2008

A Wrapper-Based Approach to Sustained Time Synchronization in Wireless Sensor Networks

Dheeraj Reddy Bheemidi
Cleveland State University

A WRAPPER-BASED APPROACH TO SUSTAINED TIME SYNCHRONIZATION IN WIRELESS SENSOR NETWORKS

DHEERAJ REDDY BHEEMIDI

Bachelor of Technology (B.Tech)

Electronics and Communication Engineering(E.C.E)

Jawaharlal Nehru Technical University

May 2006

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

at the

CLEVELAND STATE UNIVERSITY

November 2008

This thesis has been approved for the

Department of **ELECTRICAL AND COMPUTER ENGINEERING**

and the College of Graduate Studies by

Thesis Committee Chairperson, Dr. Nigamanth Sridhar

Department/Date

Dr. Chansu Yu

Department/Date

Dr. Wenbing Zhao

Department/Date

To my parents Mrs. Vidyavathi Bheemidi and Mr. Prabhakar Reddy Bheemidi and my sister Ms.Preethi Bheemidi .....

ACKNOWLEDGMENTS

I would like to thank all those who gave me the opportunity to complete this thesis. Firstly I would like to thank my advisor, Professor Dr.Nigamanth Sridhar, for all the support and encouragement. I would also like to thank the Department of Electrical and Computer Engineering, of which I have had the pleasure being a student the past two years. A special thanks to Professor Dr.Chansu Yu for his support during my initial stages of research.

I wish to thank my graduate friends Madhu Mudigonda, Hamza Zia Ahmed, Satya Guttula, Sunil Gavini, Trisul Kanipakam, Manohar Bathula for their encouragement during the course of my research.

This research has been supported by the National Science Foundation under grant CNS-0746632.

A WRAPPER-BASED APPROACH TO SUSTAINED TIME SYNCHRONIZATION IN WIRELESS SENSOR NETWORKS

DHEERAJ REDDY BHEEMIDI

ABSTRACT

Time synchronization is an important service for wireless sensor network applications. Nodes in the network stay synchronized by exchanging periodic messages that carry local timestamps. Several algorithms have been proposed in the literature that are suited to different kinds of application scenarios. A common problem across these time synchronization algorithms is that the energy cost of message exchange is high. In fact, the cost of radio communication far outstrips the cost of performing local operations on the processor. If the message exchanges were stopped, nodes will fall out of sync, and may no longer be able to meet application requirements.

This thesis presents a wrapper-based approach to sustained time synchronization for wireless sensor networks. As such, this solution Booster for Time Synchronization Protocol (**BTSP**) will act as a wrapper around a given time synchronization protocol, and will apply local corrector operations to extend the time duration between two message exchanges between nodes. The wrapper performs at least as good as the original protocol provided, reduces the number of message exchanges on average, and consequently the energy consumed, significantly. **BTSP** has been implemented for TinyOS and evaluated on XSM motes in conjunction with TPSN, a popular time synchronization protocol for sensor networks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Applications involving low power devices have been a major revolution in the last decade. The vision of sensor networks is to be able to deploy a huge number of small sensor/actuator nodes that are capable of observing the physical environment around them, and use that sensed information in some meaningful way. Such large numbers also imply that each node individually must be very inexpensive to produce, so inexpensive that losing a fraction of the deployed nodes is not a big deal. In such a setting, the hardware components of these small, inexpensive nodes are bound to be imperfect. Such imperfections creep into the behavioral function of the application, and it is the application software's responsibility to protect itself from consequences of such imperfections.

When dealing with networks of large number of sensor nodes , it becomes necessary to create some way in which each node in the network can share a common view of time. On their own, any network composed of resource constrained devices tend to behave in an *asynchronous* fashion: there are no guarantees of timing when actions in the network span node boundaries. However, in most sensor system applications,

there is a need for nodes in the network to be synchronized with the remaining nodes in the network.

1.1 The Problem

Several algorithms have been proposed for achieving both pair-wise as well as multi-hop time synchronization in wireless sensor networks [2] [3] [4] [5] [6] [8] [12] [13] [14]. These algorithms even achieve synchronization within microsecond accuracy. The problem, however, is that these algorithms are not equipped to maintain the same level of accuracy over extended periods of time. As a result, applications that require synchronized clocks for the entire lifetime of the deployment resort to invoking the synchronization algorithm periodically i.e, frequently, nodes have to be synchronized with each other in order to collaborate effectively. In message-passing based protocols, such continuous updates require the exchange of several messages among nodes. This, in turn, is expensive in terms of energy required for communication.

1.2 The Thesis

This Thesis propose's a software component, called Booster for Time Synchronization Protocol (**BTSP**) that continually monitors the drift between two nodes, and performs internal corrections of the local clock value. When the time synchronization protocol wants to initiate a message exchange with another node in order to synchronize with it, BTSP checks to see if this message exchange is actually necessary (based on the level of accuracy that the application needs). BTSP allows the message exchange to proceed only if the quality of service required by the application is in danger of being violated. Otherwise, the message exchange does not occur.

1.3 The Solution Approach

Any time synchronization protocol if it has to applicable for wireless sensor network has to provide a precision in the range of milliseconds and still consume less energy. Most of the existing time synchronization protocols require message exchange with in the network to maintain synchronization. Every protocol has a different approach to attain synchronization in the network. So a solution which can address the problem of energy consumption for any type of time synchronization protocol is required.

A wrapper based generalized solution has been designed which can be used by an application with any time synchronization protocol. The idea of wrapper based solution is, there have to be minimum changes made to the time synchronization protocol and it should not degrade the performance of the protocol.

1.4 Contributions

The main contribution of this work is it provides a wrapper based software component which can be used by most of the existing time synchronization protocols. The wrapper reduces the number of messages used by the time synchronization protocol without degrading the efficiency of the protocol. In short, the **BTSP** wrapper reduces the energy consumption of the time synchronization protocol and increases the life time of the network.

Another important energy saving contribution by the **BTSP** wrapper is, it allows the application the freedom to set the limit on synchronization error tolerable, which most of the time synchronization protocols do not provide.

1.5 Organization of the Thesis

The thesis is organized in the following way. Chapter 2 contains brief look at the architecture of Wireless Sensor Network. Chapter 3 explains about time synchronization and its need. Chapter 4 explains about the working style of the **BTSP**. Chapter 5 includes the experiments methodology, results and discussion. Chapter 6 reviews similar research. Finally, Chapter 7 contains the conclusions of the results of **BTSP**.

CHAPTER II

WIRELESS SENSOR NETWORK

Wireless Sensor Networks (WSN) can be defined as collection of low cost, low power devices which collectively do a common task. Each tiny embedded device which is used in a wireless sensor network is called a **Mote**. Deploying an application using such constrained resources needs high precision and less resource consuming protocols to achieve accurate results. Each mote has different type of sensors like humidity, temperature, light, motion etc. embedded on them. When ever there is a change in the surroundings which is above or below the normal conditions the sensor generates an impulse which denotes the amount of change in the environment. The sensed change by the sensor is sent for further processing by using an in built radio on the mote.

2.1 Hardware Specification of a Mote

The tiny embedded devices have limited resources using which they perform sensing, computation, communication. The size of these devices is small as can be

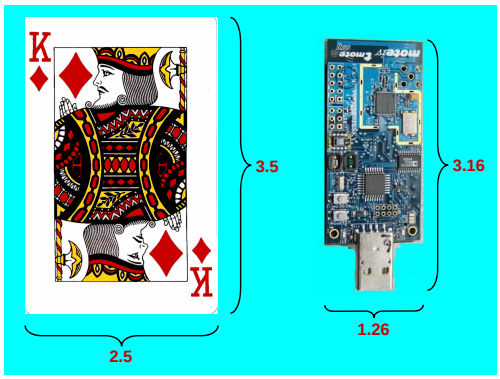

Figure 1: *Shows the size of the mote against a playing card.*

seen in Figure 1 .Some of the widely used different type of motes are Mica, Mica2, Telosb, T-Mote, I-mote, sunSPOT, extreme scale mote (**XSM**). Different motes have different types of hardware embedded on them but the amount of resources available on most of the motes is more or less the same. Some of the hardware specifications of the XSM motes are shown in Table I [20] [23]. **BTSP** Wrapper is implemented on XSM motes (see Section 5.2).

2.2 Introduction to TinyOS

TinyOS is a free and open source component-based operating system targeted for wireless sensor networks . TinyOS is an embedded operating system written in the nesC programming language as a set of cooperating tasks and processes. TinyOS applications are written in nesC, a dialect of the C programming language optimized for the memory limitations of sensor networks [18] [19].

TinyOS is an event driven and component based operating system. Components are connected to each other using interfaces. TinyOS provides interfaces and components for common abstractions such as packet communication, routing, sensing, actuation and storage.

Processor/Radio board	**MPR400CB**	**Remarks**
Processor Performance		
Program Flash Memory	128K bytes	
Measurement (Serial) Flash	512K bytes	> 100,000 Measurements
Configuration EEPROM	4K bytes	
Serial Communications	UART	0-3V transmission Levels
Analog to Digital Converter	10 bit ADC	8 channel, 0-3V input
Other Interfaces	DIO, I2C, SPI	
Current Draw	8 mA	Active mode
	$< 15\mu A$	Sleep mode
Multi-Channel Radio		
Center Frequency	868/916 MHz	ISM bands
Number of Channels	4/ 50	Programmable, country specific
Data Rate	38.4 Kbaud	Manchester encoded
RF Power	-20 to +5 dBm	Programmable, typical
Receive Sensitivity	-98 dBm	Typical, analog RSSI at AD Ch. 0
Outdoor Range	500 ft	1/4 Wave dipole, line of sight
Current Draw	27 mA	Transmit with maximum power
	10 mA	Receive
	$< 1\mu A$	Sleep
Electromechanical		
Battery	2X AA batteries	Attached pack
External Power	2.7 - 3.3 V	Connector provided
User Interface	3 LEDs	User programmable
Size (in)	2.25 x 1.25 x 0.25	Excluding battery pack
(mm)	58 x 32 x 7	Excluding battery pack
Weight (oz)	0.7	Excluding batteries
(grams)	18	Excluding batteries
Expansion Connector	51-pin	All major I/O signals

Table I: *Hardware specifications of XSM motes.*

2.3 Application of Wireless Sensor Networks

Consider a deployment of randomly placed motes, which have same amount of resources, are used for wild life monitoring (see Figure 2). All the randomly placed motes form a network to sense movement of wild life animals. All the motes when ever they sense some change in the near by surroundings, send the sensed data to sink, which has better resources compared to rest of the motes. As a part of resource saving, applications instead of sending a sensed data directly to the sink it chooses to pass the date to a neighboring node which is near to the sink and the neighboring node then passes the data to it neighbor which is near to sink. This process continues and data is finally sent to the sink i.e, the nodes form a multi-hop network.

In fact the sink acts a data collector and also as a beacon broadcaster in the network. Sink after receiving the data from the motes, sends this data to a remote base station, which has high processing resources and where further operations are done on the received sensed data from the network. In some cases the sink and base station are the same.

As seen in Figure 2, a mote sense a tiger entering into sensing range by the infrared sensor embedded on it. The mote senses the motion of the tiger and sends the sensed data to the sink via its neighboring motes. The sink then sends this data to the base station. At the base station from all the collected sensed data few conclusions can be drawn like motion of wild life animals, density of wild life animals, area of maximum densities at different time of day, habitant conditions etc.

Apart from wild life monitoring there are many applications where wireless sensor network can be used for doing a smart job like traffic monitoring, target tracking, military applications, environmental monitoring, security and surveillance applications, home monitoring, habitant monitoring etc. Using different type of sensors on the motes, make wireless sensor networks applicable for different type of applications.

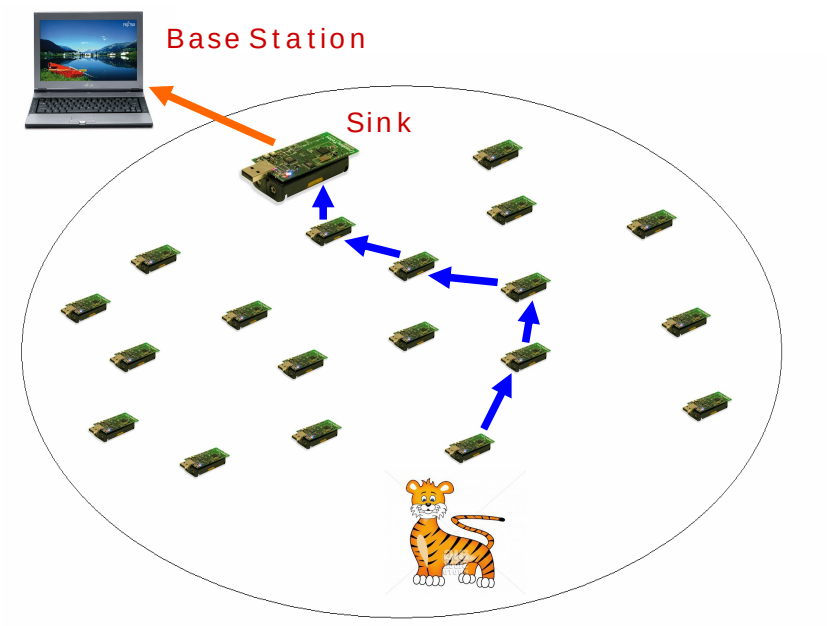

Figure 2: *Shows deployment of wireless sensor network for wild life monitoring where a mote sense a tiger entering into its surroundings and sends the sensed data to the sink.*

CHAPTER III

TIME SYNCHRONIZATION

In wireless sensor network setting, the hardware components of these small, inexpensive motes are bound to be imperfect. One such imperfection is the usage of low cost crystal oscillator, which leads to different operating frequency from mote to mote. The difference in the operating frequency is in the range of few kilo hertz (kHz) which results in the time difference in the range of few milliseconds (ms). This causes tremendous decrease in the accuracy of the deployed application. Another effect of using a low cost crystal oscillator is, as the physical environmental conditions like humidity, temperature etc. change the frequency of operation of the mote also changes.

In any wireless sensor network deployment, as the time increases the input power supply to the mote decreases which causes change in the frequency of the operation of the crystal oscillator. Due to all these effects different mote have different frequency of operation from time to time, as a result local clock time varies from mote to mote. The difference in local clock times varies in the range of microseconds to seconds in a network at any given time.

Apart from above mentioned factors, another factor which causes the difference between local times of the motes is, initial time delay, which occurs when motes start initial operation at different times. This initial time delay is called as **clock skew**

3.1 Effect of lack of Time Synchronization

All the factors mentioned above result in having different local times at different motes, but how far the local time difference can effect the operation of any application is an important question. To answer this question, consider a wireless sensor network which is deployed for traffic monitoring application along side a road (as shown in Figure 3).

Deployment consists of motes which have infrared sensor embedded on them. Infrared sensors are used to detect the motion on any object with in the line of sight. There is a sink which collects the data from the motes and sends the collected data to the base station, which analyzes the received data. The application running on the motes is designed in such a way that whenever motes detect a motion they record their local time at the time of event and send the recorded local time to the sink. Using the forwarded data from sink, the base station calculates velocity, density etc. of the traffic.

A car enters the range of the sensor, the motes record their local time stamps (see Figure 4). After recording the local time stamps all the motes send the data to the sink (see Figure 5). Sink then forwards the data to the base station. Base station observes from the collected data that, local time stamps from different mote does not lead to any useful conclusion due to the fact that different motes have different local times at the given instance of time, i.e due to lack of synchronization of local clocks between motes, the collected data becomes useless.

Figure 3: *Shows the deployment of wireless sensor network for traffic monitoring application.*

Figure 4: *Motes sense the motion of the car and record their local time at the time of event.*

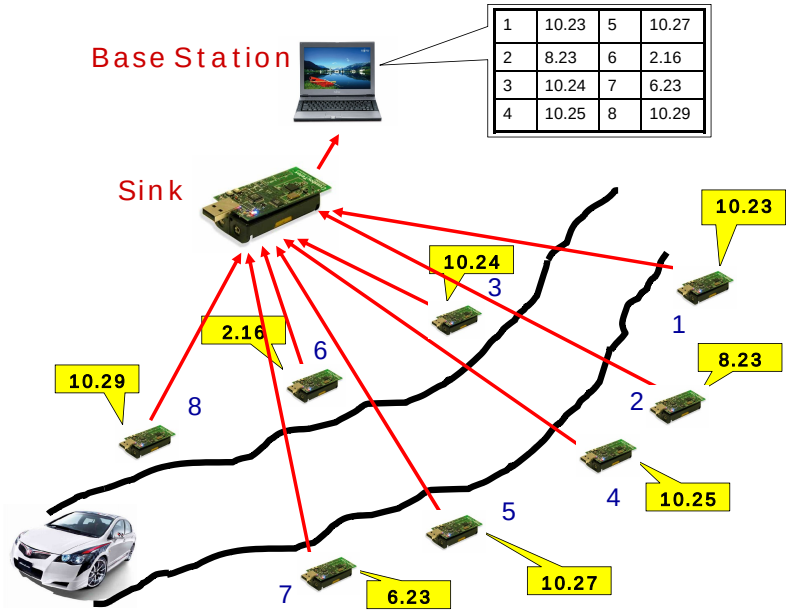

Figure 5: *Motes send their local time stamps at the time of car passing through them to the sink. Sink in turn send the collected data to the base station, where it analyzes the received data.*

3.2 Requirements of Time Synchronization Protocol

The time synchronization protocols to be applicable to these low power networks then have to meet the following requirements:

- The protocol should be able to maximize time synchronization precision among the motes.
- The functioning of the protocols should not end up using more power to maintain synchronization in the network.

3.3 Type of Time Synchronization Protocols

Many algorithms have been proposed keeping above requirements into consideration and achieved a high precision in the range of micro seconds [2] [3] [4] [5] [6] [8] [12] [13] [14]. Every time synchronization protocols depend upon message exchange between the motes to achieve the process of synchronization. The efficiency of the time synchronization protocols depends upon the amount of precision it achieves and number of message exchanges it uses to achieve synchronization. Time synchronization protocols can be broadly divided into two categories depending upon the process of achieving synchronization by the protocol. They are

- Symmetric
- Asymmetric

In symmetric protocols, mote which initializes the synchronization process and mote which responds to the initiator mote both send and receive messages as part of synchronization. In short protocols which involve in bi-directional message exchange are symmetric protocols. Pair wise synchronization is one of the examples of symmetric protocols. Protocols which come under the category of symmetric protocols are TPSN [3] , delay measurement time synchronization [14], tiny sync [6] etc.

Protocols in which there is only unidirectional message exchange as a part of synchronization, those protocols are termed as asymmetric protocols. Some of the widely used asymmetric protocols are FTSP [4], RBS [2], converge to max [13]etc.

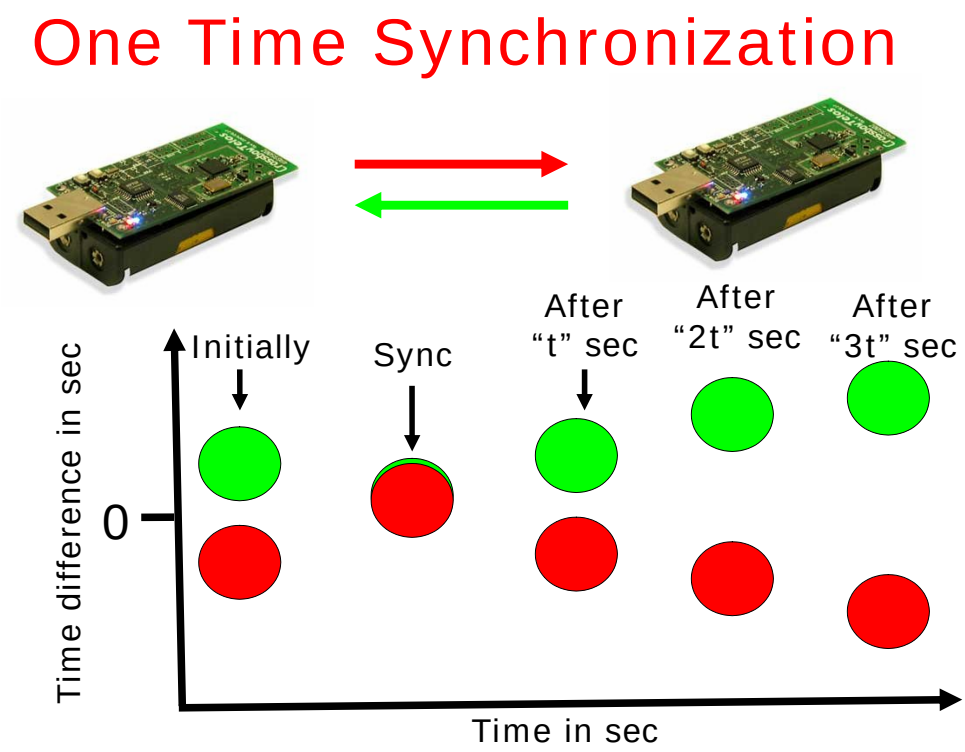

Figure 6: *Shows how one time synchronization, still makes motes ending up with different local time after certain period of time.*

3.4 Problem with Current Time Synchronization Protocols

Consider two motes running some time synchronization protocols to get synchronized. Motes initially perform process of synchronization and get synchronized to some level of precision and as time goes on, the time difference between two motes increases due to hardware limitation (discussed in Section 2.1) . The time difference at a give time between two motes after the process of synchronization is called as **Drift**. As a result, the effect of doing synchronization process vanishes and time difference between the motes tends to increase. In short, drift increases as time goes on (see Figure 6).

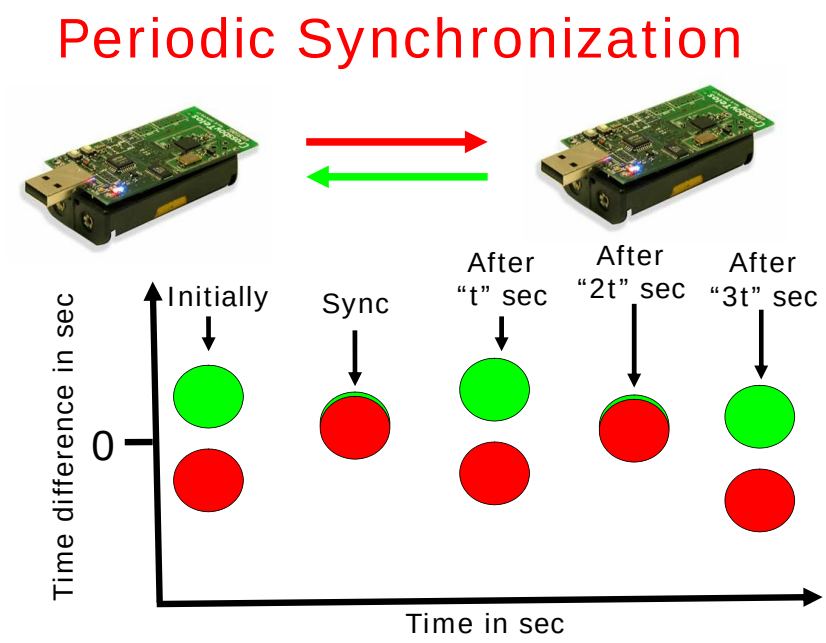

Figure 7: *Shows how periodic synchronization can keep motes synchronized.*

The existing solution to this problem is **periodic synchronization**, which is part of every time synchronization protocol. In periodic synchronization, motes periodically perform process of synchronization. The period after which motes perform the process of synchronization is called as **period of synchronization**. As a result of doing periodic synchronization time difference between two motes at any given time is limited(which depends upon on period of synchronization)(see Figure 7).

Periodic synchronization solution to maintain nodes in synchronization ends up as a costly solution as nodes periodically have to do message exchange as a part of synchronization which results in lot of power consumption. The only way power expenditure can be reduced is by actively reducing the number of messages being sent out. This problem of power loss due to message exchange can be reduced by use of software component, called Booster for time synchronization protocol (**BTSP**).

CHAPTER IV

BOOSTER FOR TIME SYNCHRONIZATION PROTOCOL

After all, every time synchronization protocol is trying to do its job by keeping the nodes as tightly synchronized as possible by doing periodic synchronization. It is the application that decides on what level of synchronization is required, and this level of accuracy can change during the lifetime of the application deployment. Further, the application only cares that the quality of service it requests is actually available. It does not care about how it is achieved. In particular, if the same level of accuracy can be provided by performing local computations, this is preferred, since the lifetime of the nodes battery source can be extended.

This thesis propose a software component, called Booster for Time Synchronization Protocol (**BTSP**) [1] that continually monitors the drift between two nodes, and performs internal corrections of the local clock value. When the time synchronization protocol wants to initiate a message exchange with another node in order to synchronize with it, **BTSP** checks to see if this message exchange is actually necessary (based

on the level of accuracy that the application needs). **BTSP** allows the message exchange to proceed only if the quality of service required by the application is in danger of being violated. Otherwise, the message exchange does not occur. Given the variety in different time synchronization protocols, it is designed in way that this behavior to be outside of the context of the protocol itself. Otherwise, to apply this wrapper, one should revisit the implementation of every time synchronization protocol in use. Accordingly, **BTSP** has been designed as a wrapper (similar to the Decorator design pattern [21]) around the component that manages messaging on the node. As such, the implementation of **BTSP** is a drop-in replacement for radio interface providing component, and the time synchronization protocol does not need any modification in its implementation other than using this component, and initializing it with the right parameters.

4.1 BTSP Algorithm

Any time synchronization protocol has a process of maintaining synchronization between a pair of nodes or in the entire network. The main function of the **BTSP** wrapper is to improve the performance of the time synchronization protocol without degrading its accuracy or efficiency. The **BTSP** wrapper is designed to work with any time synchronization protocol that involves pair-wise collaboration between nodes. For example, the Timing-sync Protocol for Sensor Networks (TPSN) [3] works by exchanging clock information between two nodes. Similarly, in the Reference Broadcast Synchronization (RBS) scheme [2], nodes compare the recorded timestamps of a reference broadcast sent by a beacon. By contrast, in the Flooding Time Synchronization Protocol (FTSP) [4], nodes synchronize based on the roots broadcast. The current design of **BTSP** is not directly applicable to such a protocol.

Consider a time synchronization protocol that fits the aforementioned profile.

Suppose that the protocol mandates that a synchronization action, in the form of a message exchange, is to be performed at the rate of every T_P seconds. Further suppose that the maximum synchronization error (between any two nodes) allowable for the application under consideration is D_{Limit}. The following description shows how the **BTSP** wrapper will interact with the time synchronization protocol in use, and how the wrapper will sustain time synchronization while reducing message exchanges. This behavior is captured in Figure 8. For the ensuing discussion, lets us assume that two motes A and B are participating, and further, we will describe the actions from the point of view of node A.

The **BTSP** wrapper first allows the time synchronization protocol (TSP) component to establish synchronization, through whatever means it uses. This might be a simple handshake in a protocol such as TPSN, or a reference broadcast followed by a comparison in a protocol such as RBS. Regardless, this first step is essential. At this time, motes A and B are apart with a minimum synchronization error of D_1.Following this, the **BTSP** wrapper forces the node to be quiet and does not allow the node to send any messages for a set period of time that we call the training period, $T_{Training}$. This $T_{Training}$ is chosen to be a multiple of T_P, and so, at the end of $T_{Training}$, the node participates in a message exchange.

As a result of this message exchange with node B, A can again calculate the relative drift between A and B. Let us call this drift D_2. Using these two drift values, the average drift (D_{Avg}) per second can be calculated as:

$$D_{Avg} = \frac{D_2 - D_1}{T_{Training}} \tag{4.1}$$

Using this average drift per second, and the limit on synchronization error (provided by the application) (D_{Limit}), the **BTSP** wrapper can estimate how long it is safe not to participate in a handshake message exchange.

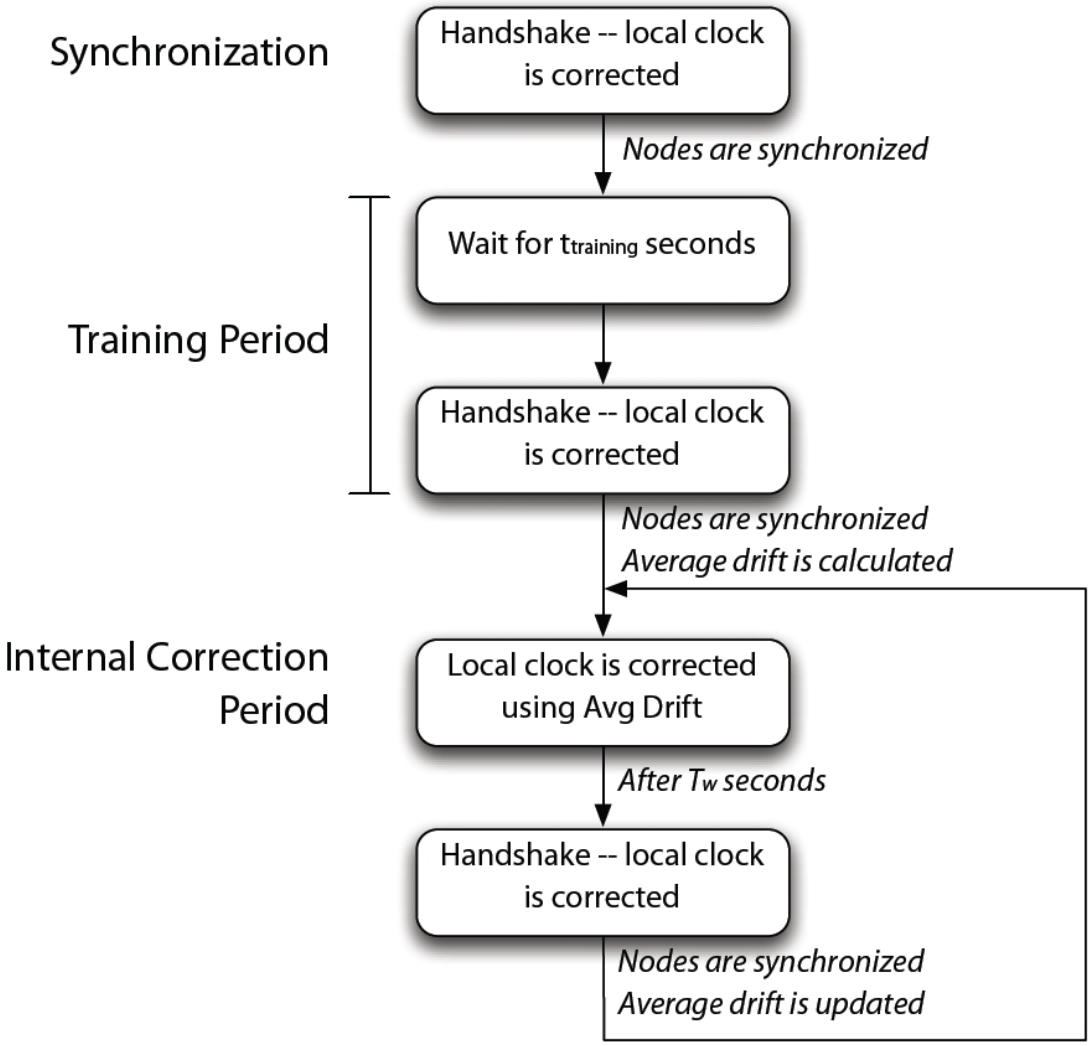

Figure 8: *BTSP Timeline.*

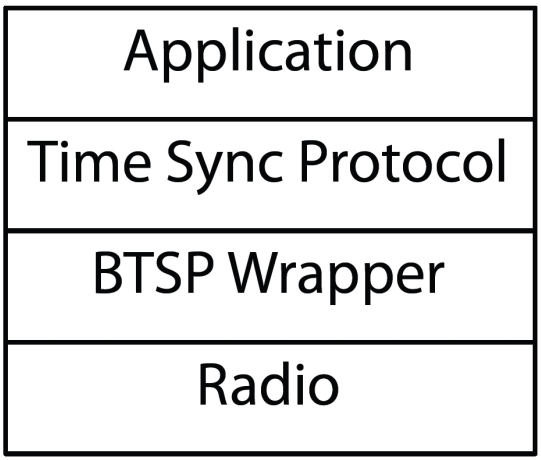

Figure 9: *Time Synchronization stack at each node.*

$$T_W = \frac{D_{Limit}}{D_{Avg}} \tag{4.2}$$

The TSP needs to interact with the **BTSP** wrapper for bootstrapping purposes, but after that it can function without any interaction with **BTSP**. But yet **BTSP** does not change the operations of the TSP. So even though TSP does know about the existence of the **BTSP** wrapper, still at every T_P interval, will initiate a handshake message exchange. This is done by invoking send command on the communication component (normally by send component provided in TinyOS, but since our wrapper has replaced it, our **BTSP** component) (Figure 9). At this point, **BTSP** will intercept this message attempt, and will make the decision of whether or not this message is sent out on the radio. Instead of sending the message, the **BTSP** wrapper will provide feedback to the TSP component as though the message exchange was completed with B. Since it can calculate what the drift should be (based on its training), the response that **BTSP** provides to TSP will not be substantially different from what it would receive from a real message exchange. We refer to this phase of the protocol as the **internal correction period**.

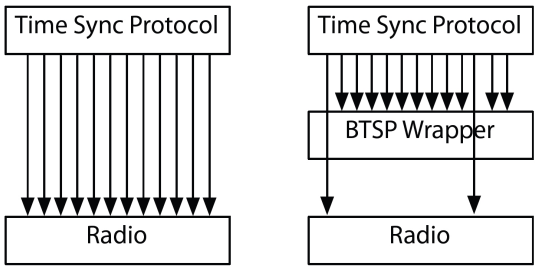

Figure 10: *Message interception by BTSP Wrapper.*

Since the internal corrections that **BTSP** is performing are all based on estimates, there will be some error creep over time. Eventually, in T_W seconds, the **BTSP** wrapper will allow the next message exchange initiated by the TSP to go through. At the end of this new message exchange, the **BTSP** wrapper will learn of a new drift value D_N. This drift value is used to update the current average drift per second (D_{Avg}). Over time, since the **BTSP** wrapper is continuously learning and updating the average drift between the two nodes, the length of time during which message exchanges are not needed (T_W) will likely increase. After the average drift has sufficiently matured, the frequency of message exchange with the **BTSP** wrapper will be substantially smaller than that of the bare TSP ($T_W >> T_P$). Consequently, the number of message exchanges that are required will decrease progressively, thereby increasing the energy efficiency of time synchronization (Figure 10) .

4.2 Application of BTSP Wrapper

Most important point that has to be noted regarding the **BTSP** wrapper is, it never initiates message exchange and it merely acts as a gatekeeper where it checks whether to allow a message to be sent out or not. **BTSP** wrapper does not affect the message structure of the message that has to send out and also it does not interfere while receiving a message. As mentioned in Section 4.1, **BTSP** wrapper can be used

with many existing time sync protocols. The changes that need to be made in existing TinyOS applications in order to leverage **BTSP**s advantages are as follows:

- The first change required in a TSP implementation is that the TSP must bootstrap **BTSP** to mark the specific kinds of messages that are used for synchronization handshakes. The purpose of this bootstrapping is that the TSP component (such as level discovery, tree membership, secure key sharing, etc.), and the application itself (sensor readings, health status, etc.), will send several kinds of messages during the deployment. The **BTSP** wrapper only concerns itself with handshake messages. For this, **BTSP** provides an interface called **TSBootstrap**. Using this interface, the TSP and the application can register the message type(s) that may need to be blocked by **BTSP**. All other kinds of messages are simply let through by **BTSP**.

- The second change that is required is for the application to tell **BTSP** what the tolerable synchronization error is. To do this, the **BTSP** component provides the **SetTolerance** interface. The application can call the SetTolerance.set() command to provide the expected quality of service level to the wrapper.

This tolerance level can be changed during the deployment lifetime as well. For example, suppose that the application were to decide, based on available energy levels, and based on advice from, say, the Energy Management Architecture [22], that the tolerance can be increased, **BTSP** can be reconfigured on the fly to use this new tolerance level.

4.2.1 BTSP Implementation in TinyOS -1.x

In TinyOS-1.x, GenericComm provides send and receive interfaces to the layers above it. The send and receive interfaces in GenericComm are linked with send and

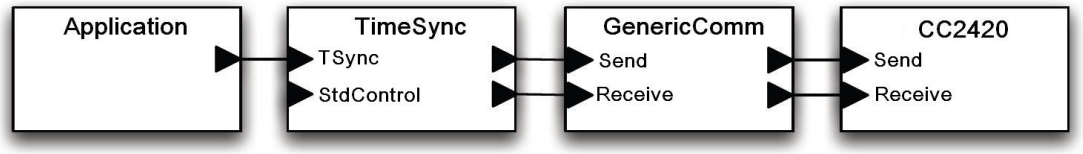

Figure 11: *Wiring diagram for time synchronization in TinyOS-1.x.*

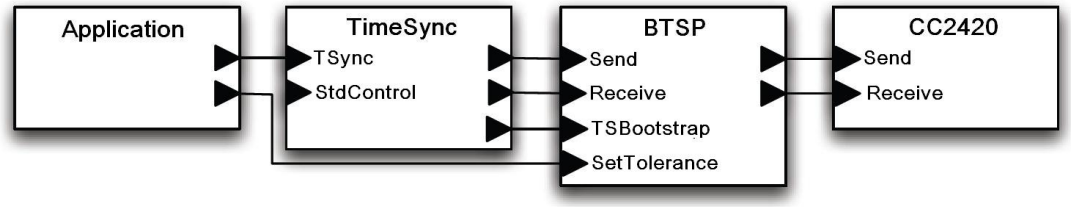

Figure 12: *Wiring diagram for time synchronization in TinyOS-1.x with BTSP Wrapper.*

receive interfaces provided by the radio respectively (see wiring diagram in Figure 11).

The **BTSP** wrapper is implemented as a drop-in replacement for GenericComm in TinyOS-1.x. As such, the **BTSP** component provides all the interfaces that GenericComm does. So using this component does not change the structure of existing TinyOS application. Such applications can be minimally modified to take advantage of **BTSP** with small localized changes(see wiring diagrams in Figure 12).

4.2.2 BTSP Implementation in TinyOS -2.x

In TinyOS-2.x, AMSend and AMReceive provide send and receive interfaces respectively to the layers above it. The send interface in AMSend and receive interface in AMReceive are linked with send and receive interface respectively provided by the radio (see wiring diagram in Figure 13).

The **BTSP** wrapper is implemented as a drop-in replacement for AMSend in TinyOS-2.x. As such, the **BTSP** component provides all the interfaces that AMSend

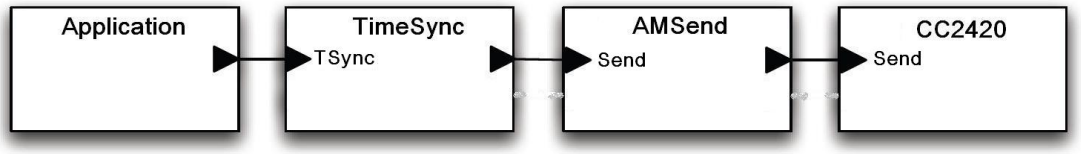

Figure 13: *Wiring diagram for time synchronization in TinyOS-2.x.*

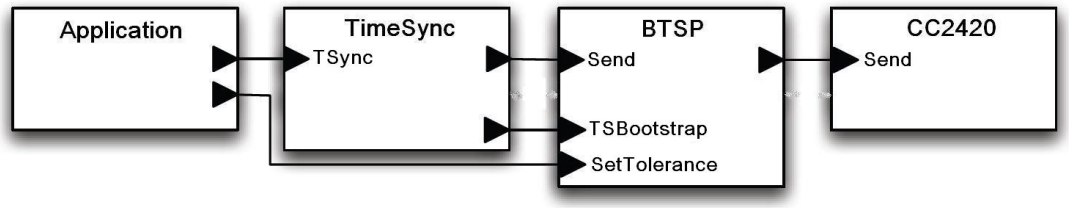

Figure 14: *Wiring diagram for time synchronization in TinyOS-2.x with BTSP Wrapper.*

does. So using this component does not change the structure of existing TinyOS application(see wiring diagram in Figure 14). The AMReceive component remains the same as **BTSP** wrapper is not concerned with receiving of messages.

CHAPTER V

IMPLEMENTATION AND RESULTS

In previous chapter (Chapter 4) we learn how to make any existing time synchronization protocol take advantage of features provided by **BTSP** wrapper. In this chapter, we will evaluate the performance of time synchronization protocol with and without use of **BTSP** wrapper.

5.1 Implementation of BTSP

The implementation of the **BTSP** wrapper is itself composed of two parts. The majority of the wrappers implementation is agnostic to the particular time synchronization algorithm being used. However, a small portion of the implementation does depend on the particular protocol being utilized. This protocol-specific part is where the wrapper is customized to learn about the protocols behavior.

Our evaluation of **BTSP** is based on using TPSN as the time synchronization protocol. TPSN is a symmetric protocol: a pair of nodes A and B are executing the same set of actions relative to each other. In a different protocol, such as RBS,

which depends on reference broadcasts, and relative differences that nodes A and B see with respect to the beacon, the bootstrapping will change a little. However, all the protocol-specific parts of the wrapper implementation are localized to the TSBootstrap interface. The other commonly-used interfaces (Send and Receive, in particular) are protocol-agnostic and do not change with the particular time synchronization protocol being used.

5.2 Results

It is hard to predict the amount by which a pair of nodes will differ from each other after certain amount of time as every node works at different oscillating frequency due to hardware limitations. Using a time synchronization protocol we can only set the period of synchronization but not the limit on synchronization error. However, by using the **BTSP** wrapper, an application has the luxury of setting the limit on the amount of synchronization error that can be tolerated. To evaluate the efficiency of the **BTSP** we have tested it with TPSN on the XSM (extreme scale motes) [23](Figure 15) for single hop as well as multi-hop networks.

5.2.1 Efficiency of the BTSP wrapper

Before we begin to discuss the energy savings of using the **BTSP** wrapper in conjunction with TPSN, we first need to establish that **BTSP** does not degrade the efficiency and accuracy of TPSN in any way. Here efficiency refers to the average synchronization error a protocol can achieve after a given process of synchronization. Figure 16 shows the minimum synchronization error achieved by bare TPSN and the **TPSN/BTSP** combination. As one can see from the Figure 16, the average synchronization error is almost identical. This means that the **BTSP** wrapper indeed maintains the same baseline as the bare implementation of TPSN alone. The protocol,

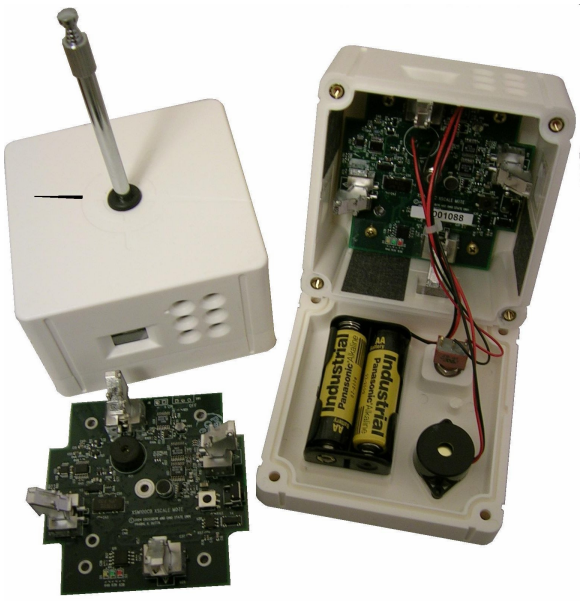

Figure 15: *XSM Mote* [24].

as explained in Section 4.1 does not do anything to interfere with the synchronization process, except reduce the number of handshake message exchanges.

5.2.2 Message Complexity with Constant T_P

Now that we have established that **BTSP** does not degrade TPSNs efficiency, we can proceed to profile the performance improvements that **BTSP** enables. In this first test, we fixed the synchronization period for TPSN (T_P) to be a constant. Then, we varied the maximum tolerable synchronization error (D_{Limit}). For different values of D_{Limit}, we measured the number of handshake messages sent out using bare TPSN, and **TPSN/BTSP** over an application deployment of one hour. The results are shown in Figure 17. When TPSN is used without **BTSP** wrapper it has to do 60 handshakes during the one hour irrespective of the limit of synchronization error. However, when **BTSP** is used, it tries to maintain time difference between nodes within the limit of synchronization error by replacing some of the handshake message exchanges by internal corrections of the clock using the estimated value of

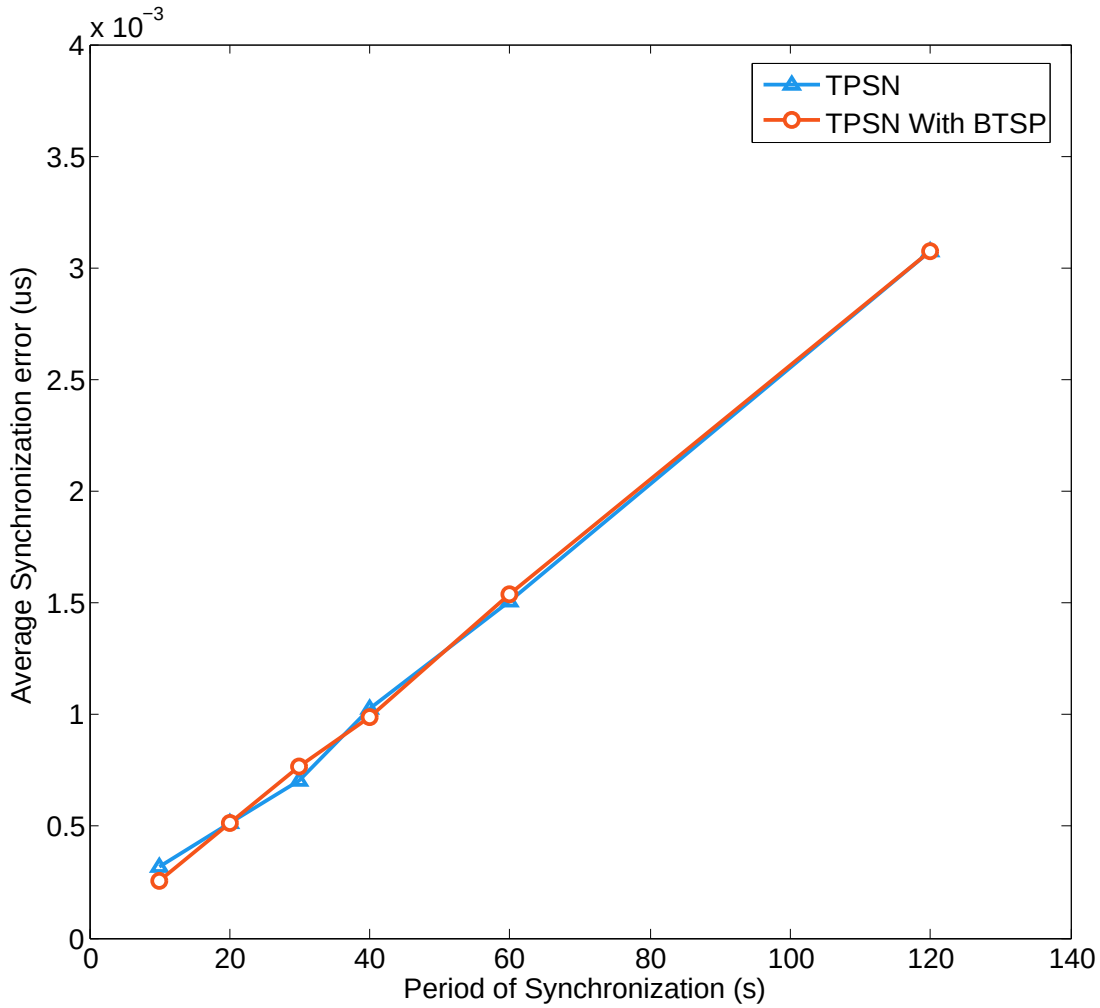

Figure 16: *Efficiency of TPSN with BTSP wrapper.*

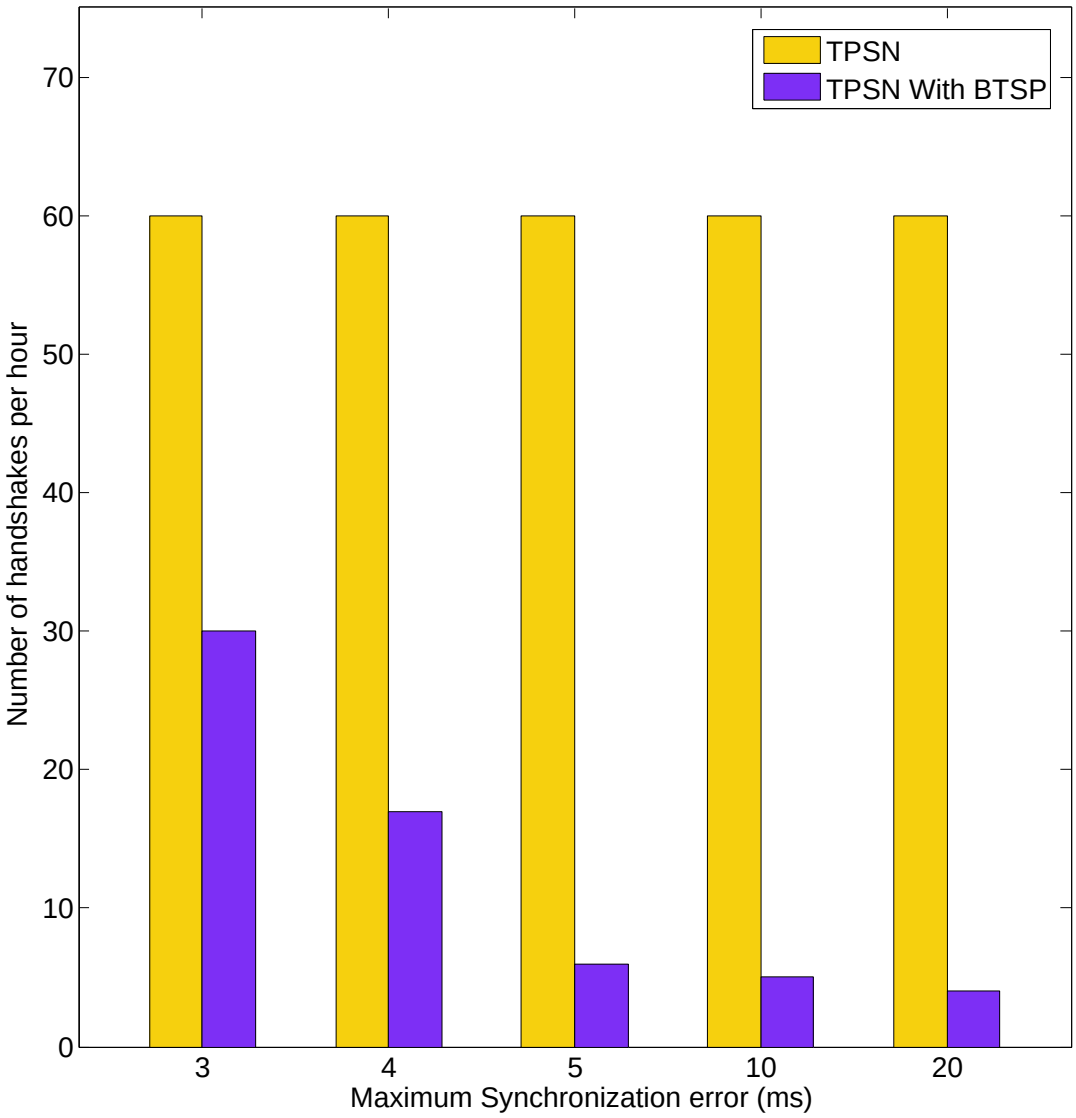

Figure 17: *Message complexity of bare TPSN and TPSN/BTSP for constant period of synchronization.*

average drift (D_{Avg}). As a result, it limits the number of handshakes. The number of handshakes required by the TPSN when used along with the **BTSP** wrapper decreases as the tolerance limit of the synchronization error increases, while TPSN has no way of such dynamic adaptation.

5.2.3 Message Complexity with Constant D_{Limit}

Next, we look at the number of handshake messages that are sent out with a constant allowed synchronization error by the application (D_{Limit}) and varying periods of synchronization. Figure 18 shows the number of handshake messages sent out in a deployment (one hour) with T_P varying from 20 seconds to 60 seconds. As the period of synchronization increases, the number of messages sent out by TPSN reduces (as expected). But in all cases, the number of messages sent out by the **TPSN/BTSP** combination is only a fraction of TPSNs message complexity this is due to the fact that **BTSP** blocks most of the messages from being sent out by still maintaining the synchronization error with in the allowed limit.

5.2.4 Energy Consumption

Lets examine the amount of energy consumed in maintaining time synchronization. TPSN uses handshakes to maintain synchronization in the network. A handshake involves messages exchanged between two nodes (two messages in total). Consider a newly-deployed network and all the motes are powered with a input of voltage of 3V. The XSM mote requires 10.4mA and 7.4mA of current to send and receive a message respectively [20],

$$\begin{aligned} Power\ required\ to\ send\ a\ message &= Voltage * Current \\ &= 10.4mA * 3V \\ &= 31.2mW \end{aligned}$$

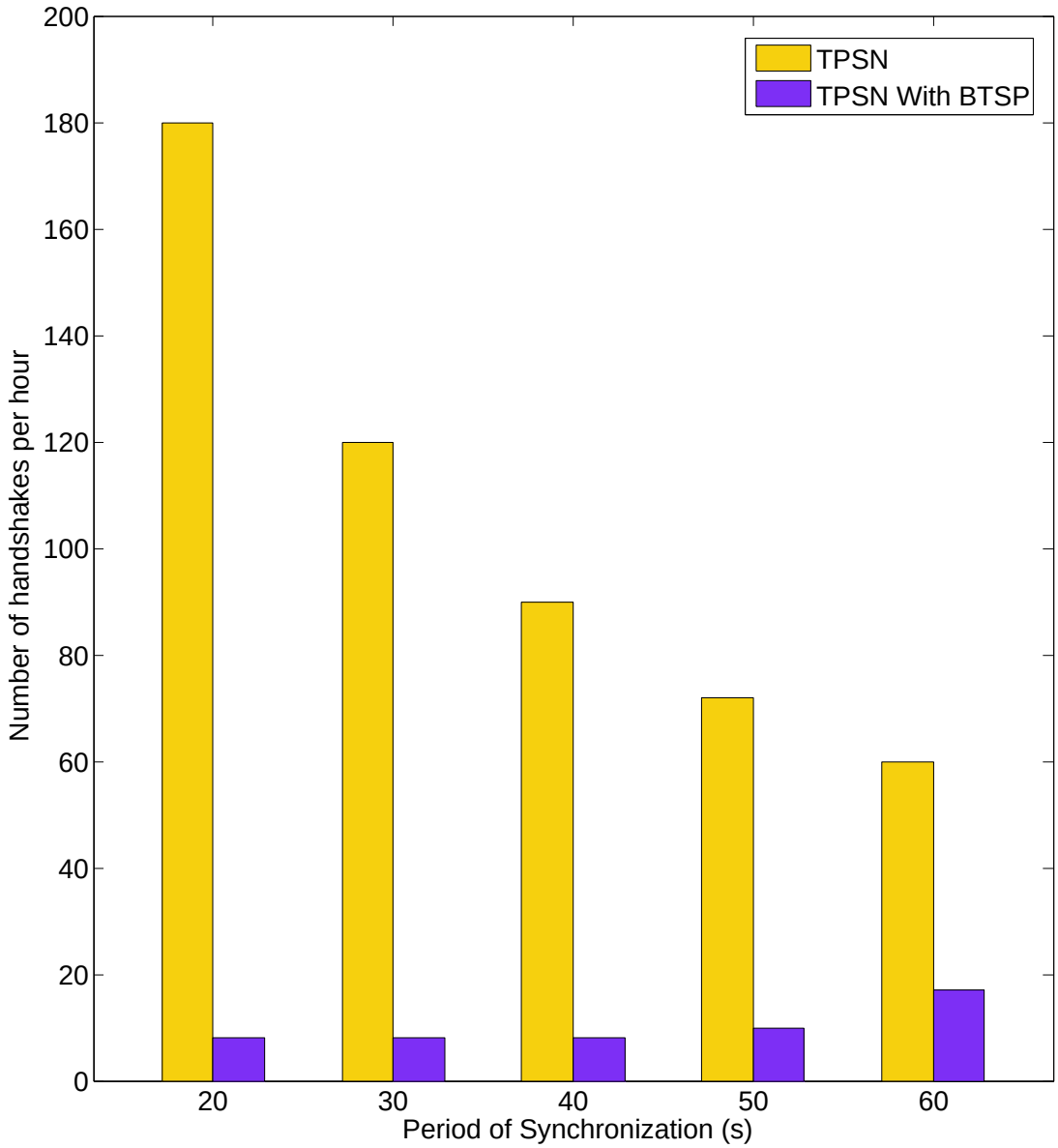

Figure 18: *TPSN with and Without BTSP wrapper for a constant limit of synchronization error.*

$$\begin{aligned} Power\ required\ to\ receive\ a\ message &= Voltage * Current \\ &= 7.4mA * 3V \\ &= 22.2mW \end{aligned}$$

A handshake involves node A sending message to node B and it replies back with a message. So a handshakes involves two send and two receive process in the network.

$$\begin{aligned} Power\ consumed\ for\ one\ handshake = {} & 2 * Power\ required\ to\ send \\ & + \\ & 2 * Power\ required\ to\ receive \\ = {} & 2^*\ 31.2\text{mW} + 2^*\ 22.2\text{mW} \\ = {} & 106.8\text{mW} \end{aligned}$$

As **BTSP** stops the node from doing the handshake, it not only saves energy at the sending node but also in the receiver node. **BTSP**, in short, reduces the energy consumption in the entire network and, consequently, increases the lifetime of the network.

Figure 19 shows the energy consumption by TPSN with and without **BTSP** for constant T_P and for varying D_{Limit}. TPSN without **BTSP** consumes same amount of energy irrespective of the synchronization allowed, since TPSN lacks the capability of dynamic adaptation based on D_{Limit}. The energy consumption of TPSN with **BTSP** decreases as synchronization error tolerance increases as fewer handshakes are required.

5.2.5 Accuracy of Calculated Drift

The next measure of importance is how close the **BTSP** wrappers calculation of drift between two nodes comes to the actual drift observed by TPSN. Figure 20 shows this comparison. As the period of synchronization increases, the actual drift

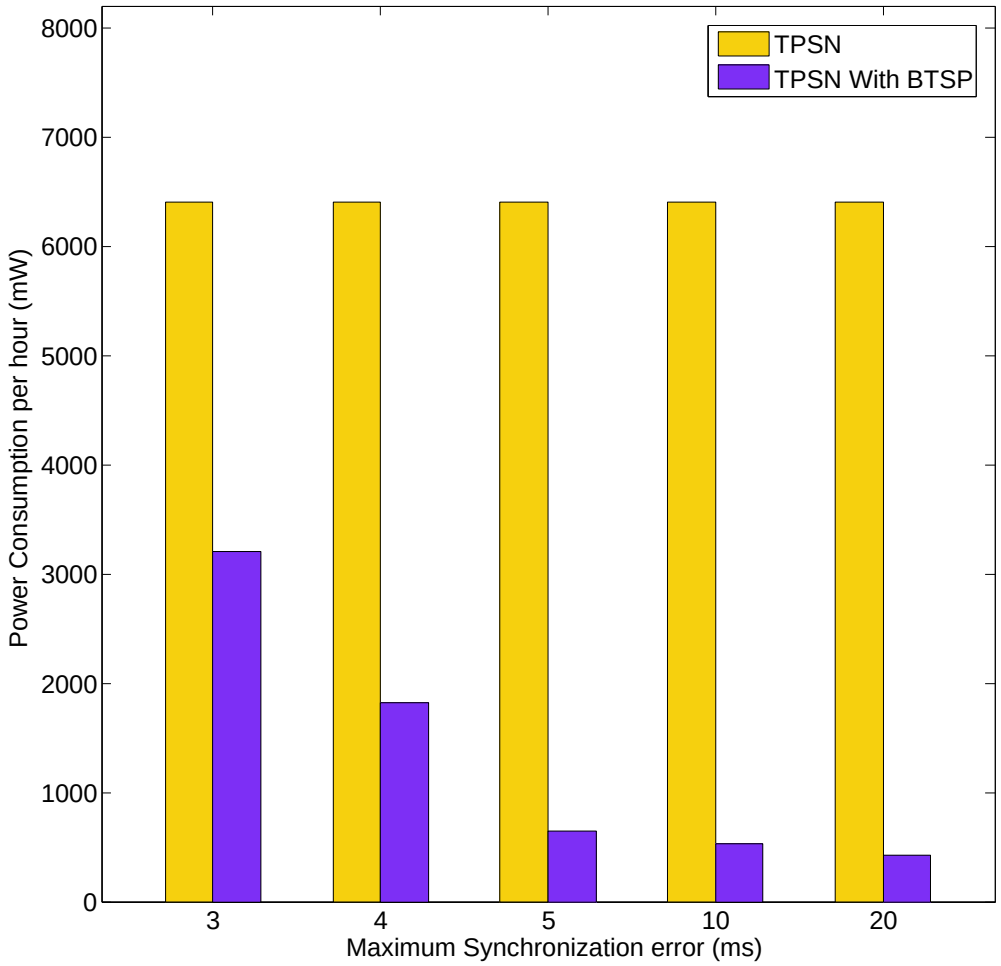

Figure 19: *Energy consumptions of TPSN with and without BTSP Wrapper.*

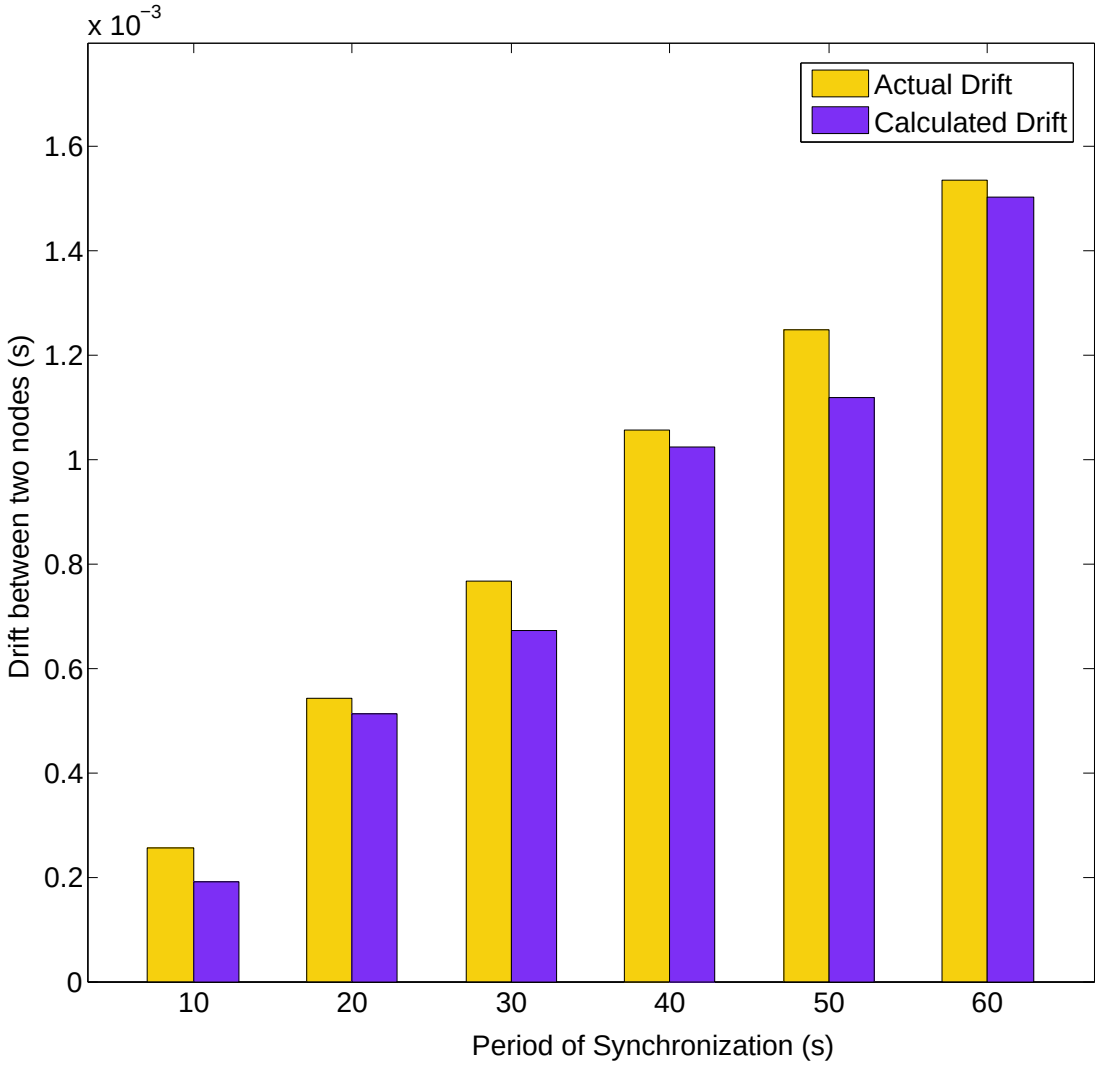

Figure 20: *The actual drift observed from handshakes in TPSN compared with the drift calculated by the BTSP wrapper between a pair of nodes for different periods of synchronization.*

measured by TPSN also increases. Of note, however, is how closely the drift calculated by the **BTSP** wrapper follows the actual drift. This measure is extremely important. If the **BTSP** wrapper were not as accurate, then the error creep in drift will begin to adversely affect the performance of time synchronization, even to the point of rendering it useless with respect to the tolerance limit set by the application. The relative drift between two nodes varies in a non-linear fashion. Towards the end of the lifetime of nodes, the oscillating frequency decreases and relative drift between nodes can change gradually. The **BTSP** wrapper does its correction, in a cumulative linear fashion. The correction value changes after every internal correction period. The **BTSP** wrapper tries to get a better estimate of the present drift between two nodes.

5.2.6 Accuracy of Drift over Multiple Hops

In a multi-hop network setting, TPSN forms a spanning tree within the network, and then initializes the process of synchronization. The drift between the root node and other nodes in the network increases as the number of hops increases. This is because of the compounding effect of error in the drift computation. Figure 21 shows the comparison of actual drift as measured by bare TPSN, and the drift calculated by the **TPSN/BTSP** combination. Note how, even as the number of hops increases, the accuracy of drift calculation does not waver.

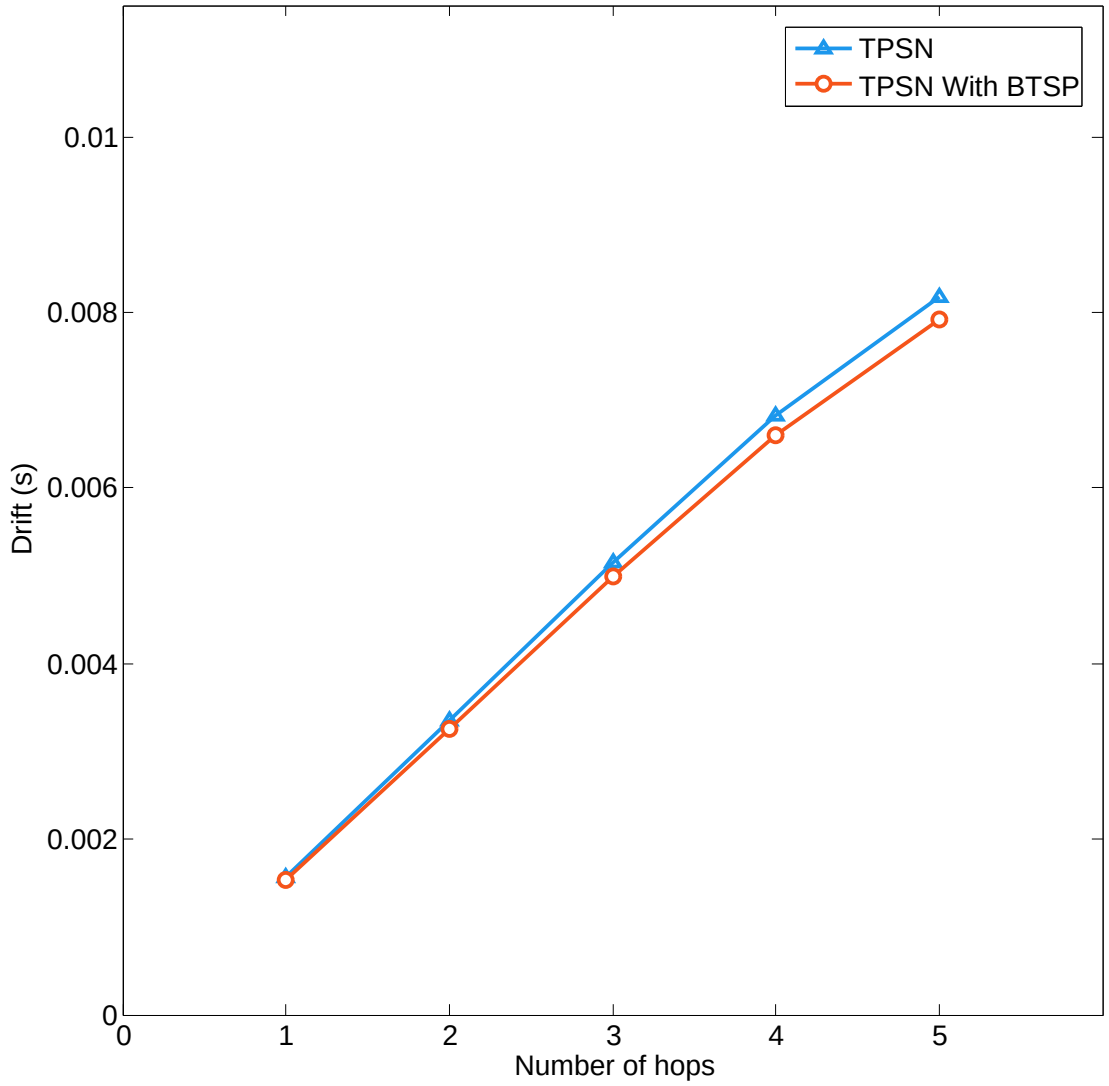

Figure 21: *The actual drift observed from handshakes in TPSN compared with the drift calculated by the BTSP wrapper between root node and node in different hops.*

CHAPTER VI

RELATED WORK

For a time synchronization protocol to be applicable to wireless sensor network it should possess some basic characteristics like less energy consumption, scope, precision, lifetime and cost [7] [17]. **BTSP** Wrapper works on the same guidelines by decreasing the number of messages exchanged to maintain the time synchronization without affecting the precision of the time synchronization protocol.

In Reference Broadcast Synchronization [RBS] [2], a beacon broadcasts a reference message. At the receiver side, the time of arrival of the message is recorded. All the receivers exchange their recorded time stamp and the relative offset between the nodes is calculated. It is a receiver - receiver synchronization. The major advantage of RBS is it does not require any time stamping at the sender side. Its major drawback is, the level of accuracy is low and its a relative synchronization.

TPSN [3] proposes a simple but a very effective way of synchronization between a pair of nodes. It is sender-receiver synchronization. It attain's synchronization between a pair of nodes by exchanging MAC layer time stamped message between sender and receiver. It does achieve a twice a better performance than RBS. The

major advantage with TPSN is, receiver is trying to get synchronized with the sender and it eliminates few delays by using MAC layer time stamping. Its disadvantage is network needs to form a spanning tree and is not suitable for dynamic networks. When TPSN is used along with the **BTSP** Wrapper it blocks the message the sender tries to send in order to maintain synchronization. TPSN performance increases when used with **BTSP** Wrapper without decreasing the efficiency (refer results Section 5.2).

FTSP [4] achieves synchronization using flooding of messages in the network. It attains an impressive high accuracy by using a MAC layer time stamping and using a linear regression to remove clock drift and offset.

Time diffusion Synchronization Protocol [TDP] [5] proposes a way of synchronizing the whole network instead of a pair of nodes. In TDP, sink broadcasts the time stamped messages and the randomly selected master nodes relay it to its neighboring nodes. The neighboring nodes reply back to the master node, which calculates the average deviation using the reply message. The master node sends the average deviation to its neighboring nodes. The major advantage with this protocol is it maintains synchronization even in presence of mobility but uses a lot of message exchange between the master node and its neighbors. **BTSP** Wrapper increases the efficiency of the TDP by blocking the messages sent by the master node.

Reach back firefly algorithm [8] presents a way of achieving synchronicity rather synchronization in the network inspired from the firefly, which means nodes are relatively synchronized rather than global synchronization. In [9] authors proposes Elapsed Time of Arrival [ETA] which is a sender- receiver time stamping service and based on ETA two more canonical services have been proposed Routing Integrated Time Synchronization [RITS] and Rapid Time Synchronization [RATS] for multi-hop networks.

In [6] author propose Tiny-Sync and Mini-Sync. There is no clock correction

in Time-Sync. The way Tiny-Sync works is, it tries to estimate the drift between two nodes. The nodes perform a handshake with time stamped messages, from these messages we obtain one data point. By performing many handshakes many data points are obtained, using these data points an estimation of relative offset and relative drift are estimated. When ever a new data point is obtained the accuracy of the estimate increases. It stores only two data points and eliminates the useless data points

The main difference between Time-Sync and Mini-Sync is that in Time-Sync when ever a new data point is collected it compares it with presently stored two data points, if it better than the previous ones then it will replace the worst , otherwise it is discarded. When as in Mini-Sync the data point before being discarded or replaced checks whether it is useful in future by doing some calculation. The paper also argues that instead of having global synchronization it is enough to have a level to level synchronization. This method does not do clock correction, it just estimates the relative drift and offset. Tiny-Sync and **BTSP** wrapper work on the same lines where the average drift between the two nodes is calculated based on the past values but the major difference and advantage of **BTSP** over Tiny-Sync is **BTSP** is a wrapper and it can used along with any time synchronization protocol, but Tiny Sync is like any another protocol which tries to maintain synchronization using the past values of synchronization error. **BTSP** Wrapper achieves better performance compared to Tiny-Sync as the former has a better averaging technique.

In [12], the authors present a different way of forming a spanning tree for attaining synchronization in entire network. It proposes a secure way of gathering data. The method of synchronization is same as TPSN the only difference is the formation of spanning tree. It uses fewer messages to attain the spanning tree compared to TPSN. It is sender-receiver synchronization.

Converge to max is an asymmetric clock synchronization protocol [13], which attains synchronization in network by following simple principle of every node adjusting its clock to at least as large as any neighbor. Whenever a node receives a beacon it records the local time at that point. It then compares the recorded local time with the time stamp in the received beacon and adjusts its clock if it is less than received time stamp.It then sends out a beacon with a time stamp of its local time. This procedure of converge to max is considered to fast converging protocol then any another time synchronization protocol. One drawback of this protocol is, when a new node joins the network with highest local time then every clock in the network has to be updated.

Delay Measurement Time Synchronization [14] is another simple time synchronization protocol, with main advantage being using less number of message to attain time synchronization in the network. The biggest drawback of this method is, accuracy attained is far less than some of the existing time synchronization protocols like RBS. In this protocol, a node is elected as a leader and it broadcasts time stamp messages. All the receivers measure the transfer delay and set their time as difference of received time stamp and measured transfer delay. Transfer delay includes transmit time delay, radio propagation time, receiver processing time, sender processing time. As result all the node will synchronized to one node. In case of multi-hop network, nodes in upper level send time stamp messages to nodes in the lower level.

Master-Slave Time Synchronization Architecture [15], point outs about the security issues for masterslave based time synchronization protocols. It proposes an Master Selection Algorithm (MSA) which selects a node to act as a backup for master node in the network, so that it can replace master node in case of its failure. The algorithm proposed adds the failure recovery feature to the existing master-slave based time synchronization protocols. The way MSA works is it selects a node with

high resources to act as a backup.

In [16], authors modify the existing FTSP, so that it can be used for master-slave type of synchronization.FTSP achieves better time synchronization compared to any other existing protocol. FTSP is modified in a such way that a master node periodically transmits the time stamped beacon messages and when the slave node receives it, removes the transition errors and corrects its clock. This modified FTSP is applied to zigbee networks.

BTSP Wrapper has been designed in a way that it can used by time synchronization protocols which achieve global synchronization [10] [11](i.e all nodes tuned to one ideal clock) or relative synchronization [8] (i.e synchronicity). Most of the time synchronization protocols does not provide the application that is using it, the luxury of setting the tolerable limit of synchronization error, but **BTSP** Wrapper work from application point of view and allows the application to set a limit of synchronization error.

CHAPTER VII

CONCLUSION

This thesis work proposes a Booster for Time Synchronization Protocols (**BTSP**) wrapper that is designed to improve the energy efficiency of time synchronization protocols in wireless sensor networks. While many protocols for time synchronization have been proposed in the literature, few have the capability of dynamically tuning themselves depending on network characteristics during a deployment. In zeal to provide as much accuracy as possible, time synchronization protocols frequently tend to be inefficient with respect to energy consumption. **BTSP** wrapper is designed as a drop-in replacement for the messaging component, and intelligently manages when message exchanges are necessary. For the most part, the wrapper uses the history of handshakes to learn about the relative drift in clock values across nodes, and uses this learning to perform internal corrections, rather than sending out messages. This reduction in message traffic greatly increases the energy efficiency of the node and results in increased effective lifetime for the sensor network. Implementation of **BTSP** for TinyOS, and have evaluated in the context of TPSN running on XSM motes.

7.1 Future Research

BTSP wrapper is designed to work with a variety of symmetric time synchronization protocols with efficiently saving a lot of energy, but there are still other asymmetric protocols that do not fit the wrapper yet. In future **BTSP** Wrapper can be modified to level of suitable to be implemented along with the asymmetric protocols by involving a little of bit adjustments to time synchronization protocols in use.

BIBLIOGRAPHY

[1] Dheeraj Bheemidi and Nigamanth Sridhar. A wrapper based approach to sustained time synchronization in wireless sensor networks. In *ICCCN'08: Proceedings of 17th International Conference on Computer Communications and Networks*, August 2008.

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

[3] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *In the proceeding of SenSys 03*, November 5-7 2003.

[4] M. Marti, B. Kusy, G. Simon, and kos Ldeczi. The flooding time synchronization protocol. In *In the Proceeding of SenSys04*, November 3-5 2004.

[5] W. Su and I. F. Akyildiz. Time-diffusion synchronization protocol. In *In the Proceeding of IEEE/ACM Transactions of networking*, April 2005.

[6] S. yoon, C. veerarittiphan, and M. L. Sichitiu. Tiny-sync: Tight time synchronization for wireless sensor networks. 3(2).

[7] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. In *IPDPS 01: Proceedings of the 15th International Parallel and Distributed Processing Symposium*, page 186. IEEE Computer Society, 2001.

[8] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Fireflyinspired sensor network synchronicity with realistic radio effects. In *SenSys 05: Proceed-*

ings of the 3rd international conference on Embedded networked sensor systems, page 142153. ACM Press, 2005.

[9] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler. Elapsed time on arrival: A simple and versatile primitive for time synchronization services. *International Journal of Ad hoc and Ubiquitous Computing*, 2(1):239251, 2006.

[10] A. swol Hu and S. D. Servetto. Asymptotically optimal time synchronization in dense sensor networks. In *in WSNA 03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 1–10. ACM Press, 2003.

[11] H. Dai and R. Han. Tsync: a lightweight bidirectional time synchronization service for wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8(1):125–139, 2004.

[12] S. Chen, A. Dunkels, F. Osterlind, T. Voigt, and M. Johansson. Time synchronization for predictable and secure data collection in wireless sensor networks. In *The Sixth Annual Mediterranean Ad Hoc Networking WorkShop*, June 12-15 2007.

[13] T. Herman and C. Zhang. Stabilizing clock synchronization for wireless sensor networks.

[14] S. Ping. Delay measurement time synchronization for wireless sensor networks. Intel Research Berkeley Lab, 2003.

[15] F. Otto, D. P. Mirembe, E. Olule, and S. Ouyang. Enhanced master-slave time synchronization architecture for wireless sensor networks. In *In the proceedings of ATNAC*, December 2006.

[16] D. Cox, E. Jovanov, and A. Milenkovic. Time synchronization for zigbee networks. In *In the Proceedings of the 37th SSST*, 2005.

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

[18] TinyOS Wiki. *http : //en.wikipedia.org/wiki/Tinyos*.

[19] TinyOS. *http : //www.tinyos.net*.

[20] Mica2 Datasheet. *http : //www.xbow.com/products/Product _pdf _files/Wireless _pdf/MICA2 _Datasheet.pdf*.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: elements of reusable object-oriented software. *Addison-Wesley Professional*, 1995.

[22] X. Jiang, J. Taneja, J. Ortiz, A. Tavakoli, P. Dutta, J. Jeong, D. Culler, P. Levis, and S. Shenker. An architecture for energy management in wireless sensor networks. *SIGBED Rev.*, 4(3):31–36, 2007.

[23] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler. Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. In *IPSN 05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 70. IEEE Press, 2005.

[24] XSM. *http : //www.cs.berkeley.edu/ prabal/projects/index.html*.