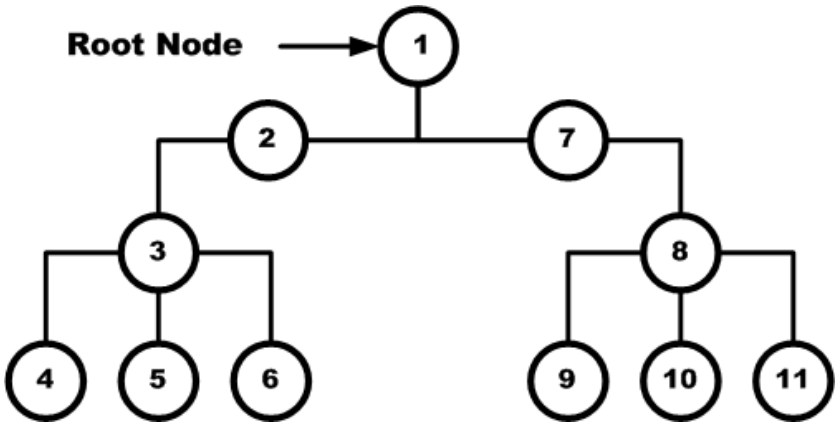


Figure 5.29: Tree Topology

Figure 5.30 shows the result of FTSP tree topology. As we can see, the results are not linear. It has great degree of fluctuations due to interference from other nodes. For example, node 3 and 8 should only receive messages from their children and parent, but they are receiving message from other nodes as well. This makes it very unpredictable to accurately determine which node is receiving from which node. Message count is high as well. One thing is for certain, each node is sending/receiving messages from legitimate nodes e.g. node 3 is receiving/sending messages to node 2, 4, 5, and 6, but this node is also receiving from other nodes as well, mostly echo messages (rebroadcasts). One very good reason is that topology is physical (no hard coded topology in the software unlike RTSP) and another reason as explained earlier, state of the individual 9V battery, if some nodes have higher power left they would have much larger radius of transmission that makes it interfere with other nodes.

Figure 5.31 shows experimentation of RTSP using tree topology. RTSP per-

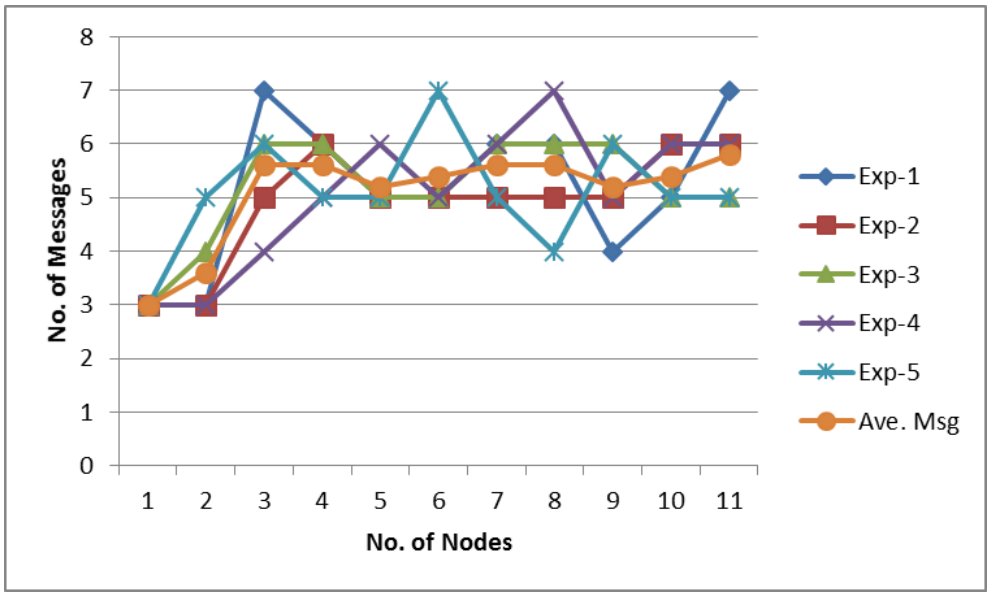


Figure 5.30: FTSP Tree Topology - Number of Messages per Node for each experiment

forms better than FTSP for tree topology because inherently this protocol is made for only tree/cluster topology. It is almost constant over five experiments there are not many fluctuations. High message count can only be seen at node 3 and node 8, because they are connected with 4 nodes, so their message count would be higher than rest of the nodes. The main reason for this behaviour is topology is physical as well as hard coded into the software as well. In this node 1, 2, 3, 7, and 8 are clusterheads. So communication is vertical most of the time, but some time it can also go horizontal between two clusterheads, but it never jumps clustergheads. For example node 8 cannot directly communicate with node 1, it can only communicate with its children, node 7 and node 3 (if necessary). Another factor could be batteries were little weaker as compared to FTSP tree topology experiment, because this was the last experiment conducted among all 4. Weaker

batteries means smaller radius of transmission hence less interference.

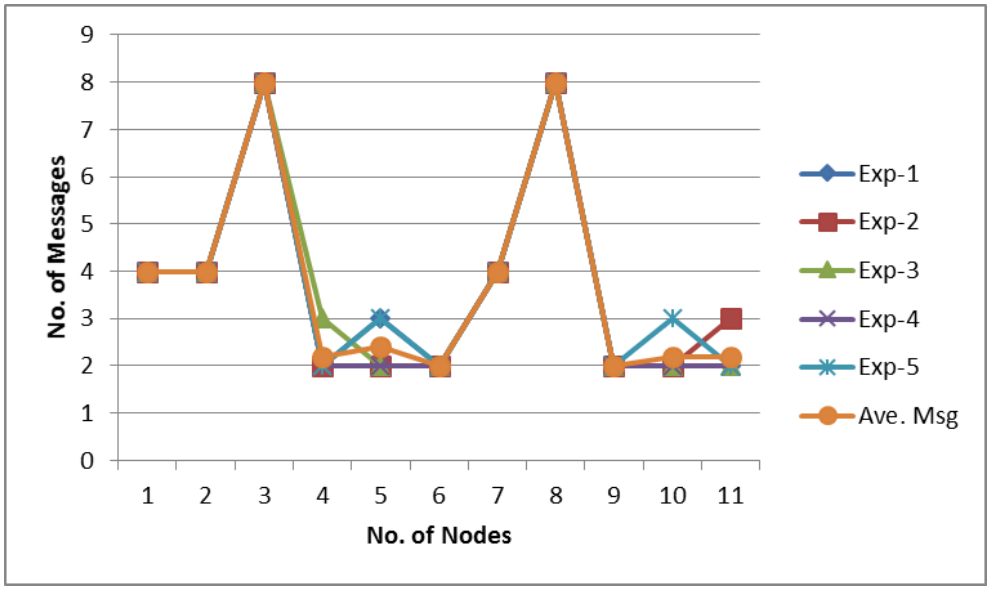
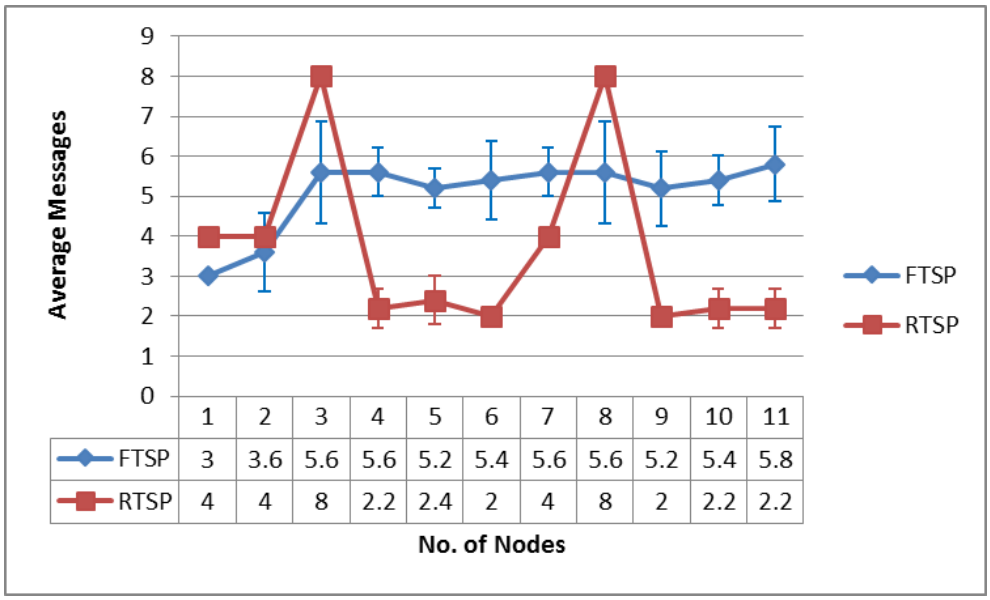


Figure 5.31: RTSP Tree Topology - Number of Messages per Node for each experiment



	1	2	3	4	5	6	7	8	9	10	11
FTSP	3	3.6	5.6	5.6	5.2	5.4	5.6	5.6	5.2	5.4	5.8
RTSP	4	4	8	2.2	2.4	2	4	8	2	2.2	2.2

Figure 5.32: Tree Topology - Average Number of Messages per node for five experiments with 95% confidence interval

Figure 5.32 shows comparison between both protocols for tree topology. Aver-

age messages/node is higher for RTSP for nodes 1, 2, 3 and 8, but for the rest of the nodes it's much less than FTSP. In RTSP high message count at some nodes is due to more connected nodes with them. Even the total message count for FTSP is high regardless of few high edges in RTSP i.e. 56 and 41, respectively. In tree topology, RTSP performs much better than FTSP for a given metric of message count.

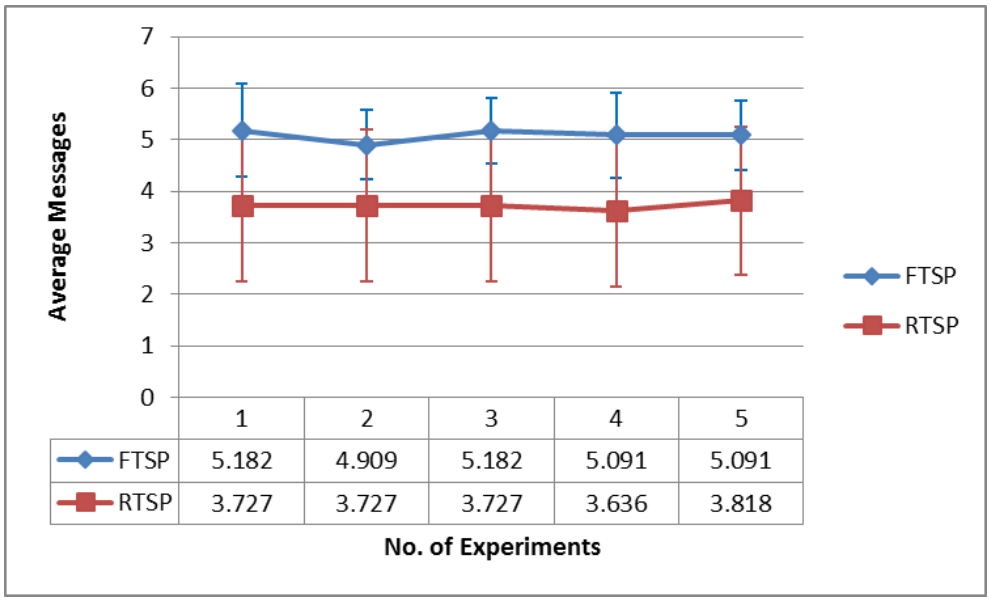
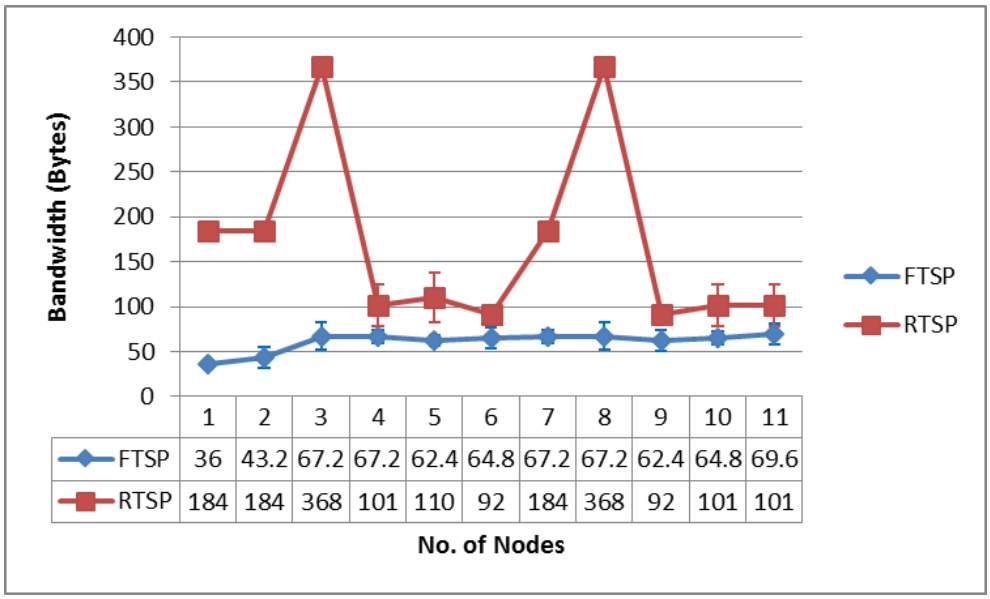


Figure 5.33: Tree Topology - Average Number of Messages for each experiment with 95% confidence interval

If we look at the performance of both protocols for five experiments, it is almost linear for both of them. Although FTSP has higher average message count throughout the experiments, but it is linear, which means it is not the best choice for the this particular topology as shown in Figure 5.33.

Only in this topology bandwidth parameter really makes a difference. As we have seen, RTSP utilizes less number of messages for synchronization in case of

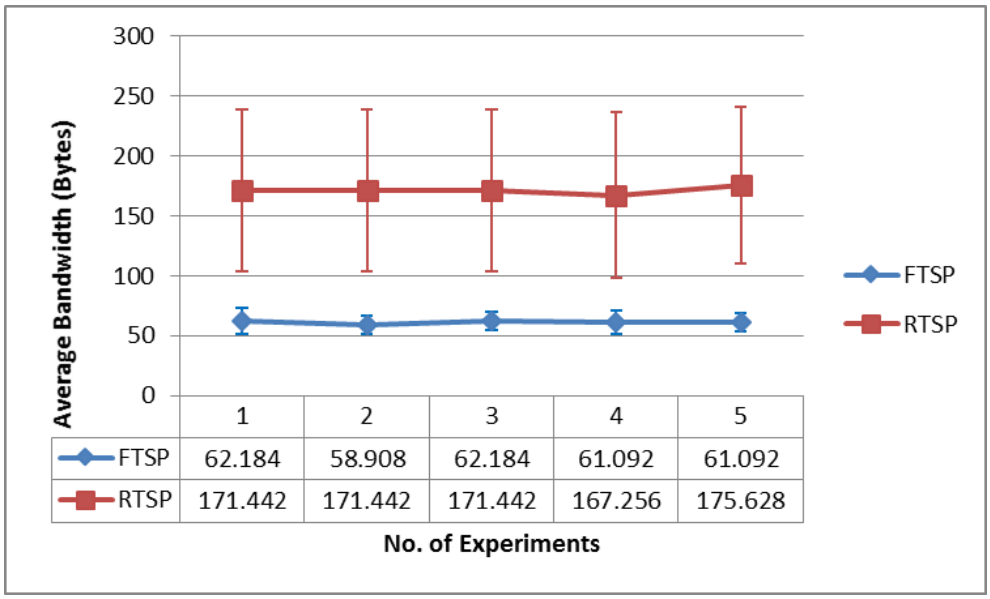


	1	2	3	4	5	6	7	8	9	10	11
FTSP	36	43.2	67.2	67.2	62.4	64.8	67.2	67.2	62.4	64.8	69.6
RTSP	184	184	368	101	110	92	184	368	92	101	101

Figure 5.34: Tree Topology - Average Bandwidth per node for five experiments with 95% confidence interval

tree topology. So, if we only rely on message count metric than RTSP is a better choice. As it can be seen in figures 5.34 and 5.35. Bandwidth consumption is very high for RTSP due to larger packet size and those spikes at node 3 and 8 are due to connectivity with large number of nodes. Difference is much more visible in Figure 5.35. Even FTSP performs slightly poorer, it is still a better protocol in terms of bandwidth consumption. RTSP is an efficient protocol in terms of synchronization for tree topology, but it has inherent disadvantage, when it comes to bandwidth.

Figure 5.36 shows convergence time for FTSP. Most of the time it is random. There is no real explanation for why there is a delay of one second for clusterhead node 3 and there is not for node 4, 5 or 6 in experiment 4. Possible reasons are explained in earlier sections.



	1	2	3	4	5
FTSP	62.184	58.908	62.184	61.092	61.092
RTSP	171.442	171.442	171.442	167.256	175.628

Figure 5.35: Tree Topology - Average Bandwidth for each experiment with 95% confidence interval

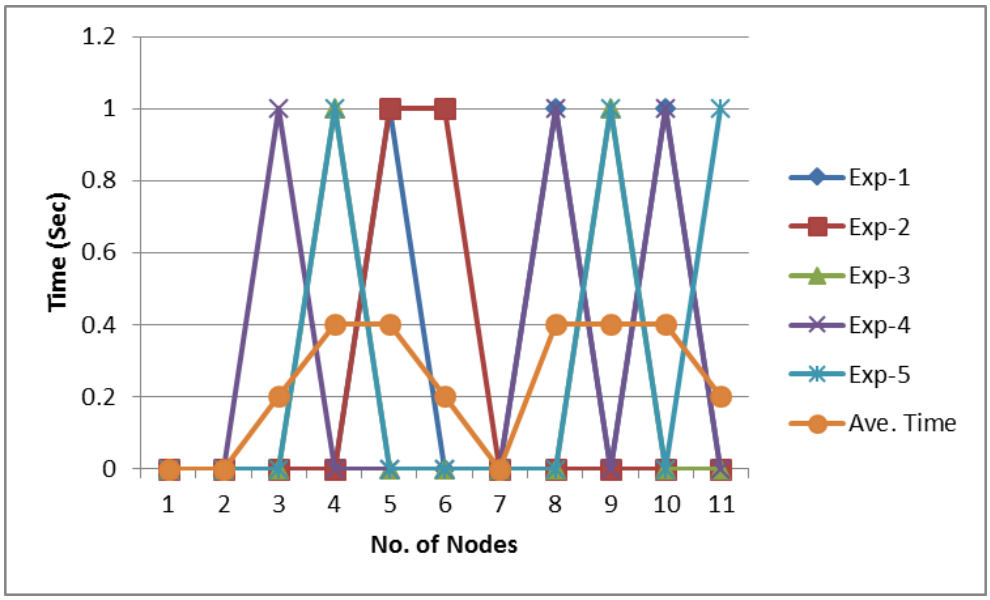


Figure 5.36: FTSP Tree Topology - Convergence Time per Node for each experiment

For RTSP, it is very low. Convergence time was zero for experiment number 3, 4 and 5. There is no delay at clusterheads or at the root node. If there is a

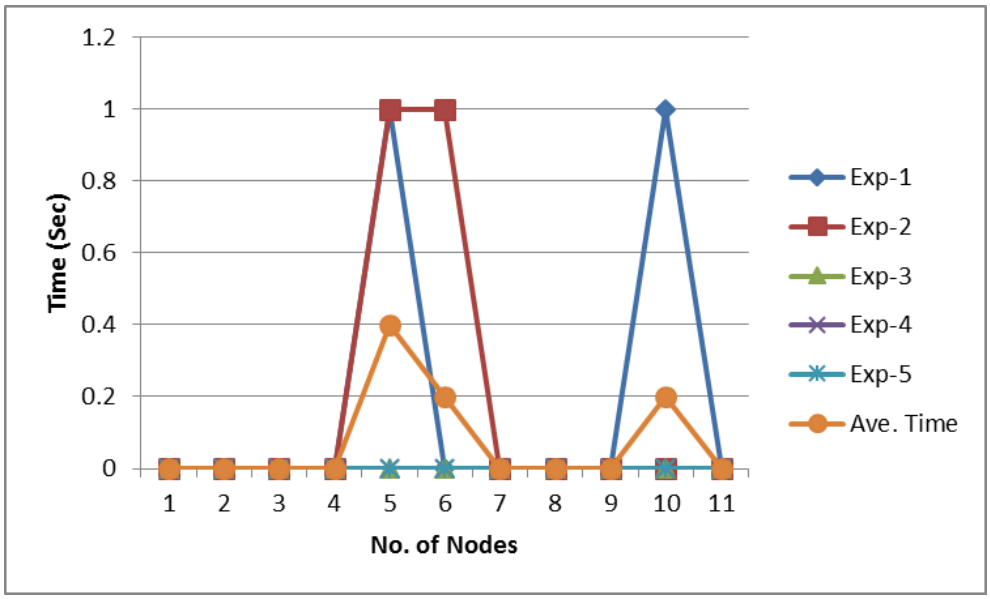
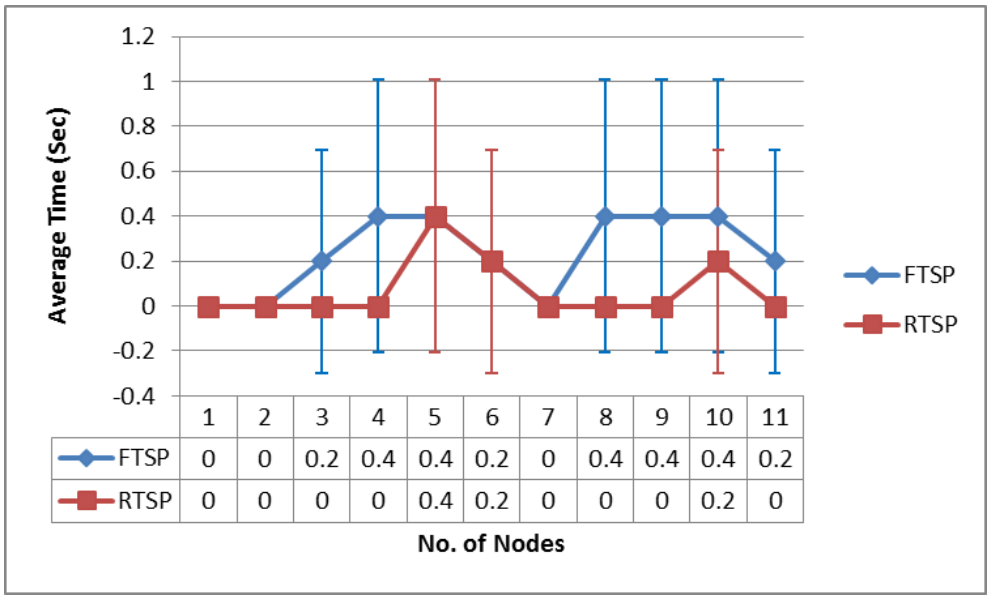


Figure 5.37: RTSP Tree Topology - Convergence Time per Node for each experiment

delay at any clusterhead at any point than those delay would have been seen in the child nodes as well, because RTSP uses peer to peer communication in this case. On the other hand, FTSP uses broadcast, so there is no way of knowing with absolute certainty that which node synchronized from which node. We can just assume and make sure while setting up the physical topology that it would work as we have configured and hoped.

In case of the average convergence time of each node over five experiments, RTSP perform much better than FTSP as shown in Figure 5.38. Average convergence time for five experiments is shown in Figure 5.39. RTSP starts with 182 milliseconds, but after second experiment it goes to zero for rest of the experiment. So, RTSP performs better in the convergence time as well. Recursive Time Synchronization protocol was designed for cluster based topology i.e. tree



	1	2	3	4	5	6	7	8	9	10	11
FTSP	0	0	0.2	0.4	0.4	0.2	0	0.4	0.4	0.4	0.2
RTSP	0	0	0	0	0.4	0.2	0	0	0	0.2	0

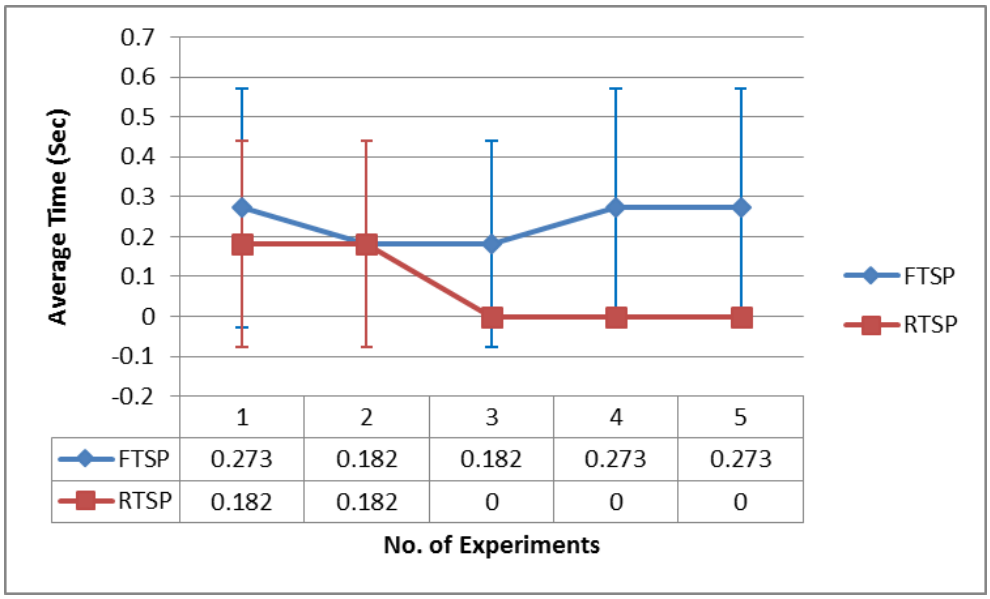
Figure 5.38: Tree Topology - Average Convergence Time per node for five experiments with 95% confidence interval

topology. So, it does perform better in this case. But FTSP was designed for all topologies, it perform better in most of them, but slightly poorer in tree topology.

Figure 5.40, 5.41, and 5.42 shows summary of all experimentation for Number of Messages, Bandwidth and Convergence Time metric, respectively.

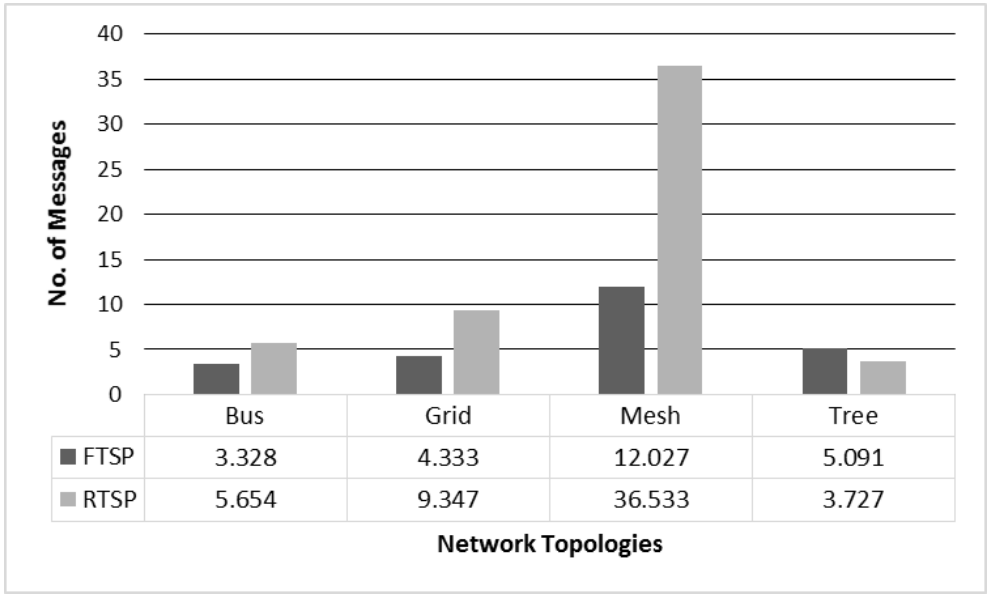
Performance evaluation metrics are effected by the simplifications (explained in chapter 4). We will discuss the effects of these simplifications for Number of Messages and Convergence Time metric. The Bandwidth metric is identical to Number of Messages metric, therefore we will discuss only one of them.

For Number of Messages metric, simplifications have not degraded the results rather they have improved them. The proposed FTSP [57] requires more number of messages for single synchronization cycle, because there is a root election process



	1	2	3	4	5
FTSP	0.273	0.182	0.182	0.273	0.273
RTSP	0.182	0.182	0	0	0

Figure 5.39: Tree Topology - Average Convergence Time of each experiment with 95% confidence interval



	Bus	Grid	Mesh	Tree
FTSP	3.328	4.333	12.027	5.091
RTSP	5.654	9.347	36.533	3.727

Figure 5.40: Comparison for all four topologies for Number of Messages metric

as well as it requires extra messages to calculate local clock drift. On the other hand, there is also a significant reduction of number of messages for RTSP as well.

It requires significantly large number of messages for root election, root discovery, energy compensation, and also we have eliminated all ERN type messages that are major contributor of messages [1, 60].

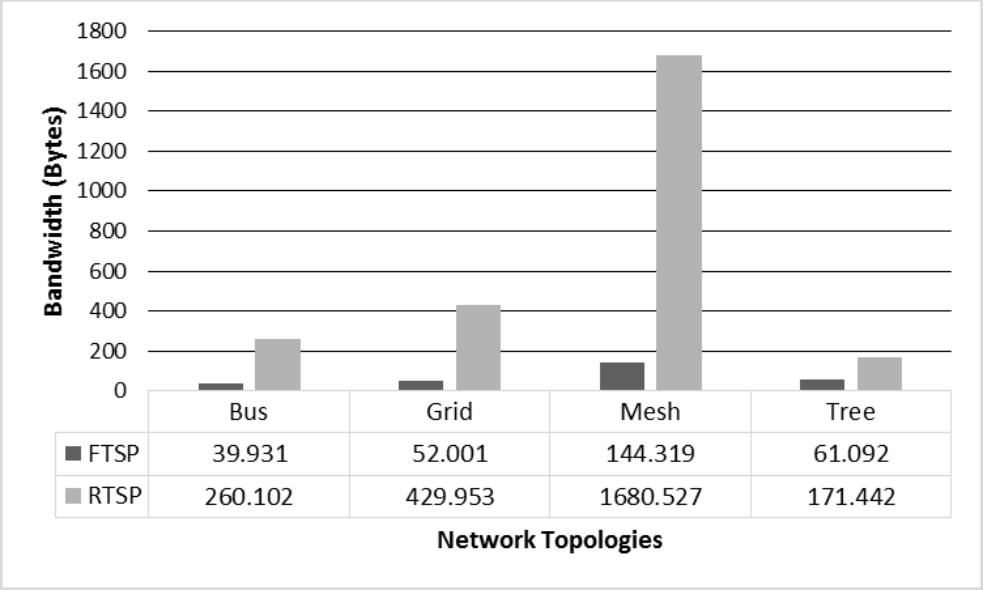


Figure 5.41: Comparison for all four topologies for Bandwidth metric

Second metric that is effected by these simplifications is Convergence Time. Major sources of delay in message transmission are send time, access time, transmission time, propagation time, reception time, and receive time [57]. Definition of these terms are as following:

Send Time- time required to assemble the message and issue a send request to transmitter's MAC layer. It could be in hundreds of milliseconds.

Access Time- time required to access the channel for transmitting the message.

Transmission Time- time required by a sender to transmit a message.

Propagation Time- the time required by the message to travel from sender to

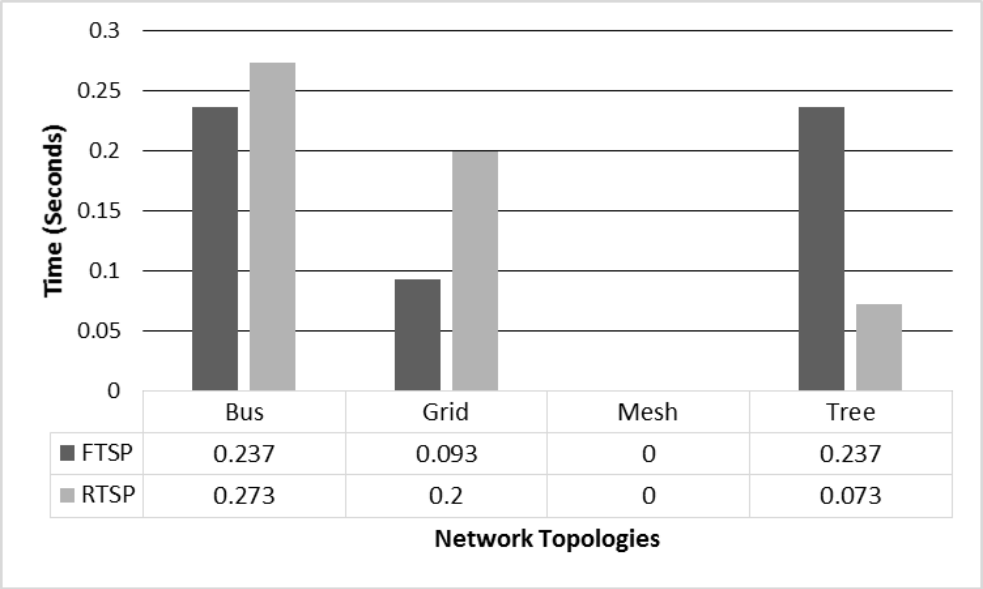


Figure 5.42: Comparison for all four topologies for Convergence Time metric

receiver.

Reception Time- time required by the receiver to receive a message. Similar to transmission time.

Receive Time- time needed to process the incoming message and to notify the receiver application.

These are some of the sources that accumulates to significant delays in the message transmission. FTSP and RTSP try to eliminate most of these delays by using MAC layer time stamping and some other methodologies [57]. But in our implementation, all of these delays are included in the convergence time metric, because hardware does not provide direct access to the network or MAC layer and protocols are implemented at application layer. Therefore, we cannot use any of the proposed methodologies to eliminate aforementioned sources of delay in the

message transmission. Our convergence time metric results does not translate to the results shown by the authors in their respective research [1, 57, 60].

CHAPTER 6

CONCLUSION AND FUTURE WORK

Wireless sensor networks consist of small low power devices that are deployed in an ad-hoc manner to perform special tasks e.g. collecting temperature information, sand storm detection, humidity etc. Wireless sensor networks have some pre-defined objective i.e. low energy consumption, bandwidth efficiency, time synchronization, and data collection. To collect sensible information from the network; time synchronization is required among nodes. Time synchronization is a crucial problem, but it is more challenging to solve in wireless sensor networks.

In this research, a performance evaluation of two state-of-the-art time synchronization protocols for wireless sensor networks have been presented, namely: Flooding Time Synchronization Protocol (FTSP) and Recursive Time Synchronization Protocol (RTSP). Both protocols are tested in a real-time environment using Arduino and XBee as hardware platform. The metrics used for perfor-

mance evaluation are number of messages per synchronization cycle, bandwidth and convergence time. Both, FTSP and RTSP have some pros and cons. FTSP is a broadcast protocol and it has lower accuracy at edge nodes, whereas RTSP is a peer-to-peer protocol with higher accuracy at edge nodes.

FTSP performs much better in bus, grid and mesh topologies for number of messages metric. But RTSP has higher performance in tree topology, because this protocol was designed for cluster based topology i.e. tree topology. On the contrary, RTSP has degraded performance for bandwidth for all four topologies, especially in mesh topology. RTSP has very high bandwidth requirements, almost four times that of FTSP. The main reason for high bandwidth requirement is the large packet size of RTSP. The last metric used for performance evaluation is convergence time. The results for convergence time are inconclusive. Both protocols show minor fluctuations in the convergence time throughout the experimentation, but the resolution for time measurement is in seconds, so on average none of the protocols go above one second. The main reasons for convergence time inconclusiveness are speed of the communication and resolution of time measurement. Speed of the communication is fast enough that at a short distance convergence time does not make much difference. If we measure the time in milliseconds or microseconds than we can see real difference even at a smaller distance. According to current measurements, both protocols converge quickly. Although, there is a tradeoff in using smaller resolutions, if smaller time resolutions are used; it would require more bandwidth to transmit information. On the other hand, vari-

ation in the convergence time are due to several other reasons e.g. hardware clock variations, clock reset delays, wireless medium collisions etc.

Another very important parameter for performance analysis is energy consumption. Energy consumption has not been measured in the experimentation due to hardware limitations, but we can interpret from the above results, because it is directly related to the bandwidth. The simplest interpretation would be; the higher the bandwidth, the higher the energy consumption. RTSP has higher bandwidth consumption throughout experimentation, so it is safe to assume that it would require significant amount of energy to perform time synchronization. Another important factor in energy consumption is that a node would require to stay awake for a longer duration of time to transmit and receive information in case of RTSP. We can also conclude that RTSP is an energy and bandwidth inefficient protocol as compared to FTSP in a small scale network.

RTSP is a good theoretical protocol, but not very practical. Although, it solves the problem of accuracy at edge nodes, but creates other problems like energy and bandwidth inefficiency, which are integral part of wireless sensor networks. On the other hand, FTSP is a simple and efficient protocol with slightly less accuracy at the edge of a large network. In the end, it depends on the requirement of an application, if it require higher accuracy and have abundance of bandwidth & energy (which is not the case in wireless sensor networks) than RTSP is a better choice or if an application can tolerate slightly less accuracy than FTSP is a better choice.

In future work, both protocols would be compared under more flexible hardware. Bandwidth efficiency can be achieved by developing a custom sleep wake mechanism. Accuracy and energy efficiency are two important parameters that are not analyzed in this thesis due to hardware restrictions, but they can be measured using open source tools and hardware.

REFERENCES

[1] M. Akhlaq and T. R.Sheltami, "Rtsp: An accurate and energy-efficient protocol for clock synchronization in wsns," *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 3, pp. 578–589, March 2012.

[2] J. Hill, M. Horton, R. Kling, and L. Krishnamurthy, "The Platforms Enabling Wireless Sensor Networks," *Commun. ACM*, vol. 47, no. 6, pp. 41–46, 2004.

[3] "The intel mote intel corporation." [Online]. Available: http://www.intel.com/research/exploratory/motes.htmi

[4] "Smartdust. autonomous sensing and communication in a cubic millimeter." [Online]. Available: http://robotics.eecs.berkeley.edu/pister/SmartDust

[5] "Tinyos. an operating system for networked sensors." [Online]. Available: http://www.tinyos.net

[6] P. Johnson and D. Andrews, "Remote Continuous Physiological Monitoring in the Home," *Telemedicine Telecare*, vol. 2, no. 2, pp. 107–113, 1996.

[7] "Cens: Seismic monitoring and structural response." [Online]. Available: http://www.cens.ucla.edu/Research/Applic-ations/seismic monitor.htm

[8] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin., "Habitat Monitoring with Sensor Networks," *Communications of the ACM*, vol. 46, no. 6, pp. 34–40, june 2004.

[9] "Distributed surveillance sensor network. onr spawar systems center, san diego." [Online]. Available: http://www.spawar.navy.mil /robots/undersea/dssn/dssn.html

[10] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao, "Habitat monitoring: application driver for wireless communications technology," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 2 supplement, pp. 20–41, Apr. 2001.

[11] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, ser. WSNA '02, 2002, pp. 88–97.

[12] J. Kim, Y. Park, and T. C. Harmon, "Real-time model parameter estimation for analyzing transport in porous media," Center for Embedded Networked Sensing,University of California, Los Angeles, USA, Tech. Rep., May 2003.

[13] A. Berlin, J. Chase, M. Yim, B. Maclean, M. Oliver, and S. Jacobsen, "MEMS-Based Control of Structural Dynamic Instability," *Journal of Intelligent Material Systems and Structures*, vol. 9, no. 7, pp. 574–586, 1998.

[14] "Chart on the web. maryland department of transportation." [Online]. Available: http://www.chart.state.md.us/

[15] Q. Li, M. De Rosa, and D. Rus, "Distributed algorithms for guiding navigation across a sensor network," in *Proceedings of the 9th annual international conference on Mobile computing and networking*, ser. MobiCom '03, 2003, pp. 313–325.

[16] R. Want, A. Hopper, V. Falcão, and J. Gibbons, "The active badge location system," *ACM Trans. Inf. Syst.*, vol. 10, no. 1, pp. 91–102, Jan. 1992.

[17] J. Werb and C. Lanzl, "Designing a positioning system for finding things and people indoors," *Spectrum, IEEE*, vol. 35, no. 9, pp. 71–78, 1998.

[18] I. Akyildiz, O. Akan, C. Chen, J. Fang, and W. Su, "Interplanetary Internet: State-of-the-art and Research Challenges," *Computer Networks*, vol. 43, no. 2, pp. 75–112, 2003.

[19] S. Burleigh, V. Cerf, R. Durst, K. Fall, A. Hooke, K. Scott, and H. Weiss, "The Interplanetary Internet: A Communications Infrastructure for Mars Exploration," in *53rd International Astronautical Congress, The World Space Congress*, Oct. 2002.

[20] L. Lemmerman, K. Delin, F. Hadaegh, M. Lou, K. Bhasin, J. Bristow, R. Connerton, and M. Pasciuto, "Earth science vision: platform technology challenges," in *Geoscience and Remote Sensing Symposium, 2001. IGARSS '01. IEEE 2001 International*, vol. 1, 2001, pp. 439–443.

[21] "The sensor web project. nasa jet propulsion laboratory." [Online]. Available: http://sensorwebs.jpl.nasa.gov

[22] "Nasa deep space network." [Online]. Available: http://deepspace.jpl.nasa.gov/dsn

[23] "Mars exploration program." [Online]. Available: http://mars.jpl.nasa.gov/

[24] "Cens: Center for embedded networked sensing." [Online]. Available: http://www.cens.ucla.edu/index.html

[25] D. E. Culler and W. Hong, "Introduction," *Commun. ACM*, vol. 47, no. 6, pp. 30–33, Jun. 2004.

[26] D. Culler, D. Estrin, and M. Srivastava, "Guest editors' introduction: Overview of sensor networks," *Computer*, vol. 37, no. 8, pp. 41–49, 2004.

[27] D. Estrin, D. Culler, K. Pister, and G. Sukhatme, "Connecting the physical world with pervasive networks," *Pervasive Computing, IEEE*, vol. 1, no. 1, pp. 59–69, 2002.

[28] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: scalable coordination in sensor networks," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, ser. MobiCom '99, 1999, pp. 263–270.

[29] L. Girod, V. Bychkovskiy, J. Elson, and D. Estrin, "Locating tiny sensors in time and space: A case study." in *ICCD*, 2002, pp. 214–219.

[30] G. Sibley, M. Rahimi, and G. Sukhatme, "Robomote: a tiny mobile robot platform for large-scale ad-hoc sensor networks," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 2, 2002, pp. 1143–1148.

[31] "Darpa advanced technology office (ato)." [Online]. Available: http://www.darpa.mil/ato/programs/SHM/

[32] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors," *Commun. ACM*, vol. 43, no. 5, pp. 51–58, May 2000.

[33] J. Elson, S. Bien, N. Busek, V. Bychkovskiy, A. Cerpa, D. Ganesan, L. Girod, B. Greenstein, T. Schoellhammer, T. Stathopoulos, and D. Estrin, "Emstar: An environment for developing wireless embedded systems software," Center for Embedded Networked Sensing,University of California, Los Angeles, USA, Tech. Rep., March 2003.

[34] H. Wang, L. Yip, D. Maniezzo, J. Chen, R. E. Hudson, J. Elson, and K. Yao, "A wireless time-synchronized cots sensor platform: Applications to beamforming," in *In Proceedings of IEEE CAS Workshop on Wireless Communications and Networking*, 2002, p. 02.

[35] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, ser. MobiCom '00, 2000, pp. 56–67.

[36] G. Asada, M. Dong, T. S. Lin, F. Newberg, G. Pottie, W. Kaiser, and H. Marcy, "Wireless integrated network sensors: Low power systems on a chip," in *Solid-State Circuits Conference, 1998. ESSCIRC '98. Proceedings of the 24th European*, 1998, pp. 9–16.

[37] P. Bonnet, J. Gehrke, and P. Seshadri, "Querying the physical world," *Personal Communications, IEEE*, vol. 7, no. 5, pp. 10–15, 2000.

[38] S. Madden and M. Franklin, "Fjording the stream: an architecture for queries over streaming sensor data," in *Data Engineering, 2002. Proceedings. 18th International Conference on*, 2002, pp. 555–566.

[39] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "Spins: Security suite for sensor networks," in *Seventh Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '01, 2001, pp. 56–67.

[40] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *Proceedings of the 9th ACM conference on Computer and communications security*, ser. CCS '02, 2002, pp. 41–47.

[41] Y. Law, S. Dulman, S. Etalle, and P. Havinga, "Assessing security-critical energy-efficient sensor networks," in *FIP WG 11.2 Small Systems Security Conference,*. Kluwer Academic Publishers, 2003, pp. 459–463.

[42] L. Girod and D. Estrin, "Robust range estimation using acoustic and multimodal sensing," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 3, 2001, pp. 1312–1320 vol.3.

[43] W. Merrill, L. Girod, J. Elson, K. Sohrabi, F. Newberg, and W. Kaiser, "Autonomous position location in distributed, embedded, wireless systems," in *In Proceedings of IEEE CAS Workshop on Wireless Communications and Networking*, 2002.

[44] D. L. Mills, "A brief history of ntp time: Memoirs of an internet timekeeper," *ACM SIGCOMM COMPUT. COMMUN. REV*, vol. 3, pp. 9–21, 2003.

[45] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, pp. 146–158, 1989.

[46] R. Gusella and S. Zatti, "The accuracy of the clock synchronization achieved by tempo in berkeley unix 4.3bsd," *Software Engineering, IEEE Transactions on*, vol. 15, no. 7, pp. 847–853, 1989.

[47] D. Mills, "Internet time synchronization: the network time protocol," *Communications, IEEE Transactions on*, vol. 39, no. 10, pp. 1482–1493, 1991.

[48] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *J. ACM*, vol. 34, no. 3, pp. 626–645, Jul. 1987.

[49] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, Dec. 2002.

[50] "Wireless lan medium access control (mac) and physical layer (phy) specications," *ANSI/IEEE Std 802.11*, 1997.

[51] M. Mock, R. Frings, E. Nett, and S. Trikaliotis, "Continuous clock synchronization in wireless real-time applications," in *Reliable Distributed Systems, 2000. SRDS-2000. Proceedings The 19th IEEE Symposium on*, 2000, pp. 125–132.

[52] S. Ganeriwal, R. Kumar, S. Adlakha, and M. Srivastava, "Network-wide time synchronization in sensor networks," Networked and Embedded Systems Lab, Elec. Eng. Dept., UCLA, Los Angeles, USA, Tech. Rep., 2003.

[53] 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, USA, 2003, pp. 138–149.

[54] S. Ping, "Delay Measurement Time Synchronization for Wireless Sensor Networks," *Intel Research, IRB-TR-03-013*, Jun. 2003.

[55] S. PalChaudhuri, A. Saha, and D. B. Johnson, "Probabilistic clock synchronization service in sensor networks," Department of Computer Science, Rice University, Tech. Rep., 2003.

[56] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, "A taxonomy of wireless micro-sensor network models," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, no. 2, pp. 28–36, Apr. 2002.

[57] M. Marti, B. Kusy, G. Simon, and kos Ldeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd International Conference on*

Embedded Networked Sensor Systems (SenSys'04), Boulder, Colorado, USA, 2004, pp. 39–49.

[58] D. Cox, E. Jovanov, and A. Milenkovic, "Time synchronization for zigbee networks," in *Proceedings of the Thirty-Seventh Southeastern Symposium on System Theory*, Tuskegee, AL, USA, March 2005, pp. 135–138.

[59] M. Aoun, A. Schoofs, and P. van der Stok, "Efficient time synchronization for wireless sensor networks in an industrial setting," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys'08)*, Zurich, Switzerland, 2008, pp. 419–420.

[60] M. Akhlaq and T. R.Sheltami, "The recursive time synchronization protocol for wireless sensor networks," in *IEEE Sensors Applications Symposium (SAS)*, Brescia, Italy, February 2012, pp. 67–72.

[61] "Arduino board mega." [Online]. Available: http://arduino.cc /en/Main/arduinoBoardMega2560

[62] "Zigbee toplogies." [Online]. Available: http://www.icpdas.com/products/ GSM_GPRS/zigbee/zigbee_introduction.htm

[63] "Xbee pro module." [Online]. Available: http://www.digi.com/products /wireless-wired-embedded-solutions/zigbee-rf-modules/zigbee-mesh-module /xbee-digimesh-900

[64] "Xbee shield." [Online]. Available: https://www.sparkfun.com /products/10854

[65] "Xbee explorer." [Online]. Available: https://www.sparkfun.com /products/8687

[66] "X-ctu." [Online]. Available: http://www.digi.com /support/kbase/kbaseresultdetl?id=2125

[67] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 90–.

[68] M. Akhlaq, "The recursive time synchronization protocol," Ph.D. dissertation, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, April 2012.

Vitae

- Name: Danish Sattar
- Nationality: Pakistani
- Date of Birth: July 01, 1987
- Email: *danishsattar@gmail.com*
- Permanent Address: CB-320/9 Lalazar Wah Cantt, Pakistan.

- Education: MS in Computer Networks (December, 2013) from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.

  BS in Telecommunication & Networking (August, 2009) from COMSATS Institute of Information Technology, Wah Campus, Pakistan.

- Publication: Danish Sattar, Tarek R. Sheltami, Ashraf S. Hasan Mahmoud and Elhadi M. Shakshuki, *"A Comparative Analysis of Flooding Time Synchronization Protocol and Recursive Time Synchronization Protocol",* published in proceedings of the 11th International Conference on Advances in Mobile Computing and Multimedia (MoMM13), Vienna, Austria, December 2013.