TIME SYNCHRONIZATION IN NETWORK-CENTRIC SENSOR NETWORKS

by

SEJAL DILIP RAJE

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2005

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Qilian Liang, my supervising professor, for providing me the opportunity to work on a topic of such magnitude for my Master's thesis. I am greatly indebted to him for all the insight and support that he showed in every phase of my research. His perseverance, patience, and dedication helped me immensely in maintaining a positive outlook during the highs and lows of my research. I feel extremely fortunate, and very proud, to have done my research under him.

I wish to thank Dr. Saibun Tjuatja and Dr. Enjun Xiao for taking time to serve on my thesis committee. Acknowledgment is also due to Dr. Vasant Prabhu for providing the latex style files.

Special thanks to my friend, Pramod Lakshminarasimha, for providing his guidance and knowledge for my thesis work. Without his help, suggestions, and reviews, it would have been difficult to fine-tune my work to this extent.

No words can describe how grateful I am to my parents, Mr. Dilip Raje & Mrs. Megha Raje, and my brother, Salil, who showed immense faith in me and whose blessings and prayers made this possible. I would also like to thank them for taking care of me in every phase of my studies by providing financial support, which made it possible to pursue my dream of higher studies and kept me focussed on my research. I owe all my success in life to them.

I want to thank all my friends at UTA, for their help and encouragement, for being there during my difficult times. The days I have spent with them at UTA are something I will treasure for a lifetime. I consider myself fortunate to have got such good friends.

Lastly, I would like to express my deepest appreciation to my fiancé, Parind, for giving me the courage, confidence, and moral support, which has a great contribution in helping me reach my goals.

July 15, 2005

ABSTRACT

TIME SYNCHRONIZATION
IN NETWORK-CENTRIC
SENSOR NETWORKS

Publication No. _____

Sejal Dilip Raje, M.S.

The University of Texas at Arlington, 2005

Supervising Professor: Qilian Liang

Time synchronization is a crucial component of infrastructure for wireless sensor networks (WSN). Most applications of WSNs make extensive use of time synchronization, for accurate timestamping of events, coordinate activities of the network, data fusion or TDMA scheduling. The unique requirements of sensor networks in terms of precision, lifetime, energy and scope of the synchronization achieved, make the traditional synchronization methods unsuitable for WSNs. This calls for design of new synchronization methods for WSNs to meet these needs. In this thesis, we describe the design and implementation of three novel techniques for achieving synchronization in network centric scenario of sensor networks: sequential least squares, Kalman filter and fuzzy logic systems. These techniques achieve a highly accurate long-term and adaptive synchronization by forming a timescale between clocks of two nodes in the network, one of which is the sender of timestamps and other is the receiver. A comparison of these approaches is also

presented. Also, a protocol is developed based on these techniques that can achieve a synchronization that is highly energy-efficient, lightweight, multimodal, tunable, scalable and which can provide relative or absolute notion of time.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Advances in wireless networking, micro-fabrication and integration, and embedded microprocessors have enabled a new generation of sensor networks that are envisioned to fulfil complex tasks and are suitable for a range of commercial and military applications. A wireless sensor network (WSN) consists of a large number of small scale devices that are able to cooperate with one another. Each of these sensor nodes is capable of limited computation, wireless communication, sensing and information processing. With this new class of networks come new challenges in many areas of system design and information processing [1].

Time synchronization is an important feature of almost any distributed system. A lot of factors make the clocks of the nodes show different times. 'Time Synchronization' simply means to *get all the nodes in a network to agree on a common time origin*, so as to avoid the confusion and problems caused by different nodes referring to different clocks. Time synchronization in distributed systems is a well-studied problem, and many solutions exist for traditional networks. For e.g., the Network Time Protocol (NTP) has been widely deployed and proven itself to be efficient in the Internet.

Since the nodes in a sensor network operate independently, their clocks may not stay synchronized with one another. In WSN, time synchronization is fundamental to its purpose, and is needed by many applications, such as:

- Data logging and Fusion
- Absolute time of occurrence and temporal delivery of events

- Target Tracking and Localization
- Configuring Beam-forming arrays and time-of-flight measurements
- TDMA radio schedules
- Cryptography

1.2 Requirements of Sensor Network Time synchronization

Here, the obvious question is : *Are traditional time-synchronization methods applicable even in case of sensor networks?* The answer is 'no'. Many assumptions in the traditional schemes do not hold true in the case of WSNs [2]. Some of these factors can be described as follows:

- *Energy Constraint:* Owing to their small size and nature of applications which they are designed for, energy is a major concern in a WSN. Classical synchronization algorithms are based on assumptions like, "CPU is available for use most of the time, listening to network is free, frequent transmissions and re-transmissions are possible". These very factors lead to a lot of energy consumption in a sensor network.
- *Dynamic Topology:* In the Internet, the topology remains more or less static, and this fact is used by NTP for being configured. There are servers available throughout, which serve as a source of external time, and other nodes synchronize with them forming a hierarchy. Also it assumes that nodes are connected before they need synchronization. In a WSN, network dynamics result from various factors like mobility of nodes, node failures, environmental obstructions etc., which prevents simple static configurations. Hierarchical structures might get nodes poorly synchronized, and nodes might not be connected when they need synchronization the most.

- *Variety of applications:* WSN are used for a variety of applications, which can have totally different needs as far as synchronization is concerned. For e.g., localization applications need a short-lived but highly precise synchronization, while target tracking applications can tolerate a little lower precision, but want the synchronization to last longer. Also some applications might need a global timescale while some others can do only with a local timescale. And while a few require absolute or 'real' time as reference, for some others, a relative notion of time is enough.
- *Cost and Form factor:* Sensor nodes are very small in size and also cheap. Now, we can obtain very good synchronization with GPS receivers, but it will be unreasonable to put a $100 GPS receiver on a disposable sensor costing $10.

All the above factors make the problem of time synchronization more challenging in case of sensor networks. Keeping them in mind, we can formulate some design requirements, in general, for sensor network time synchronization:

- **Energy efficient** The energy required to achieve synchronization should be minimum.
- **Multimodal and Tunable** The synchronization provided should be *necessary and sufficient* for the current conditions. This can be possible by combining two methods to get more advantages from a tiered architecture. We want the scheme to have a good accuracy, but at the same time save energy by not providing more accuracy than required. It will be best to have some parameters for tuning the algorithm to our requirements.
- **Adaptive** The synchronization should adapt itself to the needs of application regarding various factors such as scope, lifetime, precision, scalability etc.

- **Robust to Ad-Hoc Networks** Changes in the network topology, node mobility, node failure should not result in a total collapse of the synchronization architecture or even a drastic change in its accuracy.
- **Global *or* Local Timescale** The protocol should be able to build a network-wide timescale or a local timescale.
- **Post-facto *or* Always On** Some applications need the nodes to synchronize only after an event of interest has happened, i.e. *Post-Facto*, and some other applications, like TDMA scheduling, need the nodes to be synchronized all the time.
- **Relative *or* Absolute Time** The synchronization algorithm should have a provision to provide a relative timescale within the network or get all nodes synchronized to absolute time, as required.

1.3 Existing work

Several algorithms have been proposed and researched for time synchronization in sensor networks; but only a few of them have been implemented.

A typical GPS receiver can give a very good precision of 200ns and a wide scope and global timescale. But their size, cost and energy requirements are too large for small sensor nodes. Also GPS architecture is not always available.

Some groups have even tried using NTP for sensor network time synchronization. But the above mentioned factors make it unsuitable for this application.

The reference broadcast synchronization (RBS) designed by [3] is an important scheme. It achieves a R-R pairwise synchronization to remove sender's nondeterminism and results in a good precision of a few microseconds. It also provides frequency estimates between two receivers using a linear regression technique. But the message overhead is very large. This approach is extended in [4] to provide a global timescale. In [5] the

authors propose a protocol along with experiments to achieve a post-facto synchronization, which provides high-precision, but it is short-lived and localized. They combine the estimates obtained from their algorithm with NTP to improve performance and obtain absolute time.

Another approach suggested in [6] offers a lightweight, bi-directional and multi-modal service, having a push & pull approach. It also uses reference broadcasts and provides a good accuracy.

A probabilistic method of clock synchronization is described in [7]. The authors extend a deterministic protocol like RBS to provide bounds on the accuracy of clock synchronization, and provide protocol parameters to make it adaptive by making provision for tradeoff between accuracy and protocol parameters.

[8] suggests a mathematical approach which leads to a protocol that provides optimal synchronization in a very high density of nodes. They derive an optimal estimator for determining the state of an ideal node's clock and thus provide a global synchronization in which all the nodes are synchronized optimally with the sender.

Another method for achieving a network-wide synchronization is suggested by [9]. This approach first establishes a hierarchical structure and then performs a pairwise synchronization along it's edges, resulting in a good accuracy which does not degrade over multihop. The algorithm suggested by [10] also achieves a lightweight, multihop synchronization in a similar tree-based scenario. Another lightweight, energy efficient synch protocol is presented by [11] which provides a good precision and works well for both single hop and multi hop.

In situations where temporal ordering of events matter more than absolute time of occurrence, the scheme suggested by [12], for sparse sensor networks, provides a modest synchronization. It gives mapping of intervals between events from clock readings made on one node to the clocks of other nodes, without trying to synchronize with a common

time-base. Though this scheme has very little overhead, it provides a localized and instantaneous synchronization with only 1 ms accuracy. An improved version of this interval method is proposed by [13] which provides the worst and best case of achievable time uncertainty in an ad-hoc sensor networks scenario.

An interesting method suggested by [14] presents an adaptive protocol, which along with providing a good accuracy, adapts itself to the environmental conditions and user-defined precision by adjusting the sampling rate, thus avoiding any wastage of energy.

It can be observed that these time synchronization approaches suggested for sensor networks fall into many disparate points in the parameter space described in previous section. Each of these schemes have trade-offs and no single method is yet optimal on all axes.

1.4 Objectives and Overview of this Thesis

From the literature survey performed by us, it was observed that most of the schemes are suffering from the drawback in that, they are only able to achieve a short-lived, instantaneous synchronization. It therefore applies only to a set of applications and a lot of energy will be spent if nodes need to be synchronized often. Also, if the nodes are subjected to severe changes in environmental conditions, then the accuracy of these short-term synchronization schemes might suffer a lot.

The primary objective of the thesis is to propose a way of achieving long-term time synchronization in sensor networks, which remains a somewhat unaddressed problem in current methods. It is also required that the synchronization algorithm is adaptive to the environmental effects on the clocks of nodes. We present three methods for obtaining this. We also present a comparison to show the positive and negative points of each method, so the user has a choice to select the method best fitted for a specific application. Two of these methods also provide an estimation of the clock parameters. We also suggest a

protocol which is built around these methods. This protocol is able to achieve a long-term time synchronization that has the following characteristics: *provides high precision, adaptive to environmental effects, energy-efficient, scalable, tunable and multi-modal.*

The remainder of the thesis is organized as follows: Chapter 2 introduces the problem and presents the expression governing sensor nodes' clock behavior over long term. It also presents a synchronization protocol to achieve desired goals. Chapter 3 presents a sequential least squares approach for the problem. Chapter 4 presents Kalman filter design for the same, and discusses the use of an already tested Kalman filter approach in the current situation. Chapter 5 presents a novel fuzzy logic approach for time synchronization problem. Chapter 6 present the results from computer simulations and also a comparison of the three methods. Finally, Chapter 7 draws the conclusions from this thesis and suggests some future work.

CHAPTER 2

PROBLEM FORMATION and PROTOCOL DESCRIPTION

This chapter presents the case for long-term synchronization in sensor networks. Here, we discuss the scenario and assumptions behind it and also derive an expression, which is later used for data simulation. It also describes the goals and design of our synchronization protocol.

2.1 Synchronizing a pair of sensor nodes

2.1.1 Sources of Time Synchronization Error

The first step in designing any time-synchronization algorithm would be to understand why and where it is required. The different factors which give rise to errors in clocks of nodes or in synchronization algorithm can be divided into two main categories:

1. *Oscillator Characteristics:* The sensor nodes' clocks run on very cheap oscillators. The following two characteristics are the main sources of errors between the clocks of two different nodes.
    - *Accuracy:* This is a measure of difference between oscillator's expected (ideal) frequency and actual frequency. It is also called as resolution of the clock; it's maximum is specified by the manufacturer.
    - *Stability:* This is oscillator's tendency to stay at the same frequency over time. There can be short-term instability due to environmental effects, supply voltage etc.; or long-term instability due to temperature effects, oscillator aging etc.

2. *System and Network issues:* The non-determinism in the message delivery latency is a major source of error in any synchronization algorithm, when applied into real sensor networks. This can be categorized in four type of delays:
    - *Send Time:* The time spent at the Sender of the timestamps to build the message, i.e. the time duration between generating the timestamp and injecting it into the network.
    - *Access Time:* Delay occurred while waiting for access to the transmit channel.
    - *Propagation Time:* Time required for the message to travel from sender to receiver.
    - *Receive Time:* Time needed for processing at the receiver's network interface.

    All the above factors result in following errors between the clocks of 2 nodes:

Time or Phase offset: The oscillators of any two nodes can be out of phase at any given time, resulting into different time on both clocks. There can be some initial time-offset between nodes at the start of a synchronization procedure.

Clock Bias or skew: This is the frequency error between the two clocks, since each of them is running at a different frequency.

Clock drift: It is not just that the clocks are running at different rates, but even the frequency of each clock does not stay constant over a period of time! Clock drift arises from the instability of oscillators, because clocks drift from their initial frequency. If each clock drifts at a rate of R msec/sec, then maximum relative drift between two clocks can be $2R$ msec/sec.

Even if two clocks are assumed to have the same frequency and no drift, the above mentioned delays in the network result in phase offset between two clocks. This is because when the timestamp from the sender reaches a receiver, the receiver adjusts it's clock according to the received timestamp; but the sender's clock changes in the time required for the timestamp to reach the receiver, due to network delays.

For a network of nodes, these sources give rise to errors in accuracy from an ideal clocks, and dispersion amongst their clocks.

2.1.2 Formation of a time-conversion scale

As mentioned in the earlier chapter, the solution to the time synchronization problem in networks, is the exchange of 'timestamps' between the nodes that wish to have same reading on their clocks. These timestamps can be called *synchronization pulses*. In many of sensor networks applications, the main goal is to achieve a good precision among nodes. Our interests in synchronization arises from one well-tested approach of synchronizing sensor nodes by forming a *time conversion scale*. This sort of scale will tell us how to convert a time in the clock of one node into a time in the clock of some other node. If such a scale is available with a node, then it will have an idea of other node's clock; hence it can said to be synchronized with the other node. This can be viewed as building a *logical* or *virtual* clock on top of the node's atomic clock! But question arises that 'how could such a timescale be more useful than simply sending the timestamps from one node to other and thus synchronizing them?'

Well, the essence of building such a timescale is in predicting other node's clock before the timestamps actually arrive! If we have such a conversion scale available with us ahead of time, and also if it is accurate, that gives a lot of benefits - as shown by previous work. This will keep a good precision among nodes for intervals between the timestamps. It is possible to achieve post-facto synchronization with such a scale. It is even possible to obtain global synchronization using a similar scale. But of course, there are a number of problems to overcome in order to form such a timescale.

Like many other synchronization algorithms, formation of a timescale can be done on either a *Sender-Receiver* (S-R) basis, where a conversion scale is developed between

the sender of the timestamps and one receiver, or on the *Receiver-Receiver* (R-R) basis, where receivers of timestamps have conversion scales available between themselves.

Now, similar approach have been tried before in case of both S-R and R-R synchronization problems. These works use regression or estimation technique and get a clock model or a best-fit line between the clocks of 2 or more nodes. This can give a good estimate of the instantaneous phase offset and the frequency bias between the clocks; and is successful in forming a conversion scale over a short period of time. But the problem with these techniques is that they do not take into account the *frequency drift* or *environmental effects*. These factors can affect the accuracy of synchronization especially over a longer period of time. Hence the clock models or the best-fit line given by these approaches gives poor performance if extended over a long period of time.

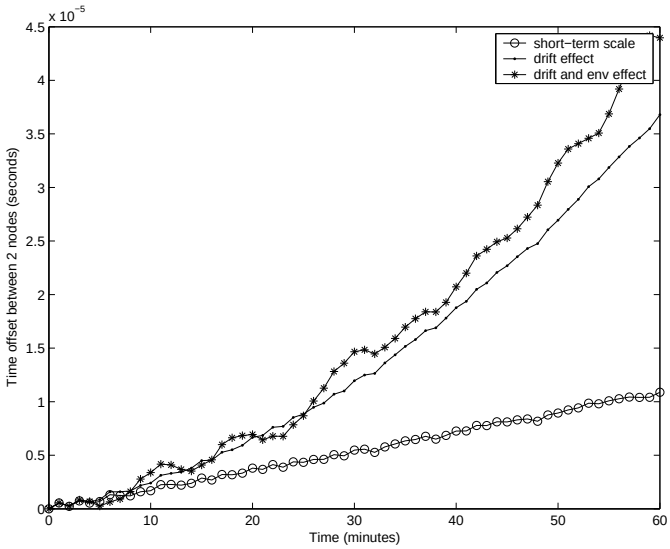

Figure 2.1. Evolution of clock offset over time and extended short-term time scale.

We believe that it is necessary to design a synchronization scheme which is valid over long term. It implies that we need the nodes to remain well synchronized even when the time period between consecutive timestamps is more than a few seconds. It is

also desired that the algorithm can keep the nodes synchronized even in the presence of changes in clock frequency due to environmental conditions. In order to attain this, we set our goals on designing a synchronization scheme which takes into account the frequency drift or aging rate and environmental effects along with the phase offset and frequency bias between the clocks of two clocks. In this paper, we have adopted the estimation theory as well as the fuzzy logic approach for achieving such a synchronization.

2.2 An equation of clock drift

We will now present an expression for the clock drift or phase error between two clocks. If we subtract the current time in one clock from the current time in other clock, we get the time-offset or error between them, at the current point of time. So predicting the value of this time-offset will be same as predicting the time in the clock of other node.

From the definition of frequency:

$$f = d\Phi/dt$$

and integrating both sides over time,

$$\Phi = \int f(t)dt \tag{2.1}$$

where

f = frequency

Φ = phase

t = time

From [15], for an oscillator,

$$f = f_{nom} + \Delta f_0 + a.(t - t_0) + \Delta f_n(t) + \Delta f_e(t) \tag{2.2}$$

$$\Phi = \Phi_{nom} + \Delta\Phi \tag{2.3}$$

where

t_0 = starting time

Φ_{nom} = nominal, time varying phase

$\Delta\Phi$ = phase, or time error

f_{nom} = nominal frequency

Δf_0 = initial frequency error

a = aging rate

Δf_n = short-term frequency instability (noise) term

Δf_e = environmental term

The above equation provides a good approximation for all common oscillators. The nominal frequency can also be identified as the ideal frequency at which the oscillator is supposed to run. Combining the equations (2.1), (2.2) and (2.3) and integrating from the starting time to the current time t, we get,

$$\Phi_{nom}(t)+\Delta\Phi(t)-[\Phi_{nom}(t_0)+\Delta\Phi(t_o)] = \int_{t_0}^{t}[f_{nom}+\Delta f_0+a.(\tau-t_0)+\Delta f_n.(\tau)+\Delta f_e.(\tau)]d\tau \tag{2.4}$$

The short term frequency variation has zero mean and does not lead to accumulated time errors. Also the amplitude of these short term variations (clock jitter) is small enough that they do not cause the clock to accelerate or decelerate erratically. But the environmental term, primarily due to temperature, can be significant.

Then, the resulting equation for clock drift of a single clock is

$$\Delta\Phi(t) = \Delta\Phi(t_0) + \Delta f_0.(t-t_0) + \frac{a}{2}.(t-t_0)^2 + \Delta f_n(t) + \int_{t_0}^{t}\Delta f_e(\tau) \tag{2.5}$$

To get the expression in term of time-offset, consider

$$f_{avg} = \Phi/t \text{ and } t = \Phi/f$$

Hence, the time offset of a single clock i from a perfect or standard time source - which does not have any frequency bias, aging or randomness and any initial phase offset or is affected by environmental term - is given as,

$$\Delta t_i(t) = \Delta t_i(t_0) + \frac{\Delta f_0}{f_{nom}}.(t - t_0) + \frac{a}{2}.(t - t_0)^2 + \frac{\Delta f_n(t)}{f_{nom}} + \frac{\int_{t_0}^{t} \Delta f_e(\tau)}{f_{nom}} \tag{2.6}$$

From this equation, we have derived the expression for clock drift or time offset between clocks of two nodes - a sender and a receiver. Assume that these two nodes' clocks are of the same type and therefore, under ideal conditions, they should run at the same frequency, say f_{nom}, without any initial phase error. But this is not possible in reality, and so each of the clocks will drift from its ideal conditions.

Let the drift of the clock of sender node from the ideal conditions, at time t, be

$$\Delta t_S(t) = \Delta t_S(t_0) + \frac{\Delta f_{0S}}{f_{nom}}.(t - t_0) + \frac{a_S}{2}.(t - t_0)^2 + \frac{\Delta f_{nS}(t)}{f_{nom}} + \frac{\int_{t_0}^{t} \Delta f_{eS}(\tau)}{f_{nom}} \tag{2.7}$$

and the offset between ideal clock and clock of receiver node at time t be,

$$\Delta t_R(t) = \Delta t_R(t_0) + \frac{\Delta f_{0R}}{f_{nom}}.(t - t_0) + \frac{a_R}{2}.(t - t_0)^2 + \frac{\Delta f_{nR}(t)}{f_{nom}} + \frac{\int_{t_0}^{t} \Delta f_{eR}(\tau)}{f_{nom}} \tag{2.8}$$

So, the relative drift or the time difference directly between clock of S and clock of R at time t will be given as,

$$\begin{aligned} \Delta t_S(t) - \Delta t_R(t) \;=\; & \Delta t_S(t_0) - \Delta t_R(t_0) + (\frac{\Delta f_{0S}}{f_{nom}} - \frac{\Delta f_{0R}}{f_{nom}})(t - t_0) + (\frac{a_S}{2} - \frac{a_R}{2})(t - t_0)^2 \\ & + \frac{\Delta f_n(t)}{f_{nom}} + \frac{\int_{t_0}^{t} \Delta f_e(\tau)}{f_{nom}} \end{aligned} \tag{2.9}$$

$$\Delta t_{SR}(t) = \Delta t_{SR}(t_0) + \frac{\Delta f_{0SR}}{f_{nom}}(t - t_0) + \frac{a_{SR}}{2}(t - t_0)^2 + \frac{\Delta f_n(t)}{f_{nom}} + \frac{\int_{t_0}^{t} \Delta f_e(\tau)}{f_{nom}} \tag{2.10}$$

To verify this result, we have tried using another approach, suggested by [8], to derive the same expression.

From (ref), the instantaneous frequency of the oscillator of node i can be medelled as,

$$f_i(t) = f_o + \Delta f + f_d(t - t_o) + f_r(t) \tag{2.11}$$

where

t_o is the initial time reference

f_o is the ideal frequency

Δf is the frequency bias or offset

f_d is the frequency drift

$f_r(t)$ is a random process for the unmodelled random

From this, a timing relationship is established by the authors. Since a clock also consists of a counter which counts to the next integer every complete cycle of the oscillator, we get,

$$t_i(t) - t_i(t_o) = \frac{1}{f_o} \int_{t_o}^{t} f_i(t)dt \tag{2.12}$$

where $t_i(t)$ is the time in clock i at ideal time t.

Then, from equation (2.1), we can write,

$$t_i(t) - t_i(t_o) = \Phi_i / f_o \tag{2.13}$$

Similarly, for clock of another node j,

$$t_j(t) - t_j(t_o) = \Phi_j / f_o \tag{2.14}$$

Hence the time error between the clocks of these 2 nodes at time t is,

$$t_i(t) - t_j(t) = |(\Phi_i - \Phi_j)| / f_o \tag{2.15}$$

If we extend the term $|(\Phi_i - \Phi_j)|$ using the equation (2.5), we get back the same equation as (2.6). Or, Combining the equations (2.11) and (2.12), we get the expression for the time of clock i at ideal time t,

$$t_i(t) = t_i(t_o) + (t - t_o) + \frac{\Delta f}{f_o}(t - t_o) + \frac{f_d}{2f_o}(t - t_o)^2 + \Psi_i(t) \tag{2.16}$$

where

$$\Psi_i(t) = 1/f_o \int_{t_o}^{t} f_r(t)dt \tag{2.17}$$

Rearranging terms, the error between clock i and the ideal clock at time t at time t is obtained as,

$$t_i(t) - t = (t_i(t_o) - t_o) + \frac{\Delta f_i}{f_o}(t - t_o) + \frac{f_{di}}{2f_o}(t - t_o)^2 + \Psi_i(t) \tag{2.18}$$

If we add the environmental term to the above equation, then we arrive at the equation (2.6)! Then, we can derive the expression for the error between clock of S and clock of R at time t , in the same way.

So, by different ways, we have arrived at the same expression for the time offset between two nodes. We can modify the above equation to give us the offset between the receiver's clock and the sender's clock, at time t_R in the receiver's clock, as,

$$\Delta t_{SR}(t_R) = (t_{0S} - t_{0R}) + \frac{\Delta f_{0SR}}{f_{nom}}(t_R - t_{0R}) + \frac{f_{d_{SR}}}{2f_{nom}}(t_R - t_{0R})^2 + \Psi_n(t_R) + \Delta t_{eSR}(t_R) \tag{2.19}$$

and t_{R0} can be assumed to be 0.

Thus, the phase or time offset between two clocks at any given time results from a combination of initial phase offset, frequency bias, frequency drift and the environmental terms. As more time passes from the synchronization point, the drift and environmental terms become more significant.

This resulting expression is used as a model for designing the solutions, as well as for data simulations.

2.3 A Synchronization protocol

2.3.1 Sensor Network Scenario

We aim at achieving our synchronization goals for the case of Sender-Receiver synchronization. Consider a network consisting of many small clusters of nodes. Each of theses clusters have many nodes ranging from about 5 to even 100. Without any loss of generality, and as found in many applications, we assume the presence of a clusterhead in each cluster which has the task of managing data and monitoring activities of the cluster. All the other nodes in any cluster are assumed to be in the broadcast range of the respective clusterhead, i.e. they are at a *single hop* distance away from the clusterhead.

We will refer to the clusterhead as the *Sender* node; since it is the one which sends timestamps to all other nodes in its cluster. The timestamps mainly include the current time in the sender's clock, and are also called as *synchronization pulses* or *synchronization beacons*. These timestamps need to be sent at regular intervals. Thus the synchronization beacons have a fixed sampling period. It is also referred as *re-synchronization instances*. In this scenario, our goals is to *get all the nodes in a cluster to synchronize with the sender*. Our idea of achieving this is by forming a relative timescale or scale between the sender and each receiver, according to the clock of that receiver. By using the method proposed by us, such a timescale should yield a long-term synchronization. If each receiver in a cluster has such a scale available with them, then a network-centric time synchronization is achieved, wherein all nodes are synchronized to a common time-base.

We have made following assumptions regarding the network issues:

- *Uniform transmission delay:* The sender node makes use of the physical broadcast channel to transmit pulses to all the receivers in the cluster. Hence the send and access time are same for all the nodes, and thus all the receiver nodes are synchronized with the sender within the same accuracy. Also, we assume that the

synch pulses are transmitted at the exact time they are generated. This is a fair assumption since the synch pulses are simple pulses and do not have any message overhead.

- *No propagation delay:* In case of sensor networks, the distances between nodes are very small. Hence the propagation time needed for the synchronization pulses, which are radio waves, to travel such small transmission distance is very small. For our scenario, since all the nodes in a cluster which need synchronization are within the broadcast region of the sender, the propagation delay will be negligible and will not add to a significant error in the synchronization.
- *No receiver delay:* We assume that the receiver has the clock reading at the instance it receives the synchronization pulse. This can be achieved by using some kernel timestamps and some hardware or system modifications, as shown by (ref - RBS). The remaining error in the receiver can be absorbed in the random clock jitter.

This takes care of most of the errors arising due to the system and network issues. Once we have achieved an adaptive, long-term synchronization using our methods, any compensation for other errors due to the above factors can be added later as a 'delay correction'.

With this protocol and the proposed algorithms, we will be able to achieve the following goals of synchronization:

- *Continuous or Post-facto (Multimodal):*

  This protocol is able to provide both *'Always On'* or a *'Post-Facto'* synchronization - as per the needs of application. For Always On type of synchronization, the sender needs to send the synch pulses periodically. The Post-facto synchronization can be either sensor-initiated or clusterhead-initiated. In any case, the algorithms will need only a few pulses to be transmitted between S and R to form the timescale.
- *Long Term:*

The network stays synchronized for a very long period of time - i.e. even when the sampling period of synch beacons is several minutes. This is exactly opposite of instantaneous synchronization, where nodes go out of synch after a few seconds. Further, in indoor applications where the environmental effects and other factors responsible for high and random variation in clock drift are absent, adjusting the clocks of nodes from the available estimates of clock parameters can keep them synchronized over few hours.

- *High Accuracy:*

  The algorithm should provide a very good precision, which can vary from 1 msec to a few microseconds as per the needs of application. For our case, this is calculated from the error between the predicted value of the time-offset and the actual measurement of time-offset, between the clocks of sender and receiver.

- *Adaptive to environmental effects:*

  Especially in outdoor scenarios, the environmental condition and the resulting clock drift will affect the synchronization accuracy a lot. The synchronization protocol will account for such changes and adapt itself to environmental conditions to keep nodes synchronized with as much more precision and less energy as possible.

- *Energy-efficient:*

  Energy consumption as well as energy wastage should be very less while still achieving the required accuracy. In case of a long-term synchronization, nodes do not require to receive timestamps every few seconds. hence in the interval between consecutive timestamps, nodes can turn off their radios. Since for a sensor node, the maximum energy is spent in transmission and reception, all the nodes in the cluster will save a lot of energy because they can sleep in the periods between synchronization pulses. Also, having a S-R synchronization rather than a R-R synchronization will reduce overhead and also avoid collisions in the network. This

will result in very less energy consumption for both individual nodes and the whole cluster, making the synchronization energy efficient.

- *Scalable:*

  All the nodes in the cluster must be synchronized to a common timescale, i.e. the clock of the sender, irrespective of the number of nodes in the cluster. The accuracy and characteristics of the synchronization algorithm should remain unaffected when the clusters are highly populated or become sparse due to dying of nodes.

- *Tunable:*

  We will provide some protocol parameters, so that changing them could tailor the accuracy of this scheme as per the needs of application. Thus this 'necessary and sufficient' will be precise and energy-efficient at the same time.

- *Lightweight Protocol:*

  The simple nature of the synchronization pulses, and the S-R type of synchronization, which requires broadcasting of synch pulse from one node to many, reduces the overhead significantly, for each node as well as for a cluster, making this a very lightweight protocol.

- *Relative or External Timescale:*

  Synchronizing all the receivers in a cluster to the clock of the sender will provide a relative timescale for the network-centric case. If the sender of synch pulses, which is assumed to be the clusterhead, has an external timescale like UTC or GPS receiver available with it, then we can obtain the absolute time too.

Although we have not aimed at achieving a global or multi-hop synchronization for this report, the proposed methods can be easily extended to achieve the same by using some well-tested techniques. Also, in case of dynamic topology, a change in clusterhead, i.e. the sender node, will not affect the synchronization drastically. It will only need a few initial pulses to build a new timescale with respect to the newly elected sender.

CHAPTER 3

MINIMUM VARIANCE SEQUENTIAL LEAST SQUARES SOLUTION

This chapter presents our first solution to achieve long term synchronization. It describes the 'Minimum Variance-Sequential Least Squares (MV-SLS)' method and explains how it can be used in this time synchronization problem. Simple least squares solutions have been applied in formation of short-term time scales. As an extension to that approach, and recognizing the usefulness of sequential least squares in calculating the parameters of a system in real-time, we have decided to test this method for our synchronization problem.

3.1 Description of MV-SLS method

3.1.1 Linear batch Estimation

The least squares approximation is used in a wide variety of applications such as: curve fitting of a data, parameter identification and system model realization [16]. For any variable or parameter in estimation, there are three quantities of interest; the true value (x), the measured value ($\tilde{x}$) and the estimated value ($\hat{x}$). The measurements are usually modelled using a function of the true value plus some error. Estimated values are determined from the estimation process itself and are found using a combination of a static/dynamic model and the measurements. Therefore,

$$\tilde{x} = x + v$$

and,

$$\tilde{x} = \hat{x} + e$$

where, v = measurement error and e = residual error.

A particulary challenging facet of practical estimation application is correctly specifying the system's mathematical model. Consider that we have a batch of measured values, $\tilde{y}_j$, of a process $y(t)$, taken at known discrete instants of time t_j,

$$\tilde{y}_1, t_1; \tilde{y}_2, t_2; ....; \tilde{y}_m, t_m \tag{3.1}$$

and a proposed mathematical model of the form

$$y(t) = \sum_{i=1}^{n} x_i h_i(t), m \geq n \tag{3.2}$$

where

$$h_i(t) \in \{h_1(t), h_2(t), ..., h_n(t)\} \tag{3.3}$$

are a set of independent *basis* functions. The x_i are a set of constants whose numerical values are unknown. What we seek is a set of estimates, given by $\hat{x}_1, \hat{x}_2, ..., \hat{x}_n$, which can be used in equation(3.2) to predict $y(t)$. But errors arise in this predicted (estimated) value due to measurement errors, incorrect choice of x-values and modelling errors. So, we have,

$$\tilde{y}_j \equiv \tilde{y}(t_j) = y(t_j) + v_j = \sum_{i=1}^{n} x_i h_i(t_j) + v_j, j = 1, 2, ..., m \tag{3.4}$$

$$\tilde{y}_j = \hat{y}(t_j) + e_j = \sum_{i=1}^{n} \hat{x}_i h_i(t_j) + e_j, j = 1, 2, ..., m \tag{3.5}$$

so,

$$e_j \equiv \tilde{y}_j - \hat{y}_j \tag{3.6}$$

The above equations in the compact matrix form are given by,

$$\tilde{y} = H\hat{x} + e \tag{3.7}$$

$$\tilde{y} = Hx + v \tag{3.8}$$

$$\hat{y} = H\hat{x} \tag{3.9}$$

where

$\tilde{y} = [\tilde{y}_1\tilde{y}_2...\tilde{y}_m]^T$ = measured y values

$e = [e_1e_2...e_m]^T$ = residual errors

$\hat{x} = [\hat{x}_1\hat{x}_2...\hat{x}_n]^T$ = estimated x values

$x = [x_1x_2...x_n]^T$ = true x values

$v = [v_1v_2...v_m]^T$ = measurement errors

$$\mathrm{H} = \begin{pmatrix} h_1(t_1) & h_2(t_1) & . & . & h_n(t_1) \\ h_1(t_2) & h_2(t_2) & . & . & h_n(t_2) \\ . & . & . & . & . \\ . & . & . & . & . \\ h_1(t_m) & h_2(t_m) & . & . & h_n(t_m) \end{pmatrix}$$

$\hat{y} = [\hat{y}_1\hat{y}_2...\hat{y}_m]^T$ = estimated y values

According to the Gauss's principle of least squares, a particular value is selected as an optimum choice for the unknown parameters, such that, that particular $\hat{x}$ minimizes the sum square of the residual errors, given by,

$$J = \frac{1}{2}e^T e \tag{3.10}$$

Then, the explicit solution for such an optimal estimate is given as:

$$\hat{x} = (H^T H)^{-1} H^T \tilde{y} \tag{3.11}$$

Now, if we want to place different weights for different measurements, owing to the relative accuracy of measurements, then the solution becomes,

$$\hat{x} = (H^T W H)^{-1} H^T W \tilde{y} \tag{3.12}$$

where W is the weight matrix.

3.1.2 Sequential Least Squares Estimation

In many of the real world applications, the measurements become available sequentially in subsets rather than as a batch. In such cases, it is desirable to calculate new estimates for the parameters based upon all the previous measurements as well as the currently received subset of measurements. Before going directly to the generalization, let us consider a simple case of two measurement subsets.

$$\tilde{y}_1 = [\tilde{y}_{11}\tilde{y}_{12}...\tilde{y}_{1m_1}] = m_1 \times 1 \text{ vector of measurements}$$

$$\tilde{y}_2 = [\tilde{y}_{21}\tilde{y}_{22}...\tilde{y}_{2m_2}] = m_2 \times 1 \text{ vector of measurements}$$

$$\tilde{y}_1 = H_1 x + v_1$$

$$\tilde{y}_2 = H_2 x + v_2$$

where

$$H_1 = m_1 \times n \text{ known coefficient matrix, } n \leq m_1$$

$$H_2 = m_2 \times n \text{ known coefficient matrix}$$

$$v_1, v_2 = \text{vectors of measurement errors}$$

$$x = n \times 1 \text{ vector of unknown parameters}$$

Thus, the least squares estimate of the unknown parameters, $\hat{x}$, based on the *first* measurement subset is,

$$\hat{x}_1 = (H_1^T W_1 H_1)^{-1} H_1^T W_1 \tilde{y}_1$$

and $\hat{x}$, based on both the measurement sets is,

$$\hat{x}_2 = (H^T W H)^{-1} H^T W \tilde{y}$$

where W is the merged weight matrix, having a block diagonal structure such as,

$$\mathrm{W} = \begin{pmatrix} W1 & : & 0 \\ .. & & .. \\ 0 & : & W2 \end{pmatrix}$$

and $\tilde{y}$ is the merged measurement set,

$$\tilde{y} = Hx + v$$

But the essence of the sequential approach is not in forming merged solution sets; it should rather make use of previous estimates to provide calculation required for the new estimates. Therefore, a better way leads to the solution,

$$\hat{x}_2 = \hat{x}_1 + K_2(\tilde{y}_2 - H_2\hat{x}_1) \tag{3.13}$$

where

$$K_2 \equiv P_2 H_2^T W_2 \tag{3.14}$$

This is the mechanism to sequentially provide an updated estimate $\hat{x}_2$, based on the previous estimate $\hat{x}_1$, and the associated side calculations. Generalizing this, we can obtain the $(k+1)^{th}$ estimate using the k^{th} estimate and $(k+1)$ measurement subsets as,

$$\hat{x}_{k+1} = \hat{x}_k + K_{k+1}(\tilde{y}_{k+1} - H_{k+1}\hat{x}_k) \tag{3.15}$$

where

$$K_{k+1} = P_k H_{k+1}^T [H_{k+1} P_k H_{k+1}^T + W_{k+1}]^{-1} \tag{3.16}$$

$$P_{k+1} = [I - K_{k+1} H_{k+1}] P_k \tag{3.17}$$

This is called the covariance recursion form. The equation (3.15) modifies the previous best correction (estimate) $\hat{x}_k$ by an additional correction to account for the information contained in the $(k+1)$ measurement subset and thus obtains an improved estimate $\hat{x}_{k+1}$. This equation is called Kalman update equation and the correction term

is called Kalman gain matrix. All the calculations shown here are for the linear sequential least squares; but it can be extended to a nonlinear case too.

3.1.3 Minimum Variance SLS

Minimum variance estimation gives us the 'best way (in a probabilistic sense)' to find the optimal estimates. Consider a linear observation model,

$$\tilde{y} = Hx + v$$

We want to estimate x as a linear combination of the measurements $\tilde{y}$ as,

$$\hat{x} = M\tilde{y} + n$$

Now, the minimum variance definition for optimum M and n is that the variance of all n estimates $\hat{x}_i$ from their respective true values is minimized. Therefore,

$$J_i = \frac{1}{2}E\{(\hat{x}_i - x_i)^2\}, i = 1, 2, ..., n \tag{3.18}$$

Solving for M and n we get,

$$\hat{x}_i = M_i\tilde{y}, i = 1, 2, ..., n \tag{3.19}$$

so the equation (3.18) can be rewritten as,

$$J_i = \frac{1}{2}E\{M_i(vv^T)M_i^T\}, i = 1, 2, .., n \tag{3.20}$$

Assuming that v has a zero mean and the covariance matrix as

$$cov\{v\} \equiv R = E\{vv^T\}$$

the equation (3.20)reduces to,

$$J_i = \frac{1}{2}M_iRM_i^T \tag{3.21}$$

Then, from the necessary and sufficient conditions, we obtain,

$$M = (H^T R^{-1} H)^{-1} H^T R^{-1} \tag{3.22}$$

which leads to the optimal solution,

$$\hat{x} = (H^T R^{-1} H)^{-1} H^T R^{-1} \tilde{y} \tag{3.23}$$

Thus, we see that, the minimum variance estimator is identical to the least squares estimator, provided that the weight matrix is the inverse of the observation error covariance. So, to obtain the sequential minimum variance least squares estimator, we simply have to use R^{-1} in place of W (assuming R^{-1} still has the block diagonal structure), in the sequential least squares formulation.

This technique also gives an unbiased estimate of x. The error covariance matrix is given by,

$$P = E\{(\hat{x} - x)(\hat{x} - x)^T\} \tag{3.24}$$

3.1.4 *A Priori* Conditions in SLS and MV-SLS

The sequential process can be started at any step by an *a priori* estimate $\hat{x}_1$ and the covariance matrix P_1. If the *a priori* estimates are not available, then the first data subset can be used for initialization by using a batch least squares to determine $\hat{x}_q$ and P_q, where $q \geq n$. Then the sequential least squares algorithm can be started for $k \geq q$. Another is to start sequential process for $k = 1, 2, ...q - 1$ by using

$$P_1 = [\tfrac{1}{\alpha^2} I + H_1^T W_1 H_1]^{-1}$$

$$\hat{x}_1 = P_1[\tfrac{1}{\alpha}\beta + H_1^T W_1 \tilde{y}_1]$$

where α is a very large number and β is a vector of very small numbers.

For the minimum variance case, it is assumed that the *a priori* estimates are given as the sum of the true state x and the errors in the a priori estimates w, so that

$$\hat{x}_a = x + w$$

and the errors have zero mean and an *a priori* covariance matrix (assumed known),

$$cov\{w\} \equiv Q = E\{ww^T\}$$

It is also assumed that the measurement errors and the *a priori* errors are uncorrelated, so $E\{wv^T\} = 0$.

So for the minimum variance sequential least squares with *a priori* conditions, we have,

$\hat{x}_1 = \hat{x}_a$

$P_1 = Q$

and process the subsequent measurement subsets $\{\tilde{y}_k, H_k, W_k\}$ with $W_k = R^{-1}$ and using sequential least squares results.

3.1.5 Cramér-Rao and 3σ Bounds

The Cramér-Rao inequality gives a lower bound on the expected errors, between the estimated quantities and the true values, from the known statistical properties of the measurement errors. The Cramér-Rao inequality for an unbiased estimate $\hat{x}$ is given by,

$P \equiv E\{(\hat{x} - x)(\hat{x} - x)^T\} \geq F^{-1}$, where $F = [E\{[\frac{\partial}{\partial x} ln'(\tilde{y}; x)][\frac{\partial}{\partial x} ln'(\tilde{y}; x)]^T\}]^{-1}$

For the case of minimum variance estimation, the Cramér-Rao bound modifies to,

$$P \geq (H^T R^{-1} H)^{-1}$$

Now, the estimate error covariance matrix for minimum variance case is,

$$P = (H^T R^{-1} H)^{-1}$$

which satisfies the inequality, proving that these are most efficient estimates.

This covariance equation can be used to give us boundaries on the expected errors. Taking square root of the diagonal elements of P and multiplying them by 3 (and this is calculated without any measurement information , since it depends only on H and R), gives us the 3σ boundaries, which have a significance in probability theory. Thus we can

use probabilistic concepts to compute estimate error boundaries and analyze the expected performance in a dynamical system. It is also useful to calculate errors introduced by using an approximate solution instead of the optimal approach.

3.2 The Design

3.2.1 Estimation problem:

As described in the previous chapter, in our synchronization scenario, the receiver node receives *timestamps* from the sender node at regular intervals. These intervals are also referred to as *sampling period of synchronization beacons* or *re-synchronization instances*. Using these timestamps received at regular intervals of time, we wish to build a *time-conversion scale* between the clock of the sender (S) and the receiver (R). This scale will tell the receiver, the time in the sender's clock at any point of time in the receiver's own clock. Thus, by having such a reference scale, the receiver is supposed to be synchronized with the sender, because it knows the sender's clock at any time, within some accuracy. So this is like having a virtual or logical clock available with the receiver.

Now, subtracting the value of timestamps (which indicates the current time in the sender's clock) from the current time of the receiver's clock, we get the time-offset between S and R according to R's clock. These are the measurements for our time-scale formation problem. So, every time a timestamp is received from the sender, the receiver has a new measurement of this 'time-offset' available with it.

For this first basic design of MV-SLS, we only consider the effect of *clock drift* on the time-scale formation; however, we do not involve the *environmental terms*. Since drift can play a major part for the case of long-term synchronization, it is important to develop a technique which can compensate for it.

Now, the estimator needs to estimate the clock parameters between S and R, and predict the time-offset between them at the next time step, depending on these estimated parameters. As a new timestamp and thus a new measurement is available, we calculate the residual error between the *predicted (estimated) value* of the time-offset at that instance (from our estimator) and, the *measured value* of time-offset (from the current timestamp).

If the residual error, which is the synchronization error of our estimator is very small, *especially for a high sampling period*, it means that even though the timestamp was received after a long period of time, the receiver clock had a good knowledge of the sender's clock. Hence, we can say that our algorithm has achieved long-term synchronization! If the time-offset is calculated for all the small time steps between the long time period from estimator or by interpolation, then this timescale makes the clock of the sender available with the receiver node at any point of time within some accuracy. This way we can say that we have also achieved *always on* synchronization.

3.2.2 Measurements and System Model:

Since the measurement data in our case is the value of time-offset between S and R at the resynchronization instances, we have made use of the time-offset equation derived in chapter 2, without considering the environmental terms. We have simulated the measurements using the following model:

$$\Delta t_{SR}(t_R) = (t_{0S} - t_{0R}) + \frac{\Delta f_{SR}}{f_o}(t_R - t_{0R}) + \frac{f_{dSR}}{2f_o}(t_R - t_{0R})^2 + \Psi_n(t_R) \tag{3.25}$$

where the terms have the same meaning as explained in the previous chapter.

For simplicity, we call this as

$$\tilde{y}(k) = \tilde{t}_{off}(k) = IO + FB \times t_k + FD \times t_k^2 + FN(t_k) \tag{3.26}$$

where,

IO = initial time-offset = $t_{0S} - t_{0R}$

FB = term due to frequency bias = $\frac{\Delta f_{SR}}{f_o}$

FD = term due to relative frequency drift = $\frac{f_{dSR}}{2f_o}$

t_{R0} = initial time on receiver's clock = 0

t_k = time on receiver's clock at k^{th} time-step, $1 < k \leq n$ (seconds)

The data is simulated at discrete time-steps of 1 second for some total time period of a few hours.

Comparing with the standard MV-SLS measurement model, we get,

The basis function matrix,

$H(k) = \begin{bmatrix} 1 & t_k & t_k^2 \end{bmatrix}$

And the parameter matrix is given as

$$\hat{x}(t_k) = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} = \begin{bmatrix} \hat{IO_k} & \hat{FB_k} & \hat{FD_k} \end{bmatrix} = \begin{bmatrix} (t_{0S} - t_{0R}) & \frac{\Delta f_{SR}}{f_o} & \frac{f_{dSR}}{2f_o} \end{bmatrix}$$

Here, the random clock jitter is used as the measurement noise, v, at any instance of time. Hence the measurement error covariance is given as, $R^{-1} = cov\{\Psi_n(t_R)\}$, which is assumed to be a zero-mean Gaussian random variable.

The receiver receives the measurements at the chosen sampling periods, which are mentioned ahead.

3.2.3 Initial or *a priori* conditions:

A priori conditions are needed to start the sequential process. We have considered 3 cases of initial conditions to see the effect of the choice of initial conditions on estimation. They are as follows:

1. The initial values for IO, FB and FD and the *a priori* error covariance matrix P_0 are assumed close to those given in other papers, theory and, found from experiment conducted by other groups in real situations.

2. Initial values of IO, FB, FD as well as P_0 are derived by choosing appropriate values for α and β as given in section 3.1.4.
3. Initial parameters and the initial estimation error covariance matrix are derived from batch estimation of a few initial measurement samples. For example it is assumed that the receiver receives and stores the first 5-10 measurements (timestamps) without running the estimator. This can be called as a training period. Then it runs a batch least squares estimation on these stored samples to derive x_0 and P_0, which are then used as *a priori* values for the MV-SLS algorithm which is started from the next data sample.

3.2.4 Simulations

The MV-SLS algorithm is simulated using the above design. Window is 1 sample, i.e. only 1 measurement is received from the sender at a re-synchronization instance, to update the parameters. These timestamps are received at a constant sampling period. The algorithm is tried for different values of sampling period, taken mostly from other experiments. It is also tried for different values of parameters as well as high and low levels of noise. The performance is also compared with theoretical bounds.

CHAPTER 4

KALMAN FILTER SOLUTION

In this chapter, we present the Kalman filter solution for achieving a long term synchronization. The dynamic and recursive nature of Kalman filter can prove useful to make the synchronization adaptive to the environmental effects. We first describe the Kalman filter, and then discuss our design ideas in forming the solution.

4.1 Description of Kalman Filter Algorithm

4.1.1 Sequential State Estimation

The Kalman filter extends the results of least squares techniques to allow estimation of parameters in the model of a *dynamic* system. The basic resulting equations as well as the probability concepts from sequential least squares estimation remain valid even in this case, upon making new interpretations of the matrices involved and accordingly adding extra calculations. These extensions in the previous calculations are necessary to take into account the 'motion' of the dynamical system between measurements and estimation epochs.

In the case of SLS estimation, it was assumed that the set of parameters that are being estimated remain constant - even on addition of new sets of data. The situation becomes complicated when the set of parameters being estimated is allowed to change during the estimation process. In such a case, when a new subset of data becomes available, it is desired to obtain an optimal estimate of the state of the system at that instant, so as to use the best current information, to base prediction of measured variable

upon. Therefore it is often called as 'sequential state estimation' instead of 'sequential parameter estimation'.

As in the SLS estimation, even here, it is assumed that we have with us, some measurements and a model which gives the relationship between the measured variable and the parameters to be estimated. But the main difference here is that there is also a dynamic model available for estimation of states, which gives a relationship between current and previous estimates of set of states. Thus, we have a measurement model and a system model. But of course, in practice, the true system model is not known. Hence in the estimation of the states, we also make use of the available measurements. A feedback term is thus used to give the best results.

Now, depending on the nature of the system and measurement models, the Kalman filter-estimator can be classified as : *Discrete-time Estimator* , *Continuous-Discrete Estimator* or *Continuous-time Estimator*. All these cases involve linear models. For non-linear models, there are *Extended Kalman Filters* having the same categories. For our case, we have designed a Discrete-time linear Kalman Filter, hence we restrict ourselves to the description of only that category.

4.1.2 Discrete-time Kalman Filter

The discrete-time Kalman filter [16] assumes that both the measurements and the system model are available in discrete-time form. The measurements as well as the model are supposed to be corrupted by noise and, are given as:

$$x_{k+1} = \Phi_k x_k + \Gamma_k u_k + \Upsilon_k w_k \tag{4.1}$$

$$\tilde{y}_k = H_k x_k + v_k \tag{4.2}$$

where $\tilde{y}_k$ is the m-dimensional measurement vector, x_k is the n-dimensional vector representing the state of the system at instant t_k, Φ_k is the $n \times n$ transition matrix and H is the $m \times n$ matrix of independent basis functions. The v_k is the m-dimensional measurement noise vector and w_k is the n-dimensional process noise vector. They are assumed to be zero-mean Gaussian white-noise processes, i.e., the errors are not correlated forward or backward in time; hence,

$$E\{v_k v_j^T\} = \begin{cases} 0, & k \neq j; \\ R_k, & k = j. \end{cases} \tag{4.3}$$

and

$$E\{w_k w_j^T\} = \begin{cases} 0, & k \neq j; \\ Q_k, & k = j. \end{cases} \tag{4.4}$$

Its is also assumed that v_k and w_k are uncorrelated, so that $E\{v_k w_k^T\} = 0$ for all k.

Now, we want to obtain the next estimate of the state ($\hat{x}_{k+1}$) by updating the current estimate ($\hat{x}_k$) based upon all $k+1$ measurement subsets. Thus for obtaining the next estimate, we first need to update the current estimate, using the current measurement. This is done by adding a feedback term which is a weighted difference between the current measurement and the estimated output. We can say that the final estimate at time t_k is the weighted average of the predicted stage at time t_k and the measurement at time t_k and the respective uncertainties. The weights are inversely proportional to the variances affecting the information. Thus the estimator is described by two coupled equations as:

$$\hat{x}_{k+1}^- = \Phi_k \hat{x}_k^+ + \Gamma_k u_k \tag{4.5}$$

$$\hat{x}_k^+ = \hat{x}_k^- + K_k[\tilde{y}_k - H_k \hat{x}_k^-] \tag{4.6}$$

The first equation is known as the *prediction* or *propagation* equation and the second is known as the *update* equation. Similarly, the '$-$' signs denote the estimate at the end of propagation stage, and the '$+$' sign denotes the updated estimate after using the

feedback from current measurement subset. K is called the Kalman gain and it can vary with time.

Also, as in the case of SLS, an error covariance matrix is defined for the errors in the estimation of states as,

$$P_k^- \equiv E\{(\hat{x}_k^- - x_k)(\hat{x}_k^- - x_k)^T\} \tag{4.7}$$

$$P_k^+ \equiv E\{(\hat{x}_k^+ - x_k)(\hat{x}_k^+ - x_k)^T\} \tag{4.8}$$

Now, it just remains to have an expression for the estimation error covariance and the gain. These are given as:

For *Update* stage,

$$K_k = P_k^- H_k^T [H_k P_k^- H_k^T + R_k]^{-1} \tag{4.9}$$

$$P_k^+ = [I - K_k H_k] P_k^- \tag{4.10}$$

For *Propagation* stage,

$$P_{k+1}^- = \Phi_k P_k^+ \Phi_k^T + \Upsilon_k Q_k \Upsilon_k^T \tag{4.11}$$

It can be seen that a small uncertainty in measurements ($R \to 0$) makes gain K high, making the estimator dependant on measurements and very small system dynamics ($Q \to 0$) makes the gain K very small, so the estimator depends on the system model. One more thing to be noted is that the state error covariance matrix does not depend on the measurements, but only on the modelled statistics, therefore, it is possible to get an idea of the uncertainty and error in the estimates without doing any measurements.

Thus, the working of the discrete-time Kalman filter can be explained as follows. We are given initial estimates of the states and initial error covariance matrix. If a measurement is available at the initial time, then the states and covariance are updated using equations (4.6), (4.9) and (4.10) with $\hat{x}_0^- = \hat{x}_0$ and $P_0^- = P_0$. These state estimates

and covariance are propagated to the next time step using equations (4.5) and (4.11). If a measurement is not available at the initial time, then the estimates and covariance are first propagated to the next time step and then updated. This process is carried out sequentially until all measurement times have been used. The gain of the Kalman filter is derived from the minimum variance approach.

Thus, we can say that, the propagation stage gives a time update through a *prediction* of $\hat{x}^-$ and covariance P^-. The update stage gives a *correction* based on the measurements to yield a new *a posteriori* estimate $\hat{x}^+$ and covariance P^+. This *predictor-corrector* form of the Kalman filter is best-suited for estimation and prediction problems in real-time dynamic systems.

4.2 The Design

Kalman filter approach can be viewed as an extension to the MV-SLS algorithm so as to incorporate the effect of environmental noise and the dynamic clock drift on the time offset between two nodes. There have been numerous attempts in past on using Kalman filter as a tool in the formation of stable timescales like TA(NIST) or GPS composite clocks, where the goal is to produce a virtual clock which is more stable than any of the physical clocks involved, in both short term and long term [17] [18].

Our approach is motivated from those designs. Since the basic underlying case of the Kalman filter applied to the estimation of the difference between two clocks is the same and is already tried in the real-time cases, this has led us to test it in the sensor network time synchronization problem. The difference in the two scenarios can be that the sensors run on very cheap and simple oscillators and can also be exposed to severe environmental conditions, making the problem worse in most situations. Also, the clock states are defined between the clocks of two nodes, and not a clock of a single node and some ideal clock.

Thus in a way, we have reproduced those findings and tried to apply it in case of sensor networks.

4.2.1 Estimation problem:

The estimation problem and scenario remains almost the same as that described in case of MV-SLS solution. A receiver (R) node receives *timestamps* (*synchronization beacons*) from the sender (S) at regular sampling intervals which contain the value of sender's clock. Subtracting the value of timestamps from the current reading of receivers clock, we get the 'time offset between S and R' according to R's clock. These are the measurements. Using these measurements and the Kalman estimator, we wish to build a *time-conversion-scale* between the sender and the receiver according to the receiver's clock. This virtual clock will be available with the receiver at any point of time, and tells the receiver the current time in sender's clock.

Now, it is assumed that at least one of the clocks undergo changes in it's frequency due to environmental and temperature effects, which has high and unmodelled variations. Effectively, we can say that the clock experiencing such effects will drift with different rates at different times, as opposed to a constant drift assumed previously.

Then, once again, the estimator needs to estimate the clock parameters between S and R, and predict the next value of timeoffset depending on these estimates, so as to form a time-scale between two adjacent synchronization instances in this dynamic scenario. If the synchronization error, (which is the residual error) is very small *even for a high sampling period* and even in the *presence of dynamics in clock parameters*; then we can say that our algorithm has achieved, within some accuracy, a long-term, always ON synchronization, which is also adaptive to the environmental effects!

4.2.2 Model for Kalman Filter

The main difference of this approach from the SLS is in the way the expression for phase-offset between two nodes is presented. We call x_{SR} as the phase-offset state, y_{SR} is the frequency bias state and z_{SR} is the relative drift state between the sender's clock and the receiver's clock. Then the evolution of this offset between S & R from time step $(t-\tau)$ to t is given by following equations:

$$x_{SR}(t) = x_{SR}(t-\tau) + \tau.y_{SR}(t-\tau) + \frac{1}{2}\tau^2.z_{SR}(t-\tau) + w_{x_{SR}}(t) \tag{4.12}$$

$$y_{SR}(t) = y_{SR}(t-\tau) + \tau.z_{SR}(t-\tau) + w_{y_{SR}}(t) \tag{4.13}$$

$$z_{SR}(t) = z_{SR}(t-\tau) + w_{z_{SR}}(t) \tag{4.14}$$

where the process noise vector, which specify the level of relative noise components between the two clocks, is given as

$$W_{SR}(t) = [\ w_{x_{SR}}(t) \quad w_{y_{SR}}(t) \quad w_{z_{SR}}(t)\ ]^T \tag{4.15}$$

We can explain this as follows: the next value of the relative drift between two nodes is obtained by adding some noise to the previous value of the relative drift. As discussed in some earlier chapters, the sources of this noise lie well in the environmental effect and also the aging of the oscillator. Since the drift of each clock affects its frequency, the frequency bias between the two clocks changes too. The value of the frequency bias at the next time step is found by adding to the last estimate of frequency bias, a term, which multiplies the change in relative drift by the time elapsed between two subsequent synchronization instances, and also by adding some noise representing short-term random frequency variations. Similarly, the phase-offset at next time step is given by adding some terms and noise to the last value of phase-offset. These terms involve multiplying the frequency bias by time elapsed and relative drift by square of time elapsed.

The above W and its covariance matrix Q are derived from the Hadamard and Allan variances. The complete description of these components are out of the scope of this work. Interested readers can refer to [18].

4.2.3 Measurement and System Model

The measurements can be simulated using the above phase offset expression and adding some measurement noise to it. They can be simulated for a continuous case or a discrete case by choosing a transition matrix, to reflect the intervals as which the change in clock states takes place. For simulating a sensor-network scenario, we have assumed that the clock drift changes at every iteration due the environmental and other noise. So,

$$y_k = x_S(t_k) - x_R(t_k) = H.x_k + v_k \tag{4.16}$$

where y_k is the phase or time offset between S and R, and $x_S(t_k)$ and $x_R(t_k)$ are the current phase states of the sender and the receiver respectively. Here, x_k is same as $x_{SR}(t)$ in the equation 4.12, and y_k should not be confused with $y_{SR}(t)$. Even though these expressions are in terms of phase offset, they can be easily converted in terms of time offset, by dividing each term by the ideal frequency of nodes' clocks. Doing so will not change the nature of the problem which is basically estimating the dynamic states and predict the next value of offset between nodes.

This leads us to a basis vector

$H = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$

The random clock jitter is used as the measurement noise, v, at any instance of time. Hence the measurement error covariance is given as, $R^{-1} = cov\{\Psi_n(t_R)\}$ which is assumed to be a zero-mean Gaussian random variable.

The receiver receives the measurements at the chosen sampling periods.

The system model is defined by the above-mentioned three equations for the phase, frequency and drift state. Hence the state vector is:

$x_k = [\ x_{SR}(t) \quad y_{SR}(t) \quad z_{SR}(t)\ ]$ which is updated at every re-synchronization instance, using Φ and W.

The transition matrix gives us the transition of states from time step t_k to t_{k+1}. It is,

$$\Phi = \begin{pmatrix} 1 & \tau & \frac{\tau^2}{2} \\ 0 & 1 & \tau \\ 0 & 0 & 1 \end{pmatrix}$$

Where the terms in x and Φ has same meaning as described above. The process noise covariance matrix Q is taken as variance of the combined noise vector W. Since all variances and covariances of all three noise components are covered in this definition of Q, there is no need to consider a separate Υ.

4.2.4 Simulation

Initial conditions for the states are assumed from various other papers or experiments. Even the initial estimate error covariance matrix is assumed in the range of an Identity matrix.

The discrete time Kalman filter is simulated using the above design. Window size is 1, i.e. only 1 measurement is received from the sender at a time to update the Kalman states. These timestamps are received at a constant sampling interval. The algorithm is tried for different values of sampling period, taken mostly from other experiments.

Kalman filter solution was also tried on different values of clock parameters as well as high and low values of three process noise components. At every re-synchronization point, the synchronization error is calculated between the estimated value of time-offset, and the measured value of time-offset.

CHAPTER 5

FUZZY LOGIC SOLUTION

In this chapter, we present the Fuzzy Logic solution for achieving a long term and adaptive synchronization. We start with describing the Fuzzy Logic Systems, followed by our design to the current problem.

In the *model-free* approach of fuzzy logic, rules are extracted from the numerical data itself, and hence it is very applicable when real data do not agree with an assumed data-generating model, which happens in many real-life situations. Thus uncertainties can be handled within the framework of FL. This is the reason behind trying this novel approach to the time synchronization problem, especially when the offset of two clocks is highly affected by un-modelled environmental factors in an outdoor scenario.

5.1 Fuzzy Logic Systems

For many engineering problems, two form of knowledge exists: *objective knowledge* e.g. mathematical models and *subjective knowledge* that is hard to quantify using traditional mathematics e.g. rules, expert information, design requirements etc. These two forms of knowledge can be coordinated in a logical way using *fuzzy logic* (FL). Fuzzy logic leads to a system called *fuzzy logic system* (FLS). In general, a FLS is a nonlinear mapping of an input data (feature) vector into a scalar output.

5.1.1 From Crisp Logic to Fuzzy Logic

A fuzzy set F is a generalization of a crisp set. It is defined on a universe of discourse X and is characterized by a membership function $\mu_F(x)$ that takes on values

in the interval [0, 1]. Extension from crisp logic to fuzzy logic is made by replacing the bivalent membership function of crisp logic to fuzzy membership function. A membership function provides a measure of the degree of similarity of an element in X to the fuzzy set. A fuzzy set F in X may be represented as a set of ordered pairs of a generic element x and its grade of membership function, $\mu_F(x)$, i.e.

$$F = \{(x, \mu_F(x))|\forall x \in X\} \tag{5.1}$$

Hence the IF-THEN statement like "IF x is A, THEN y is B", where $x \in X$ and $y \in Y$, has a membership function $\mu_{A \longrightarrow B}(x, y)$ where $\mu_{A \longrightarrow B}(x, y) \in [0, 1]$. This is given as,

$$\mu_{A \longrightarrow B}(x, y) \equiv \mu_A(x) \bigstar \mu_B(y) \tag{5.2}$$

where $\bigstar$ suggests a fuzzy intersection operation, called as 't-norm'. It can be a product or minimum, and is referred to as *Mamdani implication.*

Membership functions characterize fuzzy sets, be they type-1 or type-2. The most commonly used shapes for membership functions are triangular, trapezoidal, piecewise-linear, Gaussian and bell-shaped. Greater resolution is achieved by using more membership functions at the price of greater computational complexity. One of the great strength of FL lies in making the membership functions overlap. That way, we can distribute our decisions over more than one input class, which makes FL systems robust.

For type-1 fuzzy sets the terms membership function and grade of membership function are used interchangeable. A type-1 fuzzy set, F, in terms of a single variable x is expressed as given by (5.1). Type-1 membership function, $\mu_A(x)$, is constrained to be between 0 and 1 for all $x \in X$.

Now, for p variables, when the membership function $\mu_{A_{\mathbf{x}}}(x_1, x_2, ..., x_p)$ is separable, then

$$\mu_{A_{\mathbf{x}}}(\mathbf{x}) = \mu_{A_{x1}}(x_1) \bigstar \mu_{A_{x2}}(x_2) \bigstar ... \bigstar \mu_{A_{xp}}(x_p) \tag{5.3}$$

which is written as

$$\mu_{A_{\mathbf{x}}}(\mathbf{x}) = \mu_{X_1}(x_1) \bigstar \mu_{X_2}(x_2) \bigstar ... \bigstar \mu_{X_p}(x_p) \tag{5.4}$$

A fuzzy logic described completely in terms of type-1 fuzzy sets is called a *type-1* FLS.

5.1.2 Components of a FLS

We have restricted ourselves to explaining the basic type-1 singleton FLS, which has been used in our solution. This FLS is also known as Mamdani FLS. In this case, all the fuzzy sets are type-1 and measurements are perfect and treated as crisp values, i.e., singletons.

The rule-based FLS contains 4 main components - rules, fuzzifier, inference engine and defuzzifier, that are inter-connected [19]. Once the rules have been establish, a FLS can be viewed as a mapping from inputs to output/s, which can be expressed as $y = f(x)$. We want to write a mathematical formula that relates the output of the FLS to its inputs.

Fuzzifier

A fuzzifier maps crisp numbers into fuzzy sets. It is needed to activate rules, which have fuzzy sets associated with them. The fuzzifier maps a crisp number $x = (x_1, ..., x_p)^T \in X_1 \times ... \times X_p \equiv \mathbf{X}$ into a fuzzy set $A_{\mathbf{x}}$ in $\mathbf{X}$. A singleton fuzzifier is a fuzzy singleton, i.e. each of the separable components of $\mu_{A_{\mathbf{x}}}(\mathbf{x})$ is a fuzzy singleton so that $\mu_{X_i}(x_i) = 1$ for $x_i = \acute{x}$ and $\mu_{X_i}(x_i) = 0$ for $\forall x_i \in X_i$ and $x_i \neq \acute{x_i}$.

Rules:

The rules can be expressed as logical implications, i.e. in form of IF-THEN statements. The IF part of a rule is called *antecedent* and the THEN part is called its *consequent*. Fuzzy sets are associated with the antecedents and consequents of the rules

and sometimes also with the inputs to and outputs of the FLS. Membership functions are used to describe these fuzzy sets.

Consider a type-1 FLS having p inputs $x_1 \in X_1, ..., x_p \in X_p$ and one output $y \in Y$. Suppose it has M rules, where the l^{th} rule has the form

$$R^l : \text{IF } x_1 \text{ is } F_1^l \text{ and ... and } x_p \text{ is } F_p^l, \text{ THEN } y \text{ is } G^l \; l = 1, ..., M$$

This rule represents a type-1 fuzzy relation between the input space $X_1 \times .... \times X_p$ and the output space, Y, of the FLS.

Fuzzy Inference Engine

In the fuzzy inference engine, fuzzy logic principles are used to combine fuzzy IF-THEN rules from the fuzzy rule base into a mapping from fuzzy input sets in $X_1 \times .... \times X_p$ to fuzzy output sets in Y. Each rule is interpreted as a fuzzy implication, and can be expressed as:

$R^l : F_1^l \times ... \times F_p^l \longrightarrow G^l = A^l \longrightarrow G^l \; l = 1, ..., M$

R^l is described by the membership function $\mu_{R^l}(\mathbf{x}, y)$ where

$$\mu_{R^l}(\mathbf{x}, y) = \mu_{A^l \longrightarrow G^l}(\mathbf{x}, y) = [T_{i=1}^p \mu_{F_i^l}(x_i)] \bigstar \mu_{G^l}(y) \tag{5.5}$$

where $\mathbf{x} = (x_1, ..., x_p)^T$. It is assumed that mamdani implications are used, multiple antecedents are connected by '*and*' t-norm (T is short for t-norm).

The p-dimensional input to R^l is given by the fuzzy set A_x, whose membership function is given by

$$\mu_{A_\mathbf{x}}(\mathbf{x}) = \mu_{X_1}(x_1) \bigstar ... \bigstar \mu_{X_p}(x_p) = T_{i=1}^p \mu_{X_i}(x_i) \tag{5.6}$$

Each rule R^l determines a fuzzy set $B^l = A_x \circ R^l$ in Y such that,

$$\mu_{B^l}(y) = \mu_{A_\mathbf{x} \circ R^l}(y) = sup_{\mathbf{x} \in \mathbf{X}} [\mu_{A_\mathbf{x}}(\mathbf{x}) \bigstar \mu_{A^l \longrightarrow G^l}(\mathbf{x}, y)], y \in Y \tag{5.7}$$

where, the term '*sup*' indicates the supremum operation, which is the *maximum*, on discrete universes of discourse.

This is the relationship between the input fuzzy set that excites a one-rule inference engine and the fuzzy set at the output of the engine.

For the singleton fuzzification, the *sup-star* composition is simplified because $\mu_{X_i}(x_i)$ is non-zero only at one point $x_i = \acute{x}_i$; hence,

$$\mu_{B^l}(y) = \mu_{G^l}(y) \bigstar [\mu_{F_1^l}(\acute{x}_1) \bigstar ... \bigstar \mu_{F_p^l}(\acute{x}_p)], y \in Y \tag{5.8}$$

the term in the bracket in this equation is referred to as the *firing level*. At each value of $y \in Y$, say $\acute{y}$, $\mu_{B^l}(\acute{y})$ is a crisp number.

The final fuzzy set B is determined by all M rules, and is obtained by combining B^l and its associated membership function $\mu_{B^l}(y)$ for all $l = 1, ..., M$ as:

$$B = A_{\mathbf{x}} \circ [R^1, ..., R^M] \tag{5.9}$$

Defuzzifier

In many applications of FLS, crisp numbers must be obtained at its output. This is accomplished by the defuzzifier. Defuzzification produces a crisp output for FLS from the fuzzy sets that appear at the output of the inference block. Many defuzzifiers have been proposed, such as, height, centroid, center-of-sums, center-of-sets, mean-of-maxima etc. Since we have used a height defuzzifier in our solution, once again, we restrict ourselves to describing only that.

The height defuzzifier replaces each rule output fuzzy set by a singleton at the point having maximum membership in that output set, and then calculates the centroid of the type-1 set comprised of these singletons. The output of the height defuzzifier is given as:

$$y_h(\mathbf{x}) = \frac{\sum_{l=1}^{M} \bar{y}^l \mu_{B^l(\bar{y}^l)}}{\sum_{l=1}^{M} \mu_{B^l(\bar{y}^l)}} \tag{5.10}$$

where $\bar{y}^l$ is the point having maximum membership in the l^{th} output set, and its membership grade in that set is $\mu_{B^l}(\bar{y}^l)$.

Fuzzy Basis Functions

Fuzzy Basis Functions (FBF) can give us a mathematical formula that maps a crisp input $\mathbf{x}$ into a crisp output $y = f(\mathbf{x})$.

If we choose singleton fuzzification, max-product composition, product implication, and height defuzzification, then we get,

$$y(\mathbf{x}) = f(\mathbf{x}) = \frac{\sum_{l=1}^{M} \bar{y}^l \prod_{i=1}^{p} \mu_{F_i^l}(x_i)}{\sum_{l=1}^{M} \prod_{i=1}^{p} \mu_{F_i^l}(x_i)} \tag{5.11}$$

Generalizing this, we get,

$$y(\mathbf{x}) = f(\mathbf{x}) = \sum_{l=1}^{M} \bar{y}^l \Phi_l(\mathbf{x}) \tag{5.12}$$

where $\Phi_l(\mathbf{x})$ is called a *fuzzy basis function*. For the above case,

$$\Phi_l(\mathbf{x}) = \frac{\prod_{i=1}^{p} \mu_{F_i^l}(x_i)}{\sum_{l=1}^{M} \prod_{i=1}^{p} \mu_{F_i^l}(x_i)} \tag{5.13}$$

where $l = 1, 2, ..., M$ are the rules.

This way, we can refer to FLS as a FBF expansion. Though each FBF is affected by all the rules; each rule contributes a basis function, so the jth FBF is associated with the jth rule.

5.1.3 Designing FLS

Designing a FLS can be viewed as approximating a function or fitting a complex surface. Numerical rules and their associated FBFs required for the design can be extracted from the numerical training data itself. After choosing the type of fuzzification, implication, t-norm, defuzzification and membership functions, we need to fix the parameters of membership functions. For a given set of input-output pairs, optimal fit can be obtained by tuning. There exists a multitude of design and tuning methods which can be used to construct FLSs having different characteristics.

Fuzzy logic is extensively used for forecasting of time-series. Let $s(k)$ $(k = 1, 2, ..., N)$ be a time series. Measured values of $s(k)$ are denoted $x(k)$, where $x(k) = s(k) + n(k)$, where $n(k)$ denotes the measurement errors i.e. noise. If the measurements are noise-free, then $x(k) = s(k)$. In such case, most the problem is described as:

"We have a collection of N input-output data training pairs, $(\mathbf{x}^{(1)} : y^{(1)}), (\mathbf{x}^{(2)} : y^{(2)}), ..., (\mathbf{x}^{(N)} : y^{(N)})$, where $\mathbf{x}$ is the vector input and y is the scalar output of a FLS. Then our goal is to completely specify the FLS using training data."

This designed FLS can then be put into forecasting problem on a testing dataset. The training data are used in the FLS forecaster to establish its rules.

There are at least three ways to extract rules from the numerical training data:

1. Let the data establish the centers of the fuzzy sets that appear in the antecedents and consequents of the rules.
2. Pre-specify fuzzy sets for the antecedents and consequents and then associate the data with these fuzzy sets
3. Establish the architecture of a FLS and use the data to optimize its parameters.

5.2 The Design

5.2.1 Problem statement

In a similar scenario as described before, a receiver node receives timestamps from the sender and from that, calculates the time offset between their clocks at that instance of time. These timestamps (synchronization beacons) are sent at periodic intervals. These values of time offset are the measured data. Now, the difference in this case from the last two approaches is that, in fuzzy logic, there will be no estimation of clock parameters! Using the available measurements, we want to predict the value of the time offset between S and R at the next time step or next resynchronization instance. A time scale can be

build, between the sender and the receiver, using these predicted values. If the error between the measured and predicted value of time-offset, which is the synchronization error, is very small even for a high sampling period, then we have achieved a long-term synchronization. Even for this case we have considered both constant and dynamic clock parameters due to environmental effects.

As it is observed from the nature of problem, it is a case of forecasting a time-series using FLS.

5.2.2 Measurements data

For simulating the data, we have used the same expressions as that used for MV-SLS or Kalman Filter, given by equation (3.25) or (**??**). This data is first detrended by calculating the trend by a simple sliding window technique and subtracting it from the data samples. This detrended data was the input to the fuzzy logic system (FLS).

Also, after obtaining the output of the FLS, the trend was added into it to get the predicted value of the time-offset.

5.2.3 Forecasting a time-series using FLS

The detrended data, i.e. the input time series to FLS was first divided in training and testing subsets. Thus first few samples are used only for training - which is the learning period of FLS - and not for prediction.

Similar to the design approach taken by [20], the problem of forecasting a time-series i.e. prediction, can be defined as:

"Given a window of p past measurements of a time-series $x(k)$, namely $x(k-p+1), x(k-p+2), ..., x(k)$, determine an estimate of a future value of x, $\hat{x}(k+l)$, where p and l are fixed positive integers."

In our case, we just want to predict the next value of the data, so $l = 1$ and we have a single-stage forecaster of x.

We are given a collection of N noise-free data points, $x(1), x(2), ..., x(N)$ that is partitioned into a *training* data subset with D data points, $x(1), x(2), ..., x(D)$, and a *testing* data subset with N - D data points, $x(D+1), x(D+2), ..., x(N)$. A window of p points is used to forecast the next data point; so there are at most D - p training pairs, $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(D-p)}$, where,

$$\mathbf{x}^{(1)} = [x(1), x(2), ..., x(p), x(p+1)]^T \tag{5.14}$$

$$\mathbf{x}^{(2)} = [x(2), x(3), ..., x(p+1), x(p+2)]^T \tag{5.15}$$

$$\mathbf{x}^{(D-p)} = [x(D-p), x(D-p+1), ..., x(D-1), x(D)]^T \tag{5.16}$$

In the above equations, the first p elements of any training pair are the inputs to the forecaster and the last element is the desired output of the forecaster., i.e.,

$\mathbf{x}^{(t)} = [p \times 1 \text{ input, desired output}]^T = [x_1^t, x_2^t, ..., x_p^t, x_{p+1}^t]^T$

where $t = 1, 2, ..., D - p$.

After the training period the rules were fixed, and then the prediction takes place using these rules in the testing period.

5.2.4 Basic Architecture and Initial conditions

We use four antecedents ($p = 4$), which are the previous 4 data samples for forecasting the next sample of data, which becomes the consequent. i.e. $x(k-3), x(k-2), x(k-1), x(k)$ were used to predict $x(k+1)$. This is the same for training and testing subsets. Number of training points were chosen to be about 30 for a dataset of maximum 100 samples.

We have chosen *singleton fuzzification* and *product implication* and *Height* Defuzzifier.

Each antecedent has two membership functions (MF) - High and Low. These are chosen to be Gaussian. The initial consequent fuzzy set comprises of random numbers in the given data range. Gaussian membership function is expressed as:

$$\mu_{F_i^l}(x_i) = \exp\left\{-\frac{1}{2}\left[\frac{x_i - m_{F_i^l}}{\sigma_{F_i^l}}\right]^2\right\} \tag{5.17}$$

where $i = 1, ..., p$ and $l = 1, ..., M$.

The number of rules are $2^4 = 16$. Rules are designed as:

R^l : IF $x(k-3)$ is F_1^l and $x(k-2)$ is F_2^l and $x(k-1)$ is F_1^l and $x(k)$ is F_2^l, THEN $x(k+1)$ is G^l.

where the height of G^l is $\bar{y}^i$

The initial mean and standard deviation of the antecedent membership function were derived from the mean m_t and std σ_t of training data set. Initial mean for 'High' MF was $m_t + \sigma_t$ and that for 'Low' MF was $m_t - \sigma_t$, and the std for both MFs was σ_t. So, we have total 144 parameters.

5.2.5 Back Propagation Method

After setting up the initial design, a steepest descent algorithm (back-propagation) was used to train these parameters based on the training data.

In back-propagation method, given an input-output training pair $(\mathbf{x}^{(i)} : y^{(i)})$, FLS is designed in such a way that the following error function is minimized:

$e^{(i)} = \frac{1}{2}[f(\mathbf{x}^{(i)}) - y^{(i)}]^2 \; i = 1, ..., N$

From equation (5.11), it is seen that f is completely characterized by $\bar{y}^l$, $m_{F_k^l}$ and $\sigma_{F_k^l}$. Steepest descent algorithm provides recursions to update all of these design parameters. The learning parameter is chosen to be about ten percent of the data range to provide the best results.

After training, the rules are fixed and the this FL forecaster is used for remaining points in the testing subset. We calculate the synchronization error between the measured and predicted value of the timeoffset.

5.2.6 Sequential Design

The design described above works well mostly with batch mode of prediction; but in case of problems like the one we are trying to address, it can also be converted into a sequential forecasting system.

The basic design and problem statement remains the same even for the sequential design. In this case, the basic architecture and initial rules are set ahead of time. As the data samples start coming in, the FLS stores the samples to first calculate the trend and detrend the data. When stored samples are equal to the number of antecedents, it is fed to FLS to predict the next sample. Adding the trend to this value gives us the prediction of next data sample. When the next data sample is actually received, the same process repeats. Also, at this re-synchronization instance, the error between predicted and measured value of data is calculated, which is used in a back-propagation algorithm along with a learning parameter to train the parameters of FLS. These new values of parameters are used for prediction at the next time-step, resulting in a better online performance.

CHAPTER 6

COMPUTER SIMULATIONS AND COMPARISONS

6.1 Simulations

We have tested and compared the performances of all the three algorithms on various datasets, using the simulation software MATLAB. All datasets are of the time offset between the sender node and a single receiver node. The problem, as described earlier is to predict the next value of this time-offset using three algorithms. The y-axis shows the measured and estimated values of the time-offset, while the x-axis shows the time in that receiver's clock at which the measurements were received. These specific times are called as re-synchronization instances. At every interval of 1 second on receiver's clock, the mean squared error (synchronization error) between the predicted value of time-offset and the actual measurement is calculated and plotted for all three approaches.

The first dataset incorporates the effect of clock drift over a period of 3 hours. These measurements are generated as given in chapter 3, and has an additional term compared to the short-term scales to compensate for the clock drift. No environmental noise is added in this dataset, and also the parameters are kept constant.

The second dataset is generated from a model similar to that used for the Kalman filter, which borrows the idea of [18]. Here, the main difference is that the clock parameters are made dynamic, which is very close to the real-life situation. But the added noise due to environmental terms is kept low, and is not seen significantly in this 1 hour run.

Both these datasets can be considered to be similar to an indoor-scenario, where the clock parameters and hence the offset between 2 nodes' clock will vary in a non-linear pattern, yet without any drastic random changes.

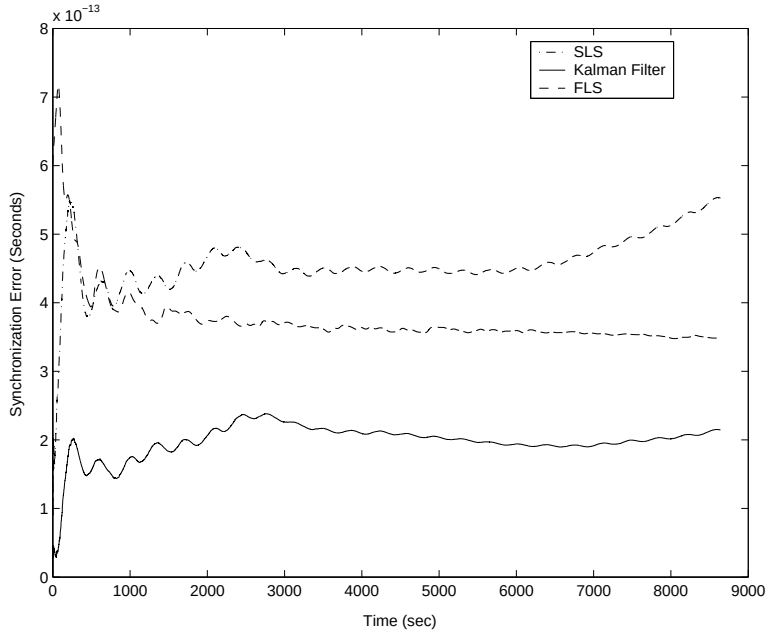

Figure 6.1. Mean squared synchronization error for dataset 1.

The third data set is an ideal example of time-offset variation in an outdoor scenario. Here, there is some variation in the clock parameters after every few seconds, and also the level of environmental noise is high, which results in highly random changes in clock drift. Various shapes of this dataset, implementing random changes in drift and thus offset, are used for the performance measure of the three algorithms.

The fourth dataset is similar to the third one, in terms of dynamic parameters and a considerable environmental noise; but it is taken over a shorter time period of 1 hour. The purpose of this data is to see how well the algorithms perform on a data with high short-term noise.

6.2 Observations

The simulations on different datasets lead us to following results:

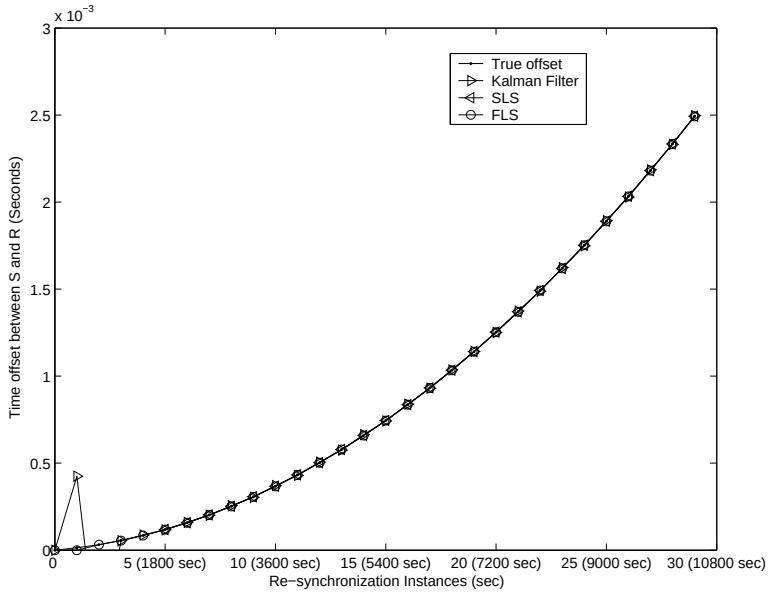

Figure 6.2. Measured and estimated values of time offset between Sender and Receiver for Dataset 1.

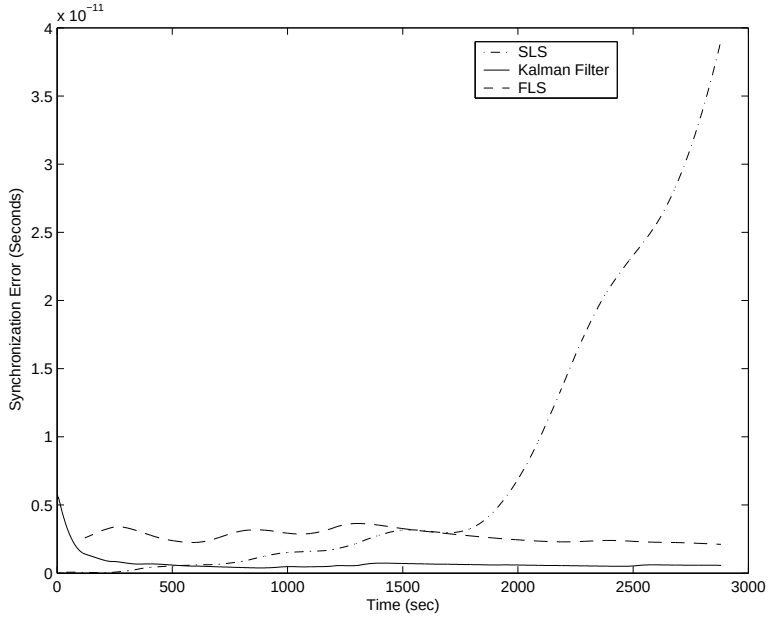

Figure 6.3. Mean square synchronization error for dataset 2.

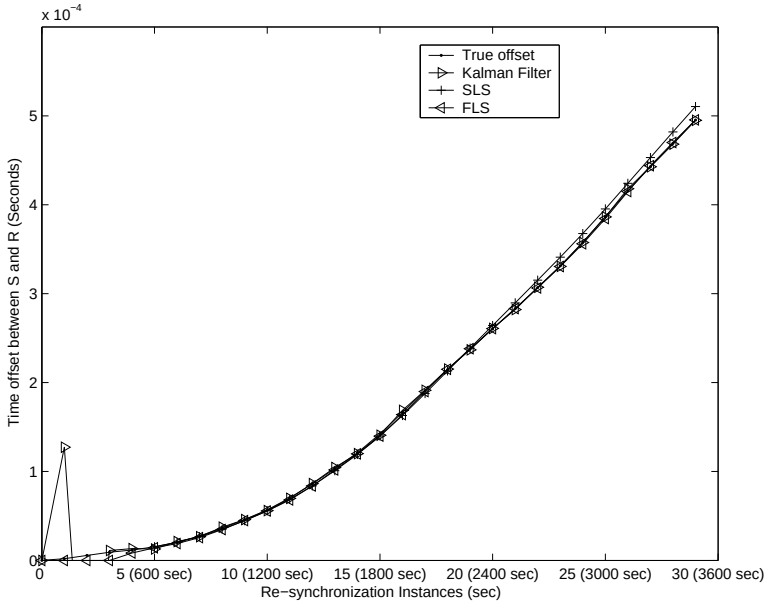

Figure 6.4. Measured and estimated values of time offset between Sender and Receiver for Dataset 2.

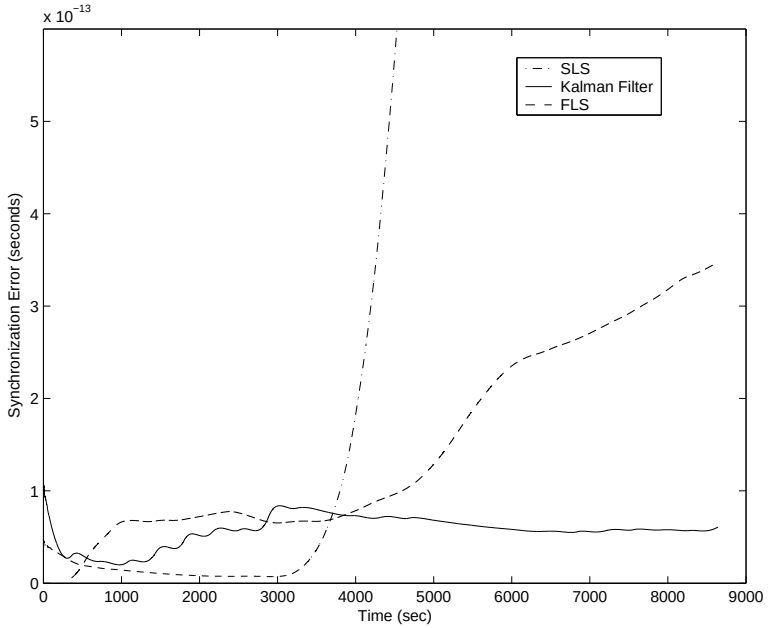

Figure 6.5. Mean square synchronization error for dataset 3a.

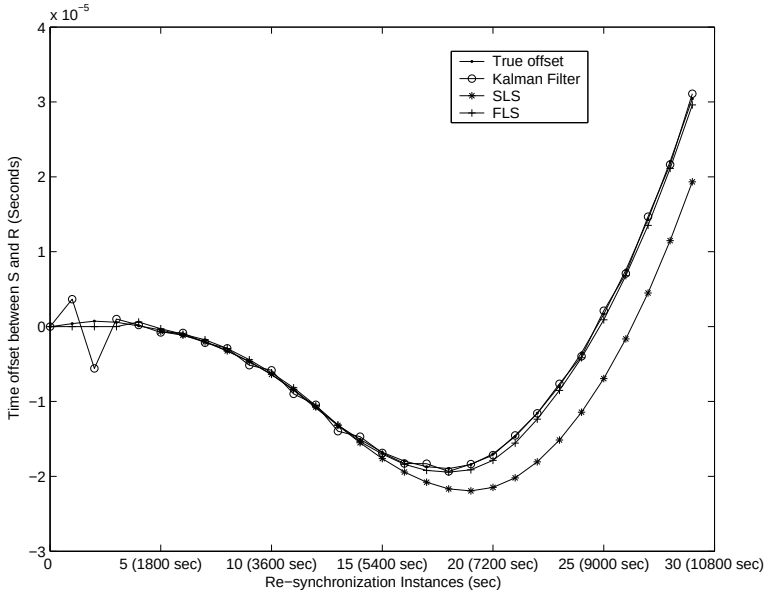

Figure 6.6. Measured and estimated values of time offset between Sender and Receiver for Dataset 3a.

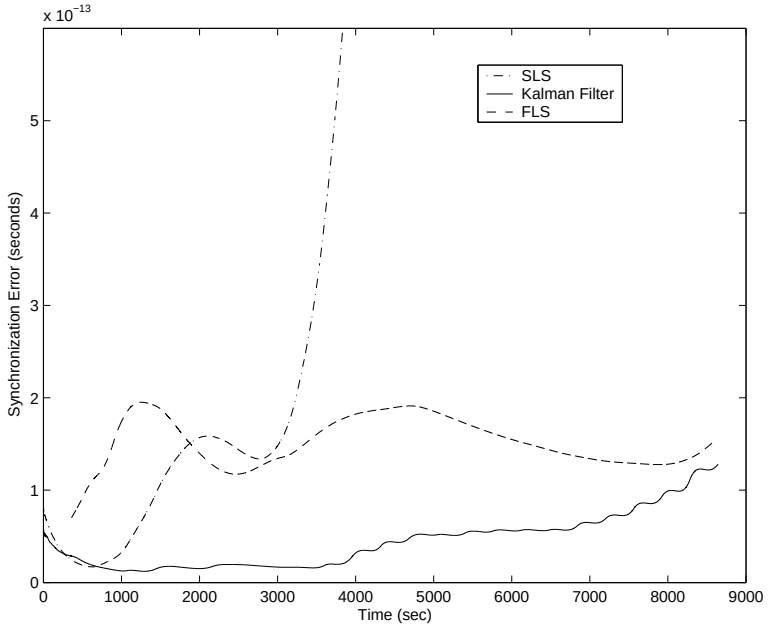

Figure 6.7. Mean square synchronization error for dataset 3b.

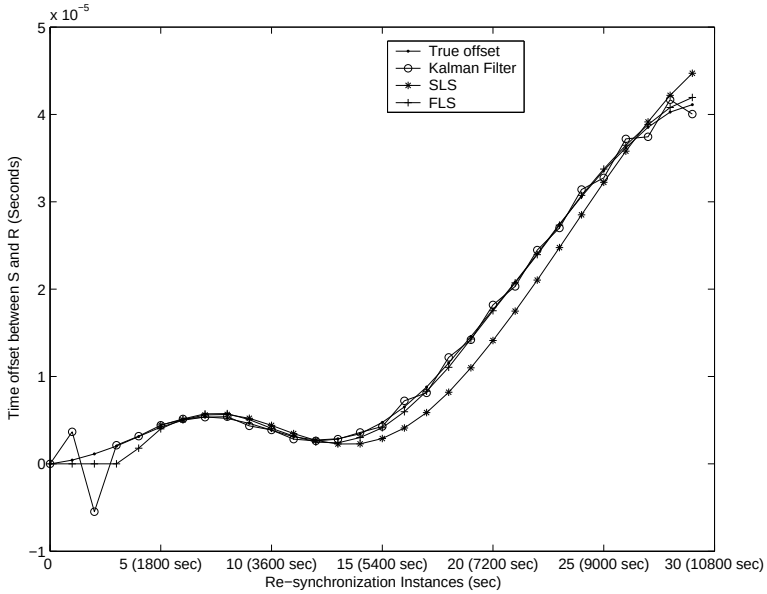

Figure 6.8. Measured and estimated values of time offset between Sender and Receiver for Dataset 3b.

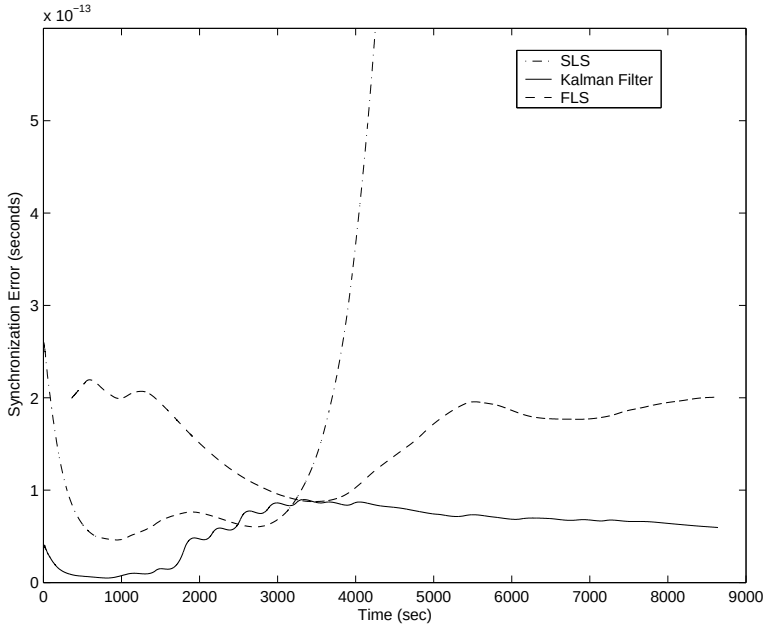

Figure 6.9. Mean square synchronization error for dataset 3c.

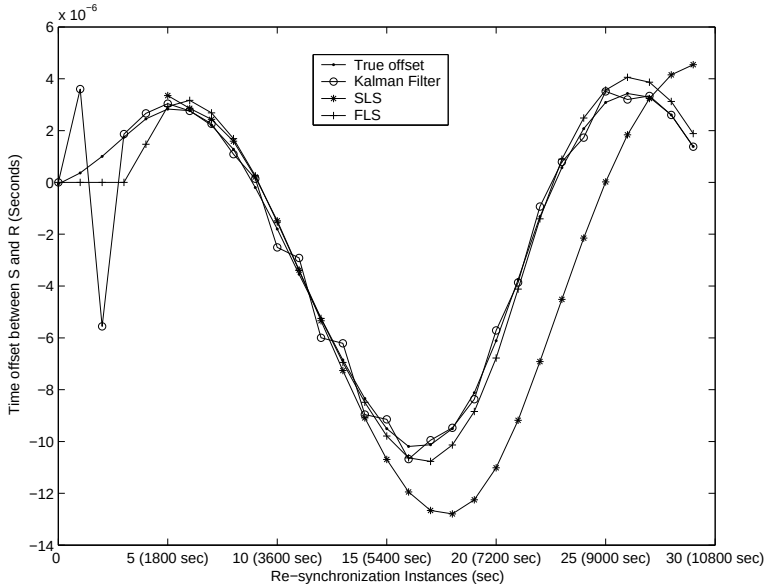

Figure 6.10. Measured and estimated values of time offset between Sender and Receiver for Dataset 3c.

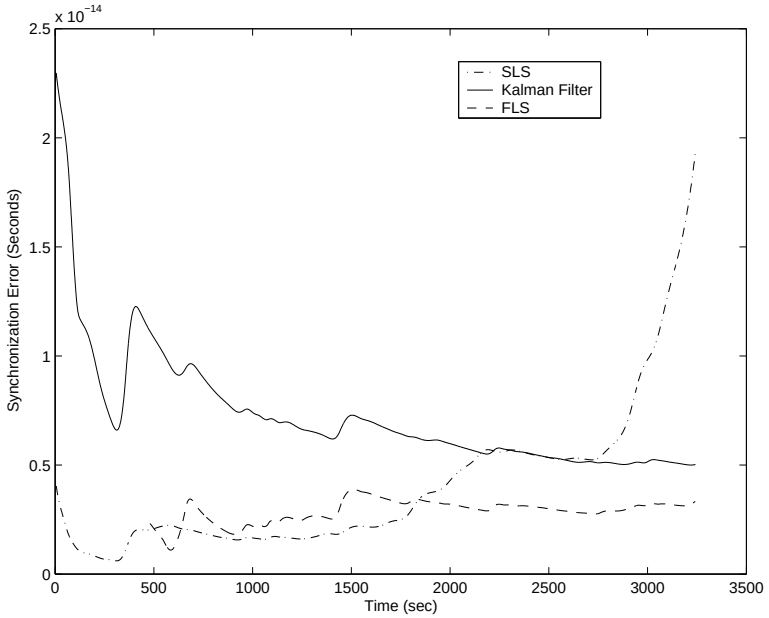

Figure 6.11. Mean square synchronization error for dataset 4.

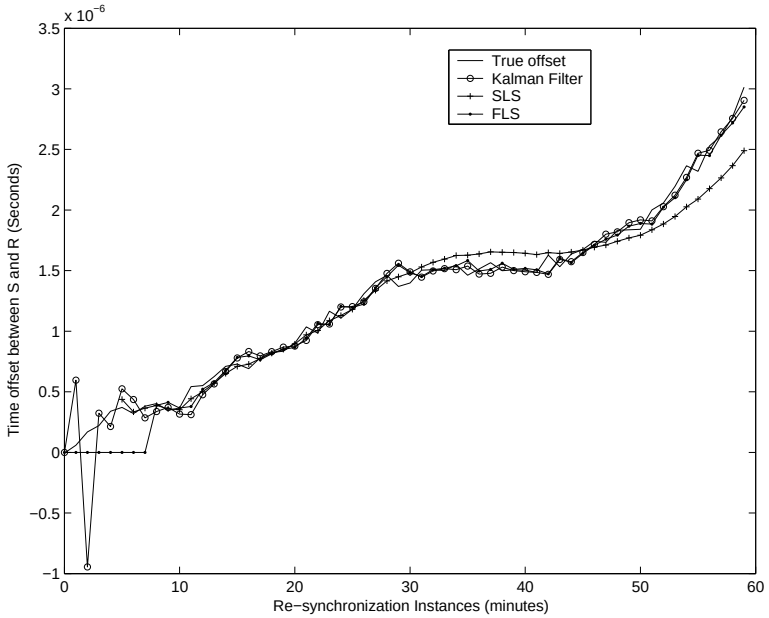

Figure 6.12. Measured and estimated values of time offset between Sender and Receiver for Dataset 4.

- All the methods are successful in forming a time conversion scale between sender and receiver over time, by processing the timestamps sequentially.
- All the methods achieve a very good precision on most of the datasets. The mean squared error is in the range of 10^{-12} or lower, which means that the nodes will be synchronized within an accuracy of less than 10 microseconds.
- Even for accuracy of a few microseconds, a high sampling period of synchronization beacons was achieved. It should be noted that though it varies with the scheme and the level of environmental noise in the data, the sampling period in all cases was much higher than that used in short-term timescale formation of current techniques, which are typically 10 seconds, 18 seconds, 30 seconds, or 1 minute.
- The protocol suggested by us has very little messaging overhead while synchronizing all the nodes to the sender, and all these three methods preserve this property too. After the initial training, the sequential prediction only uses one sample of synch

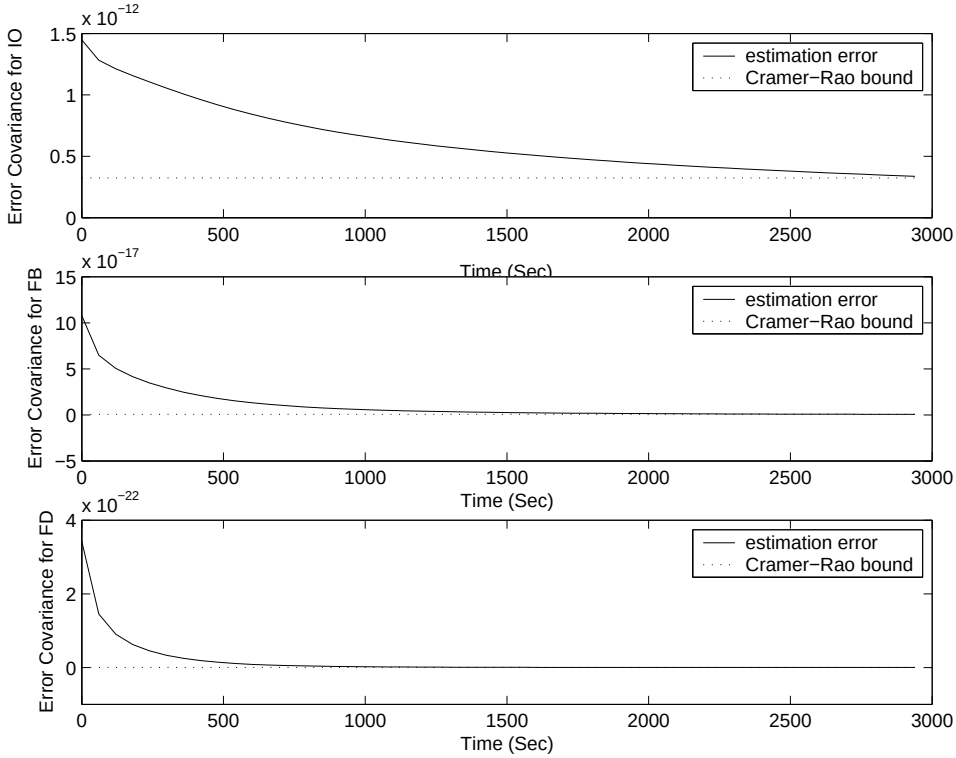

Figure 6.13. Parameter estimation errors and corresponding Cramér-Rao bounds for SLS.

beacons to update the parameters and predict next value. This window size of one keeps the overhead minimum.

- Number of training samples required for convergence are as low as 5 to 10 for both SLS and Kalman filter algorithms. Even in case of dynamic change of clock states, less than five samples are required by the Kalman filter to obtain the new estimates and track the time-offset accordingly.
- The storage requirement of SLS and Kalman filter is very small. They only need to store last estimate of parameters, and last samples of the gain and covariance matrices. The storage required for FLS is a little higher than these two methods, since it has to store last four samples of the data and the corresponding matrices,

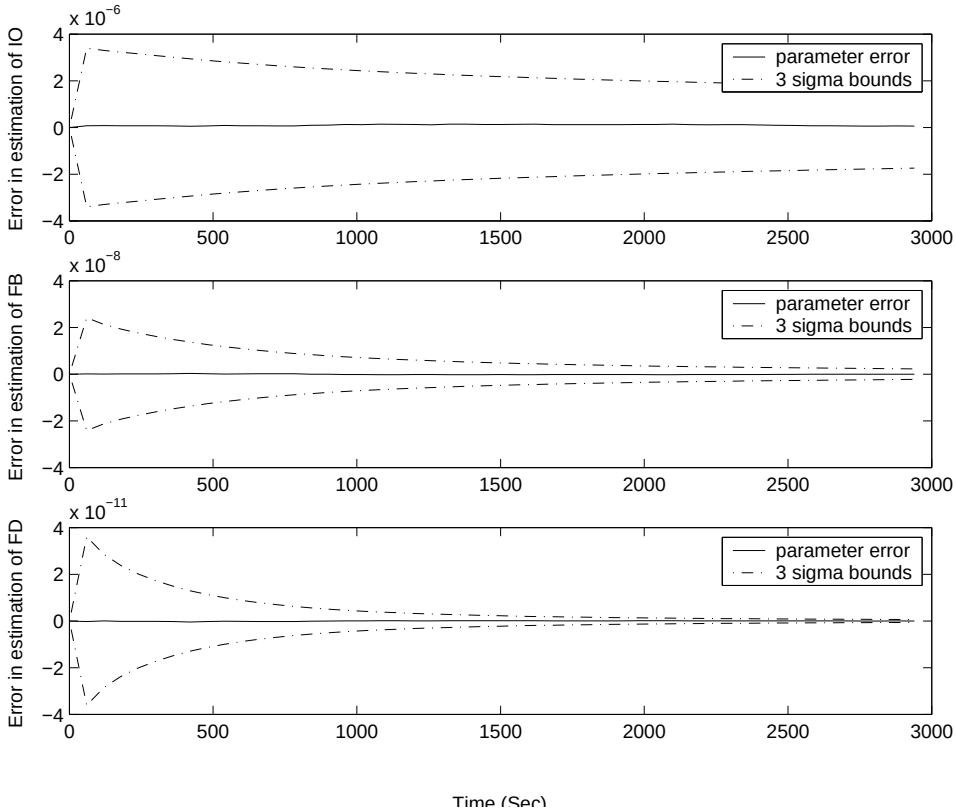

Figure 6.14. Parameter estimation errors and corresponding 3σ bounds for SLS.

and the last values of parameters of membership functions. Still this will require only for a moderate storage.

- It was observed that, all the methods converge very fast to the true value of parameters after the initial training period. Hence it can be said that though the initial offset and the clock parameters like frequency skew and drift between two nodes can vary considerably across a network, the quick convergence of these algorithms leave us some room for approximate choice of the initial conditions.
- Each of these methods give similar performance in terms of accuracy and sampling period when tested for different values of parameters.

Apart from these common goals achieved by all the three approaches, each approach shows different performance and characteristics regarding few other factors.

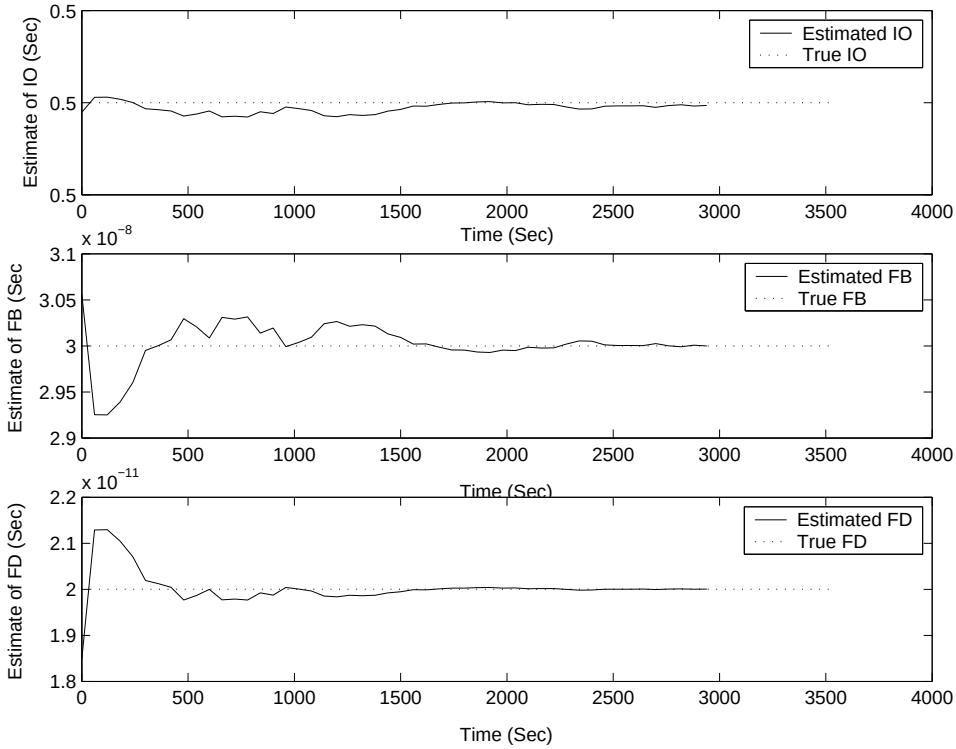

Figure 6.15. True and estimated states of clock parameters using SLS.

- SLS approach has much less complexity than Kalman filter approach and needs less training samples than FLS. Hence the computation time is also very small.

  From this, and the observed performance, we can say that SLS is good enough for indoor scenarios, where there are no sudden changes in the clock drift. In such cases, we need not go for complex schemes.

  Theoretically and even experimentally, SLS is not good for estimating dynamic clock states and predicting the offset. Still, as we can see from the graphs, SLS yields an average performance of about 10^{-11} to 10^{-12}.
- The data generated by a model similar to [18], goes very close to the real-time case. Kalman filter achieves promising results on such datasets.

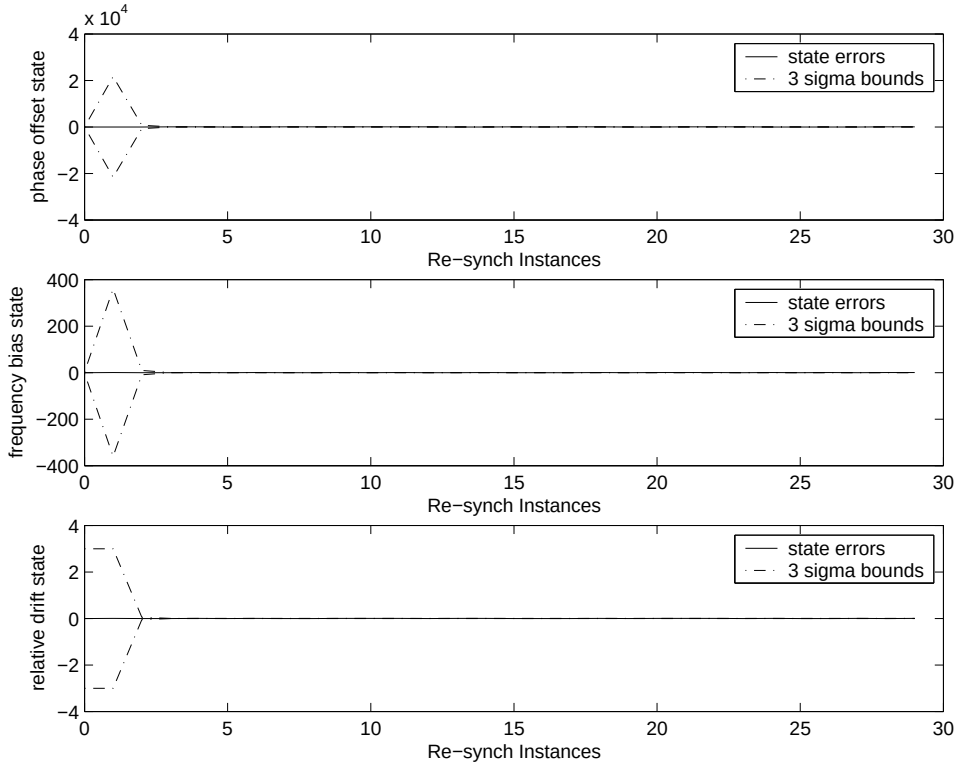

Figure 6.16. Parameter estimation errors and corresponding 3σ bounds for Kalman filter.

It also shows a remarkable performance in the case of data with highly varying clock drift and high environmental noise. This implies that this approach is well suited for an outdoor scenario.

The only hitch in this approach could be that the performance depends on the model used and our knowledge of the dynamics of the real data.

But again, for real data, increasing the window size to 2 or 3 samples can improve performance a lot in this dynamic case.

Also there is some freedom to choose the parameters such as Q, R, Φ during training period so as to get best results in the current situation. This is not possible in case of SLS.

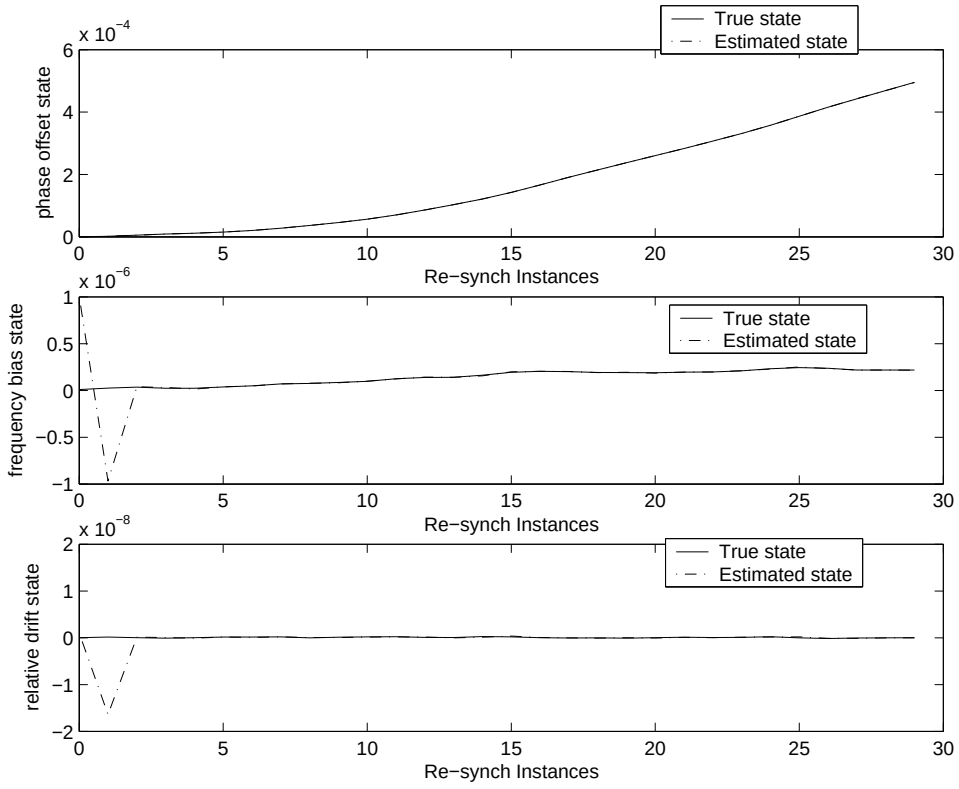

Figure 6.17. True and estimated values of clock states using Kalman filter.

- As both SLS and Kalman filter provide estimates of parameters or states of the clocks, they can be used for adjusting the node's clock. These state estimates also has an accuracy up to fourth decimal.

  Also, both the SLS and Kalman filter applied to this problem satisfy the theoretical bounds on the estimation error covariance matrices. Thus, by using this approach, we can get an idea about the range of the error in the estimates.

- FLS also gives a high accuracy even with dynamic states and in presence of environmental noise. The main advantage of FLS over the other two is in its model-free nature. This fact could be very useful in real-life situations, where the clock drift and other effects make the offset between two clocks vary in a highly random fash-

ion, i.e. when the conditions are too extreme to fit even in the realistic model of Kalman filter.

Even a simple design tracks the output very well, giving the benefits of simplicity and very less processing time.

The problem with the FLS could be that larger training period is required, like even 15 or more samples. Also batch processing does not yield good results with the dynamic data. But we have been able to reduce this problem largely, by designing a sequential forecasting FLS. While batch mode can give better results in case of an indoor-scenario, the sequential algorithm works very well for data with dynamic clock states and high environmental noise, owing to its strength of online processing.

Even FLS provides many parameters, like the number of antecedents and membership functions, means and standard deviations of membership functions, learning parameter of back-propagation algorithm, etc., to tune the FLS to provide better results for the given data.

In spite of many advantages, in some cases, FLS needs a lower sampling period between synch beacons for the error performance to be comparable. But still, it is much better than the existing schemes achieving instantaneous synchronization.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusions

Following conclusions can be drawn from this thesis:

We have designed and simulated three novel techniques as a solution for the problem of time synchronization in sensor networks: A *minimum-variance sequential least squares algorithm*, a *discrete-time Kalman filter algorithm* and a *fuzzy logic system*. The performance of each approach is shown and compared with the other approaches on basis of various factors, so that applications can have choice of selecting the algorithm which can work best in the given scenario.

All the three approaches have been successful in achieving a long-term synchronization, by forming a time-conversion scale between clocks of two nodes, which takes into account the effects of clock drift, and tries to compensate for the same. They achieve an accuracy of a few microseconds even at a high sampling interval for synchronization beacons.

The sequential least squares algorithm can be more useful in an indoor-scenario, where there is a constant and small clock drift, since even with a much simpler design, its performance matches with that of the other two techniques. For an outdoor-scenario, where the clock drift and clock offsets are highly dynamic due to the environmental effects on them, the Kalman filter and fuzzy logic algorithms still yield a very good performance because of their recursive nature. Thus these techniques make the synchronization scheme adaptive to the environmental effects, while still maintaining a good accuracy and high sampling period.

We have also developed a synchronization protocol around these techniques which provides many advantages such as: energy-efficiency, multimodal synchronization which can be always-on or post-facto, scalability, low overhead and can provide a relative or an external timescale. We also provide some protocol parameters, to make it tunable to the needs of application. These factors make our schemes applicable in a variety of applications.

7.2 Future Work

As is the case with new design techniques for any system of protocol, there is always room for improvement. Although the performance of all our approaches are more or less accurate and provide a lot of features in a synchronization scheme, we have identified some factors which could be considered to make it a good synchronization scheme on all aspects.

- *Delay correction:* Our assumptions of not considering the network delay in formation of a timescale can be valid, but they cannot be generalized for some cases. Adding terms for incorporating receiver delay and applying a technique to either calculate or avoid sender's delay can make this scheme robust to many real-life network situations, and will also get nodes more tightly synchronized with the sender.
- *Multi-hop:* Even though this synchronization protocol is scalable, it is still limited to a single-hop case. Making it achieve a multi-hop synchronization will make it suitable for a wide variety of applications. We have observed that many existing techniques for obtaining multihop synchronization can be applied on top of the schemes presented by us. To find which of these techniques will give better results is a problem for research.

- *Test on real data:* Ultimately, even though the models suggested by us are very close to real cases, and the performance of each scheme is tested on a variety of datasets, the best option would be to test their performance on real time-offset data which is collected from experiments. After such experiments, the protocol can be made ready to be implemented on sensor motes.

REFERENCES

[1] F. Zhao and L. Guibas, *Wireless Sensor Networks.* Elsevier - Morgan Kaufmann, 2004.

[2] J. Elson and K. Römer, "Wireless sensor networks: A new regime for time synchronization," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 149–154, January 2003.

[3] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review: Special issue on Physical Interface*, vol. 36, no. SI Winter 2002, pp. 147–163, 2002.

[4] R. Karp, J. Elson, D. Estrin, and S. Shenker, "Optimal and global time synchronization in sensornets," CENS Technical Report, Tech. Rep. 0012, April 2003.

[5] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks," in *In proceedings of 15th International Parallel and Distributed Processing Symposium*, April 2001, p. 186.

[6] H. Dai and R. Han, "Tsync: A lightweight bidirectional time synchronization service for wireless sensor networks," *ACM SIGMOBILE Mobile Computing and Communications Review: Special issue on Wireless Pan & Sensor Networks*, vol. 8, no. 1, pp. 125–139, January 2004.

[7] S. PalChaudhuri, A. K. Saha, and D. Johnson, "Adaptive clock synchronization in sensor networks," in *The Third International Symposium on Information Processing in Sensor Networks*, April 2004, pp. 340–348.

[8] A. swol Hu and S. D. Servetto, "Asymptotically optimal time synchronization in dense sensor networks," in *2nd ACM International Conference on Wireless Sensor Networks and Applications*, September 2003, pp. 1–10.

[9] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *1st international conference on Embedded networked sensor systems*, November 2003, pp. 138–149.

[10] J. van Greunen and J. Rabaey, "Lightweight time synchronization for sensor networks," in *2nd ACM International Conference on Wireless Sensor Networks and Applications*, September 2003, pp. 11–19.

[11] S. Ping, "Delay measurement time synchronization for wireless sensor networks," Intel Research Berkeley Lab, Tech. Rep. IRB-TR-03013, June 2003.

[12] K. Römer, "Time synchronization in ad hoc networks," in *2nd ACM international symposium on Mobile ad hoc networking & computing*, October 2001, pp. 173–182.

[13] P. Blum, L. Meier, and L. Thiele, "Improved interval-based clock synchronization in sensor networks," in *3rd International Symposium on Information Processing in Sensor Networks*, April 2004, pp. 349–358.

[14] S. Ganeriwal, D. Ganesan, M. Hansen, M. Srivastava, and D. Estrin, "Rate-adaptive time synchronization for long-lived sensor networks," in *ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, June 2005, pp. 374–375.

[15] D. Arnold, "Stochastic model estimation of network time variance," TRUETIME, Tech. Rep.

[16] J. L. Crassidis and J. L. Junkins, *Optimal Estimation of Dynamic Systems*. CRC Press, 2004.

[17] L. Galleani and P. Tavella, "On the use of the kalman filter in timescales," *IoP Journals: Metrologia: IV International Time-Scale Algorithms Symposium,*, vol. 40, no. 3, pp. 326–334, June 2003.

[18] C. A. Greenhall, "Forming stable timescales from the jonestryon kalman filter," *IoP Journals: Metrologia: IV International Time-Scale Algorithms Symposium,*, vol. 40, no. 3, pp. 335–341, June 2003.

[19] J. M. Mendel, *Uncertain Rule-Based Fuzzy Logic Systems.* Prentice Hall PTR, 2000.

[20] Q. Liang, "Ad hoc wireless network traffic-self-similarity and forecasting," *Communications Letters, IEEE*, vol. 6, no. 7, pp. 297–299, July 2002.

BIOGRAPHICAL STATEMENT

Sejal D. Raje was born in Mumbai, India, in 1980. She completed her Bachelor of Engineering in Electronics and Telecommunications from the Mumbai University in August 2002. She obtained her Master's degree from the University of Texas at Arlington in August 2005 under the Department of Electrical Engineering. Her current research interests are wireless and digital communications and networking.