Chapter 3

Time Synchronization for SRS and MRS

3.1 Introduction

Time synchronization is a critical process in any distributed system, as the information collected over a distributed network has significance only when associated with a timestamp and a location stamp. Real-time wireless control, co-operative data collection and navigation, task allocation, and task migration in a robotic network can be effectively performed only if the robots are synchronized to a common notion of time. Interestingly, swarm robotic systems and wireless sensor networks (WSNs) share several common characteristics such as use of large number of robots/nodes, limited individual capabilities of robots/nodes, distributed computing, co-operative behaviour among robots/nodes to achieve system goals, etc. Both domains can operate complementary to each other such that the swarm robots can utilize WSN for communication and co-ordination, while WSNs can take advantage of the mobility provided by swarm robots to improve its sensing range, thus forming a self-healing network. Time synchronization have been a major focus of study for more than a decade in sensornet community. Although few time synchronization protocols proposed over the last few years in the field of WSN's claim to provide an accuracy in the order of few microseconds as mentioned in Section 2.1.2, most of these protocols incur unreasonably high communication overheads (mainly due to short resynchronization interval), high convergence

time or computational complexity which make them unsuitable for practical applications. Improving the resynchronization interval, accurate time synchronization under varying environmental conditions, limiting synchronization errors within a deterministic bound in multi-hop scenarios etc are open problems of research even for static WSNs. Time synchronization error can be a within a few hundreds of milliseconds for most of the static WSN applications, whereas swarm robotic applications require higher synchronization precision (lesser error bound, in the order of few tenths or hundredths of microseconds) because mobile robots, in addition to monitoring or controlling the environment in which they are deployed, may utilize timing information for their localization, cooperative path planning, navigation, aggregation, dispersion etc. Although several works are reported in the field of routing and Medium Access Control (MAC) for mobile wireless sensor networks, or for navigation in a collaborative swarm, prior time synchronization is stated as one of the assumptions when time dependency is involved [34–36]. Time synchronization is a prerequisite for other layers of the network protocol stack such as routing and MAC and thus most of the research in mobile wireless sensor networks (MWSNs) and swarm robotics is still limited to simulation based study. With the constraints such as limited communication bandwidth, limited battery power, limited processing power, etc. time synchronization for dynamic and low cost multi-robotic and swarm robot systems is a challenging problem to solve.

In this chapter, in addition to robots in SRS, robots in low-cost MRS systems are also referred to as swarm of robots. In this chapter, the problem of time synchronization for swarm of robots, which makes use of wireless network for communication among members of the swarm is addressed. The rest of the chapter is organized as follows. Section 3.2 describes the desired characteristics of time synchronization for swarm of robots. In Section 3.3 an analysis on the errors in popular time synchronization algorithms is presented, which highlight the necessity to develop a new time synchronization technique. In Section 3.4, a novel time synchronization framework refereed as "Swarm-Sync" for dynamic networks like SRS and low-cost MRS is presented. The section also presents the experimental analysis for single, multi-hop and for dynamic environmental conditions. The section also provides a brief overview of the swarm robot designed and utilized by us for the experimental testing and validation of time synchronization protocol is presented. In Section 3.5, a fair comparison of “Swarm-Sync” framework the two major class of synchronization protocols, i.e the predictive techniques (e.g. Linear Regression (LR), Linear Prediction (LP) and Kalman Filter (KF)) and consensus based techniques (Cluster based Maximum consensus Time Synchronization

(CMTS), Clustered Consensus Time Synchronization (CCTS)) proposed for synchronization in WSNs is presented.

3.2 Time Synchronization for Swarm of Robots- Desired Characteristics

The desired characteristics of a time synchronization framework/protocol for swarm of robots is presented in this section. For scalable networks, cluster based network architecture is preferred for efficient control and data dissemination. To maximize the utilization of robots, they should be allowed to move seamlessly across clusters without necessitating resynchronization or introducing overheads in synchronization. Designing a time synchronization protocol for scalable networks like swarm of robots is challenging as the protocol should minimize the variance of local (among the members of cluster or hop) and global (across clusters or hops) synchronization errors simultaneously to support seamless mobility. Hence a topology independent time synchronization scheme is desired.

A global time-synchronization framework which synchronize the swarm of robots to the reference robot/node in a swarm network such that all robots/nodes of swarm have approximately the same time value at any instant is desired. The nodes are considered to be globally synchronized when the instantaneous offset among any two nodes i, j in the network can be represented as

$$|\theta_i(t) - \theta_j(t)| \leq \delta \tag{3.1}$$

where 'δ' is the synchronization error bound. An ideal synchronization protocol should free of cumulative error over multiple hops or clusters. The synchronization error should be bounded for a given resynchronization interval, irrespective of the distance of the robot from the reference node, or number of hops in the network. Although perfect synchronization is difficult to achieve in practice, a deterministic and bounded global synchronization error will facilitate easier design of other layers of protocol such as localization of robots, medium access control and routing.

Swarm robots wirelessly communicate with each other for several functionalities like task allocation, path planning, data routing, localization, etc. Time synchronization is a critical service or a system maintenance task which has critical influence on the other layers of the protocol stack. On

the other hand, being a background task, the synchronization technique should be computationally efficient and should incur lesser communication overhead ensuring the availability of hardware and communication bandwidth for other foreground tasks. Taking this into consideration, the resynchronization interval i.e, the interval at which the nodes should communicate to derive information about the reference node should be maximized- preferably to an order of atleast few minutes. The convergence time i.e, the interval between the starting of time synchronization by a node to the time taken to achieve the required level of global synchronization accuracy should be minimized. Likewise, the execution time of the synchronization protocol should be minimized. A tunable protocol, which provides on-demand synchronization or continuous synchronization depending upon the application requirements is desired to optimize energy efficiency. The novel time synchronization framework "Swarm-Sync" presented in this chapter is designed to cater to these desired characteristics.

3.3 Sources of Errors in Time Synchronization Algorithms- An Analysis

Timestamping is a critical step of the time synchronization process. MAC level timestamping can eliminate the send time, access time and receiver/reception time uncertainties associated with timestamping over a wireless medium [48]. Hence, the development of radio modules with MAC level timestamping gave impetus to the development of multitude of time synchronization protocols in the last decade. The radio modules which support MAC level timestamping (e.g. CC1100, CC2420 etc) assert one of their output pins when the ‘sync’ word (or Start of Frame Delimiter specified in 802.15.4) is transmitted or received. The radio module deasserts the output pin at the transmission or reception of End of Packet (EOP). The output pin of radio module is generally interfaced to an interrupt input pin of the node’s microcontroller and the RTC timestamp is recorded as a part of interrupt handling. The transmitter and receiver timestamp at a common reference point, e.g. transmission/reception of ‘sync’ or ‘EOP’ and the time difference between the pair of timestamps will provide the approximate instantaneous offset between the transmitter and receiver nodes. The nodes are then time synchronized based on the timestamps. Although MAC level timestamping will eliminate the send time, access time and receiver/reception time uncertainities,

the same cannot eliminate the error due to propagation delay, interrupt latency and jitter [48]. Various protocols synchronize the nodes by applying different synchronization techniques on the timestamps.

In this section, an analysis on the sources of synchronization errors in the different techniques utilized by prominent synchronization protocols which employ MAC level timestamping is presented. The probable reasons for their reduced synchronization accuracy resulting in smaller resynchronization intervals are identified. The presented analysis in this section also justifies the requirement of developing a new time synchronization technique. The prominent synchronization protocols are categorized into two, i.e the prediction based protocols and consensus based protocols and the sources of error in these two categories are presented in this section.

Category 1-Prediction Based Protocols: Flooding Time Synchronization Protocol (FTSP) [48], PulseSync [33], Adaptive Linear Prediction Synchronization (ALPS) [49] and Interacting Multimodel (IMM) [50] predict the reference clock value based on the previous measurements of reference and slave node timestamps taken at uniform intervals of time called as resynchronization interval. Our analysis on the possible errors in timestamping as given below, indicate that when MAC timestamping is done periodically, the uncertainties like send time, access time, receive time are reintroduced into the timestamp measurements resulting in recording of timestamps at non-uniform intervals of time. This will adversely affect the synchronization accuracy necessitating shorter resynchronization intervals.

Figure 3.1 shows a generic scheme for maintaining the time in a node and the possible source of uncertainties and delays in wireless timestamping. As mentioned in Section 2.1.1, notion of time in a node can be maintained in software or hardware. A Real Time Clock (RTC) peripheral which provides a granularity of microseconds, which can be both set and read by software is not available in most of the micro-controllers. Hence a software based RTC can be maintained in the format $hours : minutes : seconds : milliseconds : microseconds$. As shown in Figure 3.1, a timer peripheral (T_RTC) of the node generates an interrupt every microsecond based on which the software RTC is updated. To transmit the messages periodically at T_{resync}, another timer peripheral (T_resync in Figure 3.1) can be maintained which raises interrupt when the configured resynchronization period elapses. Following are the delays and uncertainties in timestamps while attempting to perform periodic timestamping at an interval of T_{resync}.

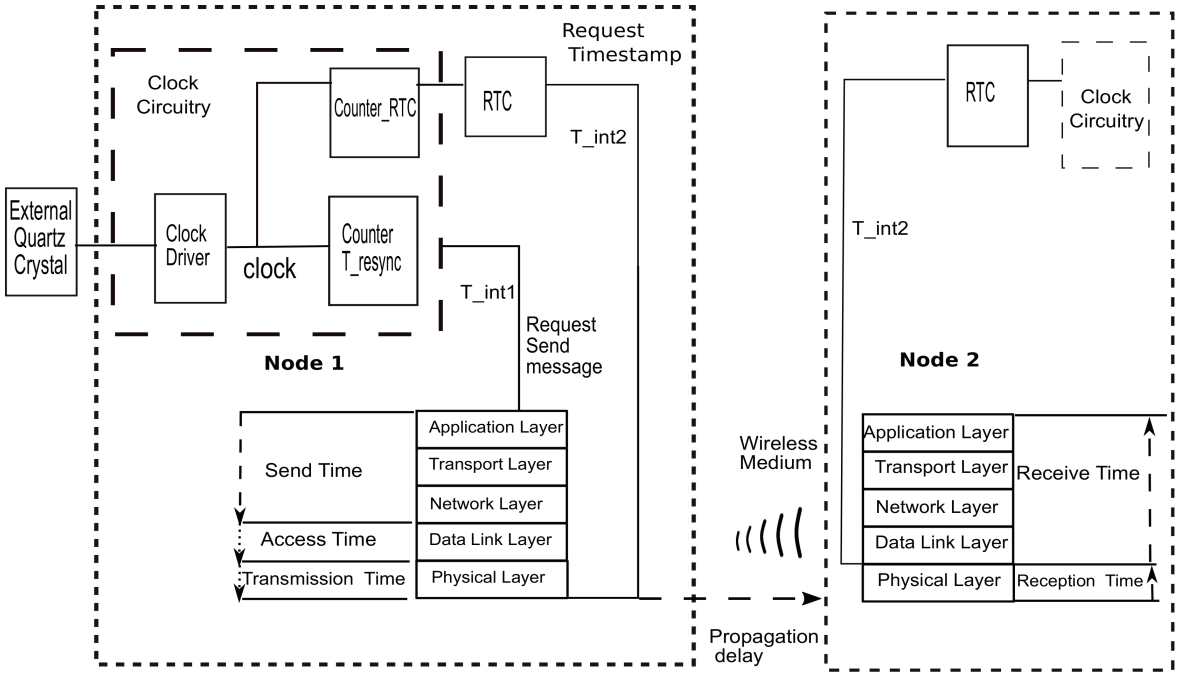

Figure 3.1. Uncertainties and delays in wireless timestamping.

1. Interrupt handling delay ($T_{\_int1}$): An interrupt is raised at T_{resync} interval by the T_resync timer. As a part of interrupt handling, a request to transmit a message is sent to the application layer. The jitter in interrupt latency can be in the order of several clock cycles depending on the priority of the timer interrupt, current task performed by robot, level of task switching etc. Mobile robots perform frequent localization, image processing, wireless message exchanges for co-ordinated navigation, etc. In a multitasking system susceptible to priority inversions, the interrupt jitter (variation in interrupt latency) is indeterministic and thus emerge as a major source of error in timestamping.

2. Send time (T_{send}): Time incurred in constructing the packet at the application layer to the time the packet takes to reach the data link/MAC layer (Figure 3.1). This delay can be highly variable in a multitasking robot.

3. Access time (T_{access}): The time for which the packets reaching data link layer will have to wait before it gets transmitted through the wireless channel. Delay varies according to the type of medium access protocol used in the application. The medium access protocols based on channel sensing can lead to an unpredictable delay in transmission.

4. Transmission time (T_{txt}): Time incurred to transmit a packet bitwise at the physical layer over the wireless medium. This delay is deterministic and can be estimated if the transmission packet size is known.

5. Propagation time (T_{prop}): Time taken ($< 1\mu$s for distances less than 300m) by the packet to traverse through the wireless media from sender to the receiver.

6. Reception Time (T_{recp}): Deterministic time incurred in receiving the bits and passing it to the data link layer.

7. Interrupt handling delay ($T_{\_int2}$): The output pin of radio is asserted on transmission/reception of 'sync' packet and deasserted at the transmission/reception of 'EOP'. At the transmission/reception of 'sync' or 'EOP', interrupt is raised to the micro-controller and RTC timestamps are recorded as a part of interrupt handler.

   MAC level timestamping for time synchronization was proposed for the first time in FTSP protocol. FTSP protocol was tested on nodes which utilize CC2420 radio module which allows timestamping at every byte boundary in-addition to the timestamping at 'sync' or 'EOP'. FTSP uses a single message/node for synchronization, but the timestamps are taken at each byte boundary as they are transmitted or received to minimize the timestamping error due to the interrupt jitter. The final timestamp in FTSP is obtained from the average of the normalized timestamps. However, even with 6 timestamps, the time-stamping precision was improved to only $4.5\mu s$ on mica2 nodes [48]. Byte level timestamping feature is not available in all radio modules. In microcontroller based nodes, skew is in the order of few μs/s and hence it can be noted that with interrupt based one-way MAC timestamping, the error due to interrupt jitter can be comparable to the relative skew of nodes.

8. Receive time (T_{rec}): Time incurred by the packet to reach the application layer in the receiver.

Even though the periodic timestamping is attempted from the application layer at an interval of T_{resync}, the actual message transmission and subsequent timestamping at transmitter will be affected by the following delays and uncertainties. The delay (T_{tx_delay}) and uncertainty ($T_{tx\_uncertainty}$) in timestamping at transmitter is as follows.

$$T_{tx\_delay} = T_{\_int1} + T_{send} + T_{access} + T_{txt} + T_{\_int2} \tag{3.2}$$

$$T_{tx\_uncertainty} = T\_int1 + T_{send} + T_{access} + T\_int2 \tag{3.3}$$

Hence period at which timestamps are recorded at transmitter will be

$$T_{actual} = T_{resync} + T_{tx\_delay} \tag{3.4}$$

Similarly the delay (T_{rx_delay}) and uncertainty ($T_{rx\_uncertainty}$) in timestamping at receiver is

$$T_{rx\_delay} = T_{recp} + T\_int2 + t_{rec} \tag{3.5}$$

$$T_{rx\_uncertainty} = T\_int2 + t_{rec} \tag{3.6}$$

As explained in Section 2.2.1, Linear Regression (LR), Linear Prediction (LP) and Kalman Filter (KF) based protocols such as FTSP, PulseSync, ALPS, IMM, etc., requires time stamping at fixed intervals for synchronization [33, 48–50]. In the presence of uncertainties as mentioned in equations (3.3) and (3.6) timestamping interval will vary and the linear model of clock may not hold true leading to synchronization errors and subsequent reduction in resynchronization interval in LR, LP and KF based predictions even for single hop synchronization.

Category 2-Consensus Based Protocols: Consensus based synchronization approaches are gaining popularity owing to their distributed nature and computational lightness. In Average Time Sync (ATS) and Maximum Time Synchronization (MTS) protocols, each node which receives broadcast synchronization messages from its neighbour nodes will attempt to adjust its logical clock with the average and maximum logical clock respectively of its neighbour nodes [51, 52]. Clustered Consensus Time Synchronization (CCTS) and Cluster based Maximum consensus Time Synchronization (CMTS) protocols proposed for scalable and clustered networks are based on average and maximum consensus respectively [53, 54]. In CCTS and CMTS, the cluster heads initiate synchronization with a broadcast message. The cluster members record their timestamps on the message reception and transmit the timestamp back to the cluster head. In CMTS, cluster head selects the node with maximum logical clock among its cluster members based on the received timestamps and then adjusts its logical clock to the node with the maximum logical clock. Subsequently, the cluster head requests the cluster members to adjust its logical clock with the cluster head. Hence, other nodes in the network can be treated as two hops away from the node with the highest clock, except in the case where cluster head itself has the maximum clock.

Suppose, in a network which utilize CMTS for synchronization, node 'A' and 'B' are the cluster members and node 'R' is the cluster head. If 'A' is the node with maximum clock, then node 'R' first synchronizes its clock to 'A', after which 'B' adjusts its clock by synchronizing to 'R'. Error in timestamping can lead to error in time offset and skew measurements represented as $\Delta_{\theta,i}$ and $\Delta_{\alpha,i}$ respectively for a node 'i'. To keep the analysis simple, it is assumed that the error in measurements of skew and offset are independent of each other and if the synchronization between node 'R' and 'A' occurs shortly before synchronization of node 'B' with node 'R', then the time offset between node 'A' and 'B' because of skew can be ignored. Therefore, the error in offset measurement between node 'A' and 'B's clock is $(\Delta_{\theta,A} + \Delta_{\theta,B})$. Suppose the frequency of node 'A's clock is $(\alpha_A + \Delta_{\alpha,A})$ times node 'R's clock and node 'R's clock is $(\alpha_B + \Delta_{\alpha,B})$ times node 'B's clock, then the clock frequency of node 'B' can be expressed as $((\alpha_R + \Delta_{\alpha,R})(\alpha_B + \Delta_{\alpha,B}))^{-1}$ times node 'A's clock. The $\Delta_{\alpha,A}$ or $\Delta_{\alpha,B}$ can be positive or negative value. In general, for a node 'n' hops away from the node with maximum clock, the measured time offset θ_n and skew α_n can be expressed as follows:

$$\theta_n = \sum_{i=1}^{n} (\theta_i + \Delta_{\theta,i}) \tag{3.7}$$

$$\alpha_n = \Pi_{i=1}^{n} (\alpha_i + \Delta_{\alpha,i})^{-1} \tag{3.8}$$

As per equation (3.8), error in skew estimation multiplies in case of consensus algorithm with each iteration. The error will thus substantially increase after inter-cluster synchronization. Cluster head rotation is recommended even for clustered static networks for improving energy efficiency and increasing life time of networks. In mobile networks, new clusters, cluster heads and cluster members may emerge based on the dispersion of the network in the field of interest. Hence the skew measurement error as in equation (3.8) will lead to substantial cumulative error in a mobile network. Similar analysis can be extended for CCTS, however a detailed analysis is not provided in this thesis as [54] suggest that, CMTS is superior to CCTS in terms of communication overhead and convergence time. The analysis presented in this section indicate that new techniques are to be developed for time synchronization.

3.4 Swarm-Sync Framework

The novel time synchronization framework "Swarm-Sync", presented in this section, aims to synchronize the swarm of robots such that all robots/nodes of swarm represent approximately the same time at any instant, irrespective of their distance from the reference robot/node. Nodes in a network can be globally synchronized using the framework, such that the instantaneous time offset between any two nodes in the network is limited to a deterministic bound, 'δ' for a given resynchronization interval. The key principles adopted for the design of Swarm-Sync framework are described as follows.

1. The framework should synchronize each robot/node in the network such that the synchronization error (T_{error}) between the reference node and any given node is $\leq \delta\ \mu s$ for a given resynchronization interval, irrespective of the number of robots in the system. This approach will ensure that T_{error} between any two nodes in the network is $\leq \delta \mu s$, thus minimizing both local and global synchronization error simultaneously.

2. The robots should be allowed to move seamlessly in the deployment area across different hops or clusters without necessitating resynchronization. Hence, the dependency of a node on cluster head (for clustered architecture) or intermediate reference node (for a multi-hop network) for synchronization should be minimal. This approach will reduce the cumulative synchronization error in a typical multi-hop/clustered network. Since cumulatice synchronization errors over multiple hops/clusters can lead to large variance in clock values of nodes across the network, reduction of cumulative error is one of our major design focus.

The framework is designed to be scalable such that the addition or removal of nodes in the system will not lead to synchronization overheads or failures affecting the whole network. The framework is designed to be tunable, to support demand based or periodic synchronization to suit different application scenarios.

From Section 2.1.2, it can be understood that the synchronization protocols like TPSN and RBS perform frequent time offset compensation and do not implement frequency offset compensation. The protocols which utilize LR, LP and KF techniques predict the reference node clock from timestamps obtained from reference and slave nodes at an interval of resynchronization interval (in

the order of few seconds) without performing initial time offset compensation. Our experimental analysis on the existing time synchronization protocols as is explained in Section 3.5, indicate that the resynchronization interval can be significantly improved if initial time offset compensation is performed on the nodes. However, the techniques for time offset compensation available in literature utilize pair-wise, two-way message exchanges leading to higher convergence time and hence are not suitable for scalable and dynamic networks. Swarm-Sync framework recommend time offset compensation followed by frequency offset compensation at resynchronization intervals for time synchronization.

The Swarm-Sync framework completely avoid the use of two-way message exchanges and propose low-complexity one-way message based, time and frequency offset compensation. Swarm-Sync framework includes two components, one for the implementation of time offset compensation and other for the implementation of frequency offset compensation. The framework propose time synchronization in two phases-

1. Time offset compensation and
2. Relative skew fingerprinting based frequency offset compensation.

Each phase is implemented by the corresponding component of the framework. Frequency offset compensation is performed in two steps- frequency offset calibration (also referred as relative skew calibration, which is an optional step) and reference clock estimation. Nodes are required to communicate for synchronization, only during time offset compensation and frequency offset calibration. Time offset compensation and frequency offset calibration of slave nodes are achieved by transmission of a control frame followed by data frames from the reference node. The common format of control and data frames is as shown in Figure 3.2. 'Message_type' bits of the command field indicate the type or purpose of the message. Reference node broadcast a control frame to slave nodes with the corresponding 'Message_type' requesting them to participate in time offset compensation or frequency offset calibration process. The slave nodes on reception of control frame may switch to the corresponding operational mode depending upon the control message received. The different message types, operational modes and number of data frames required for time offset compensation and frequency offset calibration are summarized in Table 3.1. After time offset compensation, the nodes will remain in frequency offset compensation mode until the next resynchronization interval. The nodes may temporarily switch to frequency offset calibration

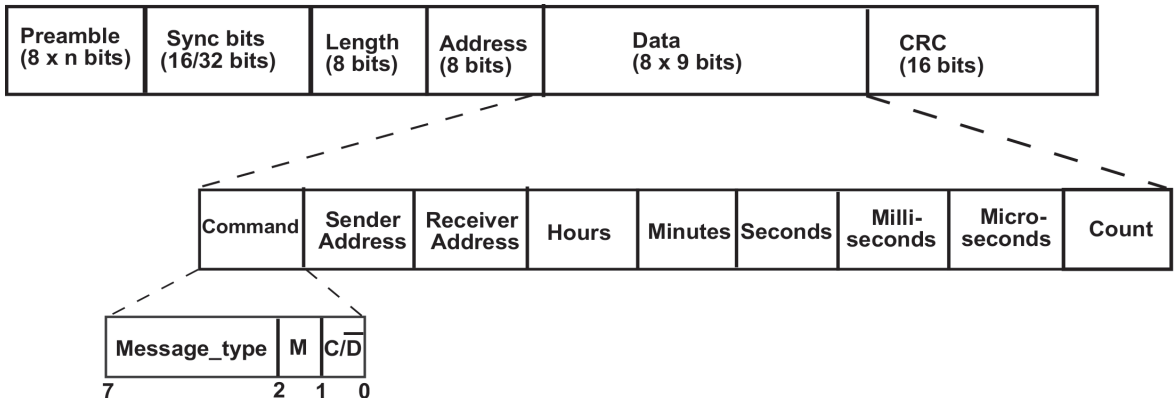

Figure 3.2. CC2500 packet and payload format.

mode after time offset compensation, if the slave node receives a control message to participate in frequency offset calibration.

Section 3.4 is organized as follows. The hardware utilized in this work for testing Swarm-Sync framework is described in Section 3.4.1. Sections 3.4.2 and 3.4.3 provide the description of time offset compensation and frequency offset compensation respectively. The performance analysis of the Swarm-Sync framework in indoor as well as outdoor environments is also presented in this section.

3.4.1 Hardware Architecture

Wheeled, differential drive, miniature robotic platform using Commercial off-the-shelf products (COTS) suitable for swarm robotic applications was designed, and is utilized as cost effective solution, which facilitates validation of algorithms/protocols pertaining to distributed wireless control of swarm of robots, navigation, time synchronization and localization techniques mentioned in this thesis. The robot architecture is modularized into two. The modules- 1) Mobility module

Table 3.1. Swarm-Sync framework- Summary of message types and operational modes

'Message_type' of Control frame	Operational Mode	Number of Data frames
$time\_offset\_sync$	Time offset compensation mode	3
$frequency\_offset\_sync$	Frequency offset calibration mode	9
-	Frequency offset compensation mode	-

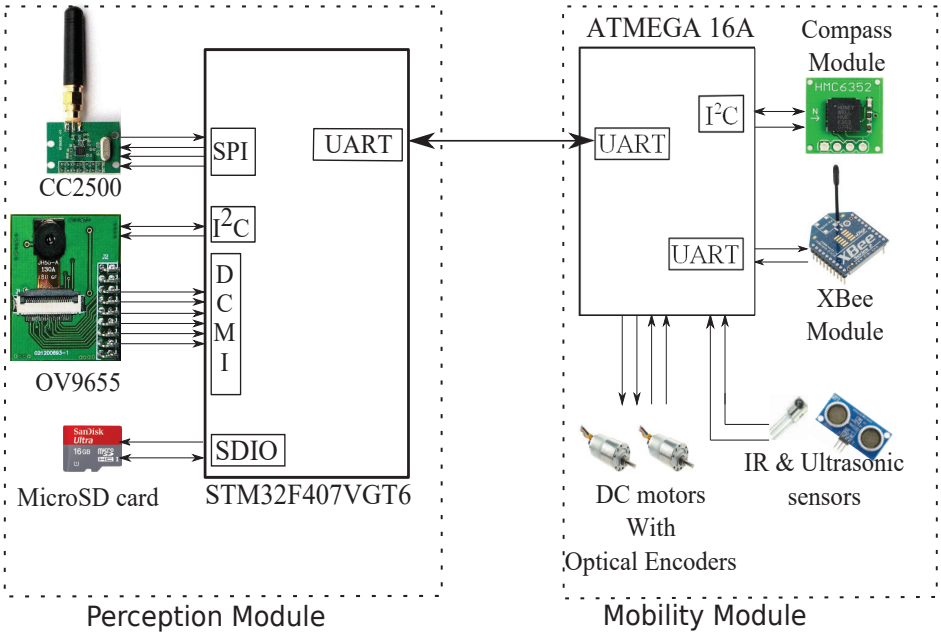

Figure 3.3. Overview of Robot Hardware Architecure Overview.

and 2) Perception module are independently controlled by different microcontrollers as shown in Figure 3.3. The mobility module (controlled by ATMEGA16A, 8-bit microcontroller) houses the required sensors for obstacle detection, motor driver and control circuitry for robot navigation. This module manages the motion control of the robot. Two DC geared motors in differential drive configuration along with a castor wheel for support is used for navigation. The module has several on board sensors, i.e HMC6532 Compass module with 0.5 degree resolution, ultrasonic sensor for obstacle detection (Maxbotix sensor with range 254 inches and resolution of 1 inch), four infrared proximity sensors-one on each side of robot for near obstacle detection for facilitating navigation of robot. Optical encoder (MOC7811) on each wheel for precise control of wheel (upto 12.92 mm and 12.85 degrees) for facilitating navigation of robot.

The perception module is designed with the following COTS components. It has STM32F4 discovery board (with STM32F407VGT6, 32-bit microcontroller based on ARM Cortex M4 architecture as processing unit), CC2500 radio (as communication unit) and OV9655 (camera module). Perception module fuses the wireless and processed image information to provide supplementary path planning inputs to mobility module. The perception and mobility modules communicate with each other using Universal Asynchronous Receiver/Transmitter (UART) protocol. Modularity in the

design ensures that the navigation activities and networking activities can be performed simultaneously, co-operatively or independent of each other. Any specific hardware for localization or time synchronization or wireless control of robots will be introduced to the perception module. Hence if the time synchronization, localization or task allocation has to be implemented for a different robot or terrain, the perception module can be resused as it is, however the mobility module will change according to the requirement of the application. This also ensures that the protocol stack developed for SRS including time synchronization can be used for different types of robot in land (walking, crawling or wheeled robots) or air without modification to the perception module. Hardware changes are required only in the mobility module to match different terrains. The robot is referred as BSwarm robot Detailed explanation on the modularized design of the BSwarm robot and choice of components is mentioned in [84] and brief description of the same is provided in Appendix A. This robot is henceforth referred as node in this thesis.

Time synchronization is implemented on perception module as communication with other robots in the system is achieved via this module. Perception module maintains the time of the robot/node. The STM32F407VGT6 microcontroller is clocked by a 8MHz quartz crystal (HC-49S-C20QSA) and using the internal PLL, the CPU clock is configured to 100MHz [85]. The crystal exhibits a frequency stability of +/-20ppm over a temperature range of -10°C to +70°C. Microcontrollers including STM32F407VGT6 do not provide a RTC peripheral with a microsecond level granularity which can be both read and set by software. Hence a software RTC with a granularity of $1\mu s$ is maintained using 'Systick', a precise timer peripheral supported by ARM Cortex M4 architecture. Systick is configured as the highest priority interrupt in perception module. The format **of** RTC is $hours : minutes : seconds : milliseconds : microseconds$.

The perception module utilizes CC2500 radio module for communication among other nodes and a Xbee S1 class radio module for transferring the timestamp values and any debug information recorded on the nodes to a base station laptop which maintains the data log. CC2500 radio module support timestamping via two of its output pins GD01 and GD02 which can be configured for various functionalities as mentioned in [86]. For our application, CC2500 is configured such that GD02 pin is asserted when 'sync' word is sent/received by the radio module and de-asserted at the transmission or reception of EOP. GD01 pin is asserted when the receive FIFO of radio module is filled with data above a configurable threshold. GD01 gets de-asserted when the receive FIFO is emptied below the threshold i.e. when data in FIFO is read by the microcontroller. The work

presented in this chapter can be replicated on any node hardware by including a radio module which supports MAC level timestamping.

For experimental validation of the framework, 6 identical robots/nodes, $N_1 - N_6$ are utilized. N_1 is the reference node. Nodes which are supposed to be synchronized with the reference node are henceforth referred as slave nodes. Nodes $N_2 - N_5$ are slave nodes and N_6 is the poller. Due to limited availability of robots the experiments are conducted on 6 nodes, however the framework is scalable in terms of size of the network.

3.4.2 Swarm-Sync Framework - Time Offset Compensation

An accurate one-way time offset compensation scheme, which account for the deterministic delays (transmission time, propagation time, reception time) and eliminates the uncertainties introduced by the non-deterministic delays (send time, access time, receive time and interrupt delay) during wireless timestamping is presented in this section. Time synchronization is desired to be performed via one-way messages instead of pairwise message exchanges between reference and slave nodes in a scalable network. Although one-way message based frequency offset or skew compensation is reported in literature, researchers have reported [32, 87] that one-way time offset compensation is not feasible, based on their observation that time offset and propagation delay cannot be distinguished with one-way messages.

An accurate one-way time offset compensation scheme is presented in this section and through experimental analysis it is established that through one-way messages, time offset can be accurately measured and compensated. Proposed time offset compensation scheme can be utilized for periodic or demand based time synchronization depending on the application requirements. Time offset compensation of all nodes at a single hop distance is achieved by transmission of a single control frame followed by three broadcast data (tsync) frames from the reference node. Time offset compensation is intiated by the broadcast transmission of a control frame by the reference node. The 'Message_type' field during the control and data frame transmissions for time offset compensation is configured as "$time\_offset\_sync$" by the reference node. 'C/$\overline{D}$' bit of command field indicate whether the transmitted frame is a control or data frame (C/$\overline{D}$=1 for control frame and C/$\overline{D}$=0 for data frame). The mandatory or 'M' bit of the command field is 'set', if the operation corresponding to control frame is enforced on the slave nodes by the reference node. If 'M' bit of

command field is 'reset' the participation of slave node in the time offset compensation is optional. When the reference node initiates an event based time synchronization, (eg: resynchronization to compensate for drastic topological or environmental changes) the 'M' bit is 'set' to enforce resynchronization on slave nodes and for periodic resynchronization, the 'M' bit is 'reset'.

After transmission of the control frame, the reference node along with the slave nodes which receive the "$time\_offset\_sync$" control frame switch to 'time offset compensation mode' to particpate in time offset compensation process by receiving data frames. If 'M' bit of command field is set, participation of slave nodes in time offset compensation is mandatory whereas when 'M' bit is reset, then the participation of slave nodes in time offset compensation is optional. After switching to 'time offset compensation mode' the nodes disable all maskable interrupts including systick for perception module, disable the systick timer, reset RTC and poll for any activity on GD02 pin. GD01 and GD02 pins of the radio modules of all nodes are configured as mentioned in Section 3.4.1. After the transmission of "$time\_offset\_sync$" control message, the reference node transmits three 'tsync' messages (Figure 3.4a). The 'message_type' field of the tsync payload is configured as "$time\_offset\_sync$" and C/$\overline{D}$ bit is 'reset' by the reference node indicating that the broadcast message is a data frame. Reference node will set 'Count' field to '1' for the first 'tsync' transmission and 'Count' will be incremented on every 'tsync' transmission. If the 'Sender address' field received during the "$time\_offset\_sync$" control and data frames do not match then the 'tsync' message is ignored by the slave node. For broadcast transmissions, the 'Receiver address' field is set to 'zero' by the transmitter. Figure 3.4.b depicts the timestamp trace captured on a logical analyzer indicating GD02 signals of reference node N_1, slave node N_2 and GD01 signal of N_2 indicating the three 'tsync' message transmission and reception during time offset

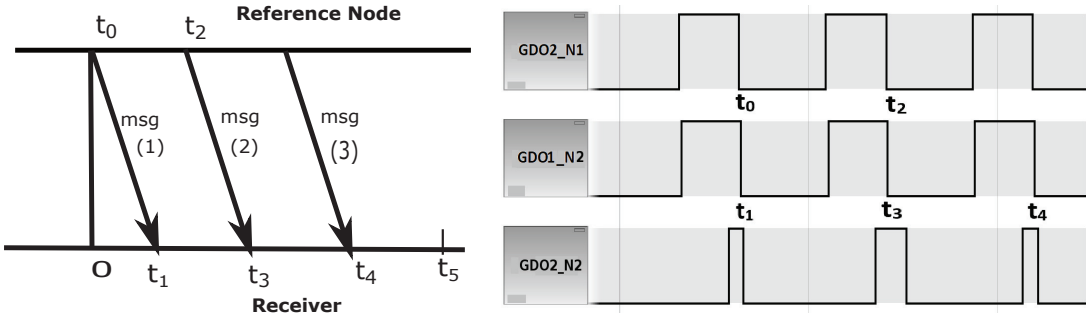

Figure 3.4. a) 'tsync' messages and associated timestamps during time offset compensation. b) GD01, GD02 signals captured on logic analyzer during tsync message transmissions ($GD02\_N1$) and receptions ($GD01\_N2$, $GD02\_N2$).

compensation.

The timestamping of data frames during time offset compensation is performed as follows. On the transmission of first 'tsync' message, timestamp t_0 is recorded by the reference node at the falling edge of GD02 (Figure 3.4b). Falling edge of GD02 during transmission, signifies EOP_transmission, i.e. the time at which complete packet is transmitted by the radio. The systick timer of slave node is started on falling edge of GD02 during the first 'tsync' message reception, effectively starting the RTC. Falling edge of GD02 during reception signifies EOP_reception, i.e. the time at which complete packet is received by the slave's radio module. t_0 timestamp is broadcast to slave nodes during the second 'tsync' message transmission and t_2 (EOP_transmission) and t_3 (EOP_reception) timestamps are recorded at reference and slave node respectively. t_2 is broadcast to slave nodes during third 'tsync' message. The timestamps t_0 and t_2 recorded by the reference node in the format '$hours : minutes : seconds : milliseconds : microseconds$' is incorporated in the payload of 'tsync' messages as indicated in Figure 3.2.

The slave nodes estimate their offset with reference node using the timestamps as follows. From Figure 3.4.a it can be observed that

$$t_0 + d = t_1 - \phi; \tag{3.9}$$

$$t_2 + d = t_3; \tag{3.10}$$

where 'd' is the delay comprising of propagation delay between the nodes and reception delay of the slave node. 'ϕ' is the deterministic time taken to start the systick timer. From equation (3.9) and (3.10),

$$t_0 + t_2 + 2d = t_1 + t_3 - \phi; \tag{3.11}$$

As slave node's RTC is reset as soon as the node is in time offset compensation mode and RTC is started only on the reception of the first 'tsync' message, $t_1 = 0$. From equation (3.11), 'd' can be expressed as

$$d = \frac{t_3 - (t_2 - t_0) - \phi}{2} \tag{3.12}$$

where $t_0 + \phi + d$ is the time offset. After determining initial offset, RTC of slave node can be adjusted to the reference node as

$$\hat{t_5} = t_5 + d + t_0 + \phi; \tag{3.13}$$

where t_5 is the current time of the slave and $\hat{t}_5$ is the time offset compensated value corresponding to t_5. Thus, each slave node can update its RTC based on its measured offset with the reference node.

One of the key benefits of the the proposed one-way time offset compensation is that the offset compensation of all nodes at a single hop distance can be completed with only 4 broadcast messages. Hence, the communication overhead for time offset compensation is independent of the number of nodes in the system, unlike the two-way message exchange based time offset compensation schemes where the number of messages required is proportional to the number of nodes. Two-way message exchange based offset compensation cause "inter-sync" error leading to variance in time values among nodes even within the same hop. The proposed offset compensation scheme is accurate, as the possible transmitter and receiver uncertainties in wireless timestamping during time offset compensation as described in equations (3.3) and (3.6) is completely eliminated as described below. The major source of error in MAC level time stamping is the jitter in interrupt latency. After receiving the broadcast "$time\_offset\_sync$" control message, timestamping is performed by polling rather than interrupt mechanism to eliminate the uncertainties introduced by interrupt jitter in transmitter and receiver timestamps. Since MAC level timestamping is implemented, send time, access time and receive time will not affect timestamping accuracy. The propagation delay and the reception delay is estimated and accounted for in the offset compensation as explained in equation (3.13). Offset compensation is completed within few milliseconds, hence disabling maskable interrupts and task switching in perception module will not affect the performance of the network, whereas the same along with MAC level timestamping ensures accurate offset compensation. Modular design of robot ensures that the robot can perform navigation autonomously under the control of mobility module while the perception module performs the offset compensation.

3.4.2.1 Experimental Validation of Time Offset Compensation

To validate time offset compensation, nodes N_1-N_5 were deployed at one hop distance from each other. The nodes were switched-on one by one after random intervals of time, varying from few seconds to 10 minutes to introduce initial offset among the nodes. Poller node N_6 was programmed to periodically transmit wireless polling messages approximately at an interval of 1.5s. Time offset compensation was performed on nodes as mentioned in Section 3.4.2. Nodes N_1-N_5 were

configured to record their RTC timestamps on the rising edges of GD02 (on reception of polling messages) after time offset compensation. The timestamps from nodes were then forwarded to the basestation using Xbee network for logging of data. The timestamp log of the reference node and one of the slave node is shown in Figure 3.5. It can be observed from Figure 3.5, that the nodes were set to a common time after offset compensation, however their time drifts apart due to inherent skew of the nodes. The proposed offset compensation scheme was tested exhaustively on nodes under the following conditions, 1) reference node and slave nodes stationary, 2) reference node N_1 stationary and $N_2 - N_5$ moving randomly at velocities varying between 50-80cm/sec, 3) reference and slave nodes moving at velocities varying between 50-80cm/sec.

Experiment was repeated 45 times (15 trials under each condition) for validating the proposed time offset compensation scheme. For 82% of the trials, all nodes in the network were synchronized within +/- 2μs, for 14% of trials, nodes were synchronized within +/-3μs and the worst-case time offset measured was +/- 4μs. It was observed that mobility has negligible effect on timestamping accuracy for node speeds ranging from 50-80cm/sec. On careful analysis of timestamps using logical analyzer, it was observed that the reported time offset of 2-4μs (based on the first timestamps captured from the nodes after time offset compensation) was not due to the inaccuracy in time offset compensation. The polling of nodes was done at an interval of 1.5 seconds and hence the nodes exhibit skew even before the first timestamps were recorded after time offset compensation. The worse case error was reported for node N_4, which exhibit maximum relative skew among the set of nodes. The worst case error reported for nodes N_3 and N_5 was +/- 2μs. The results validate that the proposed scheme can achieve precise time offset compensation. From Figure 3.5, it can also be observed that as time elapses, the time offset between nodes increase due to

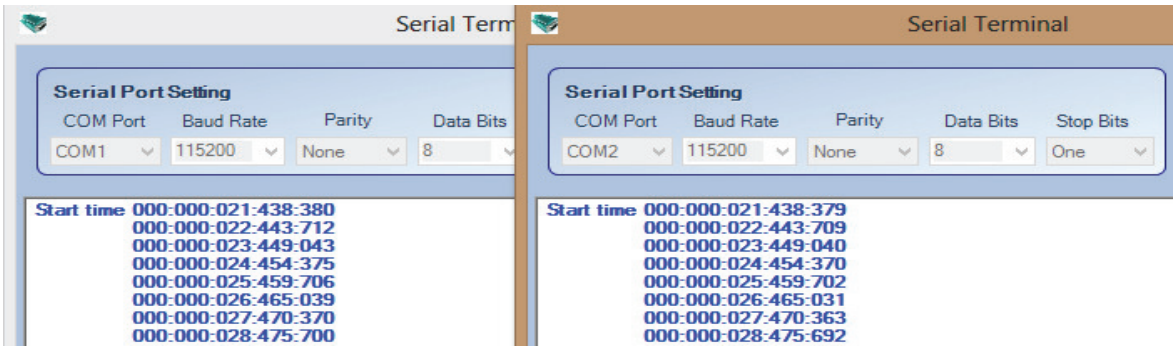

Figure 3.5. Reference node (COM2) and slave node (COM1) timestamps recorded at base station after time offset compensation.

frequency offset (skew) among nodes, thus necessiating implementation of appropriate frequency offset compensation mechanism. The proposed frequency offset compensation is elaborated in Section 3.4.3.

3.4.3 Swarm-Sync framework - Frequency Offset Compensation

The proposed frequency offset compensation is a low complexity scheme based on relative skew fingerprinting technique. Clock skew fingerprinting based sybil and wormhole attack detection has been reported in the field of computer networks [88]. [89]demonstrates that it is possible to fingerprint a hardware device on the basis of the microscopic variations in skew of their clocks. The concept of clock skew fingerprinting is utilized to achieve frequency offset compensation, ultimately leading to network wide time synchronization of swarm of robots over the wireless medium. Our analysis, as is explained in this section, imply that two nodes cannot accurately skew fingerprint each other over a wireless medium. It is proposed that a third node (poller) can initiate the relative skew measurement among nodes and the average relative skew thus measured can serve as the fingerprint of the nodes. In Section 3.4.3.1, the feasibility of utilizing average relative skew as the fingerprints of nodes is demonstrated. Frequency offset compensation based on relative skew fingerprints is elaborated in Section 3.4.3.2

3.4.3.1 Feasibility Analysis of Relative Skew Fingerprinting

Following experiments were conducted to measure the relative skew of nodes and to validate the feasibility of utilizing the average relative skew as the unique fingerprint of nodes.
Relative skew between reference node and slave nodes was measured using wired polling to ensure that the measurements were free of any inaccuracies introduced by the wireless medium. Nodes, $N_2 - N_6$ were kept stationary, close to reference node N_1. Initial time offset compensation was performed on $N_2 - N_5$ as described in Section 3.4.2. Periodic pulse of 1 second, generated by poller node N_6 was applied to N_1- N_5 via a wired connection. After time offset compensation, the timestamps recorded by nodes $N_1 - N_5$ on the rising edge of pulse from N_6 were transferred to the base station for analysis. If x_i is the RTC value of the reference node and y_i the corresponding value of slave node ($y \in N_2, ...N_5$) at i^{th} instant (on the rising edge of pulse from N_6), then their

instantaneous offset is given by,

$$\theta_i = x_i - y_i; \tag{3.14}$$

Figure 3.6, depict the instantaneous offset of nodes $N_2 - N_5$ with reference node N_1 observed during one of the experiments, after the initial time offset compensation. The experiment was repeated several times, for a duration of one hour each and it was observed that the instantaneous offset among nodes increase to an order of few milliseconds within few minutes, even after the initial time offset compensation. Average instantaneous offset reported for $N_2 - N_5$ during 10 such experiments conducted at different times of the same day is tabulated in Table 3.2. From Table 3.2 , it can be inferred that skew compensation has to be performed to maintain the instantaneous offset among nodes within a deterministic bound. Since the initial time offset among nodes is already compensated and the timestamps were taken at an interval of 1 second, relative skew among nodes can be obtained as.

$$\alpha'_i = \theta_{i+1} - \theta_i; \tag{3.15}$$

The relative skew of nodes $N_2 - N_5$ with respect to the reference node N_1, measured during one of the representative experiments is depicted in Figure 3.7. From the relative skew of nodes, $N_2 - N_5$ measured with respect to N_1 during repeated experiments (50 experiments, each of one hour duration) over three months in indoor uncontrolled environment, it was observed that the nodes exhibit unique relative skew. It is validated through repeated experimental observations that the relative skew of nodes can be modelled as a constant - with random gaussian noise for stable environments. The average relative skew α'_{avg} can be calculated using the 'n' pair of timestamps

Table 3.2. Average Time offset (in μs) between slave nodes and reference node N_1, after initial time offset compensation

Slave	Elapsed Time (in μs)				
Node	10	60	300	480	600
N2	74.6	451.6	2202.8	3556.4	4516.8
N3	34.4	203	1108	1808.8	2341.2
N4	86.2	497.8	2525.2	4040.8	5040
N5	-56.6	-301.8	-1566.6	-2517.8	-3147.6

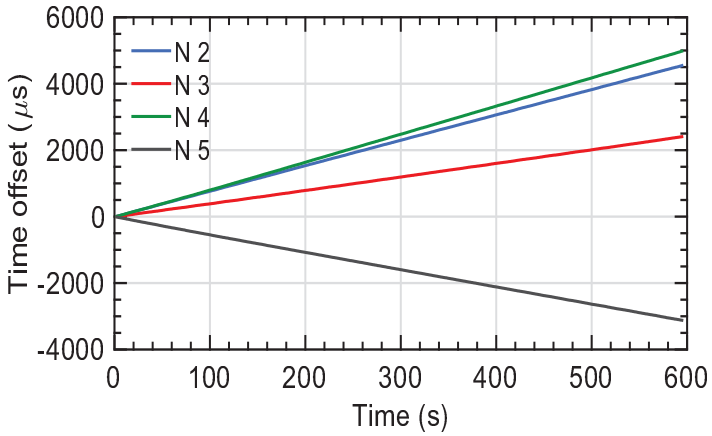

Figure 3.6. Time offset of slave nodes $N_2 - N_5$ with respect to reference node N_1, after initial time offset compensation.

from nodes as follows.

$$\alpha'_{avg} = \frac{\sum_{i=1}^{n}(\theta_{i+1} - \theta_i)}{n} \tag{3.16}$$

α'_{avg} for the given set of nodes, (for $n = 600$) never exhibited a variation ($\Delta\alpha'_{avg}$) greater than '0.5' during the three month long experiments in indoor conditions, which indicate that the average relative skew is bounded and can be utilized as the fingerprint of nodes.

Influence of Varying Temperature on Relative Skew: The most predominant environmental factor which affects clock skew is temperature [50]. [89] report that the change in clock frequency of a Mica2 node is no greater than $1\mu s$/s when temperature changes by 5°C. To estimate the effect of temperature on relative skew fingerprinting, experiments were conducted in indoor- temperature controlled environment as well as outdoor environments as follows.

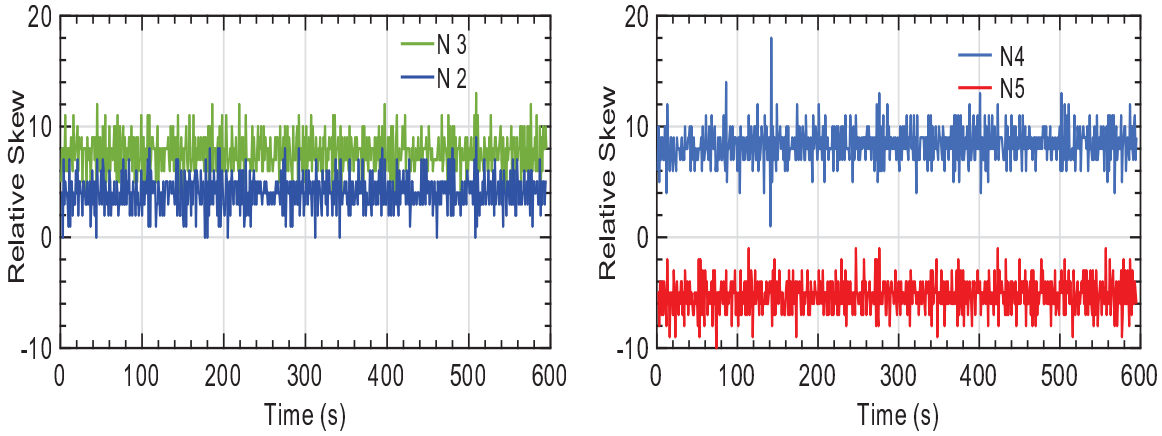

Figure 3.7. Relative skew measured for nodes $N_2 - N_5$ with respect to reference node N_1.

Scenario 1: Indoor Temperature Controlled Environment: Representative slave nodes N_2 (exhibiting positive relative skew with N_1), N_5 (exhibiting negative relative skew with N_1), reference node N_1 and poller node N_6 were kept in a temperature controlled chamber maintained at 23°C and time offset compensation was performed on the slave nodes as in Section 3.4.2. After 10 minutes, temperature of the room was increased and maintained at 27°C for 80 minutes, after which the temperature was again switched back to 23°C. Timestamps of N_1, N_2 and N_5 were recorded at an interval of 1 second after the time offset compensation using poller node N_6. Figure 3.8 provides the relative skew of N_2 and N_5 measured with respect to N_1 during one of the experiments. The variation in relative skew is evident at the temperature switching points around 600 and 5400 seconds. Most microcontrollers including STM32F407VGT6 used in our experiments are clocked using AT-quartz crystals which exhibit cubic variation of skew with temperature. For AT-cut crystals, the change in frequency due to variation of temperature can be expressed as

$$\Delta f / f_0 = a_0(T - T_0) + b_0(T - T_0)^2 + c_0(T - T_0)^3 \tag{3.17}$$

'T' is the current temperature and 'T_0' is the inflection temperature, which is approximately 25°C for AT-cut crystals [89]. The coefficients a_0, b_0, and c_0 are the first, second, and third order temperature coefficients of frequency, and are constants that depend on quartz properties and the angle of cut. Nodes (which have crystals belonging to the AT-cut) subjected to a given temperature exhibit similar variation in frequency with temperature as in equation (3.17). The increase or decrease in temperature will lead to increase or decrease in the clock frequency depending on the temperature coefficient. However, the reported variation in relative skew as in Figure 3.8 is owing to the fact that poller node is also exposed to temperature changes.

Scenario 2: Indoor Temperature Controlled Environment: The experiment was repeated as in scenario 1, with the poller node N_6 kept in a temperature insulated case and N_1, N_2, N_5 exposed to temperature changes. The measured relative skew of slave nodes is depicted in Figure 3.9, which indicate that the relative skew is nearly constant without significant variation around the temperature switching points, despite of the temperature changes. It can be concluded that when the relative skew between the reference node and slave node is measured by the poller node which is not exposed to temperature changes, the variation in average relative skew is negligible as both reference and slave nodes are subjected to similar variation in frequency as in equation (3.17). Repeated experiments validated our observation.

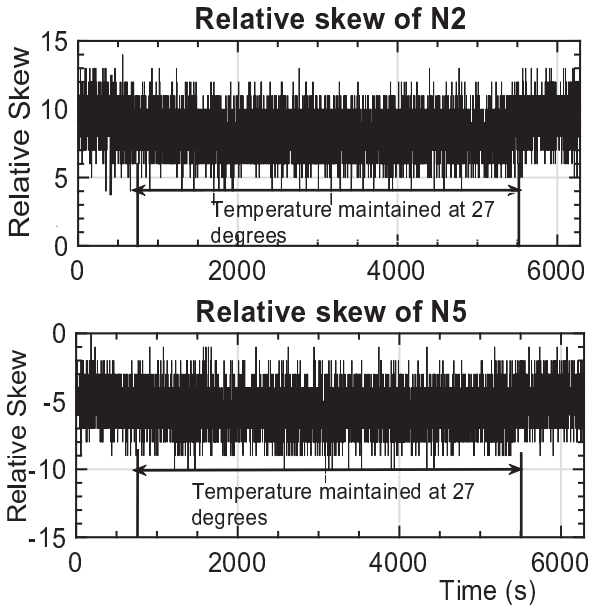

Figure 3.8. Analysis of relative skew with temperature. For t≤600 and t≥5400, temperature is maintained at 23°C.

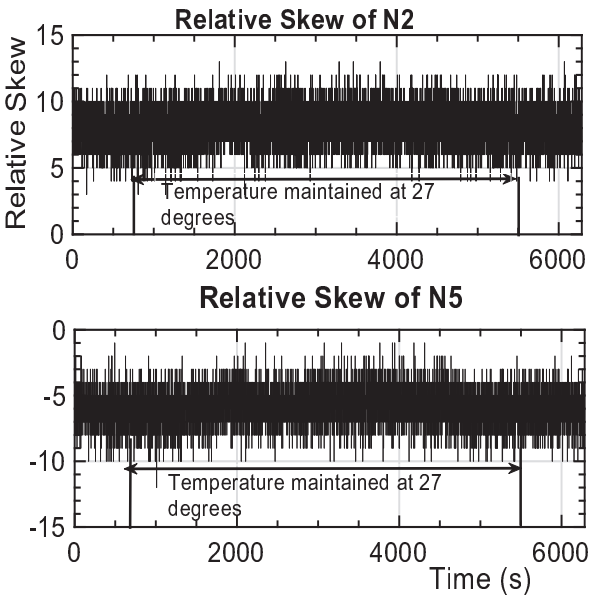

Figure 3.9. Analysis of relative skew with temperature. Poller node is kept in insulated case. For t≤600 and t≥5400, temperature is maintained at 23°C.

Scenario 3: Outdoor Environment: Nodes N_2, N_5, reference node N_1 and poller node N_6 were kept in outdoors, for a period of 8 hours from 6am to 2 pm. As in scenario 2, the poller node was kept in a temperature insulated case. Nodes N_1, N_2 and N_5 were kept close to each other, exposed to sunlight. On-board temperature sensors on N_1, N_2 and N_5 reported a gradual increase of temperature from 26°C to 37°C during the course of experiment. Using poller node N_6, timestamps of N_1, N_2 and N_5 were recorded at an interval of 1 second, after the time offset compensation. The distribution of relative skew of slave nodes measured for a duration of 8 hours is depicted in Figure 3.10. Moving average of relative skew with window size of '8' for N_2 and N_5 was calculated and the distribution of average relative skew for the duration is depicted in Figure 3.11 as box diagram. It can be observed that even when only 8 timestamps are considered for measuring average relative skew, $\Delta\alpha'_{avg}$ is restricted to +/- .5 for the entire duration. It can be observed from Figure 3.11, that despite the wide temperature changes, average relative skew of nodes resembles the average relative skew observed in Figure 3.7.

Based on these experimental results, it can be inferred that a third node (poller) can accurately measure the relative skew between two nodes. From the experimental analysis, it can be concluded that the average relative skew fingerprints estimated at a given temperature can be used for frequency

offset compensation, even in dynamic environmental conditions, if the poller node is stable for variations in temperature. To this effect, the poller node can be designed with a temperature controlled crystal oscillator (TCXO) or oven controlled crystal oscillator (OCXO) as clock source. TCXO and OCXO incur higher cost when compared to the typical crystal oscillators used in microcontroller boards, however their clock frequencies are stabilized for variation in temperature [89]. If the temperature profile of the deployment area is prone to wide variations, or nodes in the system use crystals belonging to different 'cuts', then use of TCXO or OCXO for all nodes is recommended. TCXO or OCXO generated clocks exhibit unique relative skew which can be fingerprinted. However, the variation of frequency with temperature will be negligible. The observations from these experiments indicate that the average relative skew can be utilized as the fingerprint of the nodes.

3.4.3.2 Frequency Offset Compensation

Having experimentally validated that the average relative skew can serve as the fingerprint of a node, a relative skew fingerprinting based frequency offset compensation scheme is proposed. For a network in which nodes can be dynamically introduced or removed, the nodes should be able to estimate and calibrate their average relative skew, post the network deployment. Frequency offset compensation is achieved in two phases, frequency offset calibration and reference clock estimation. It is demonstrated that with the relative skew fingerprinting based frequency offset compensation the resynchronization interval can be improved significantly to an order of few

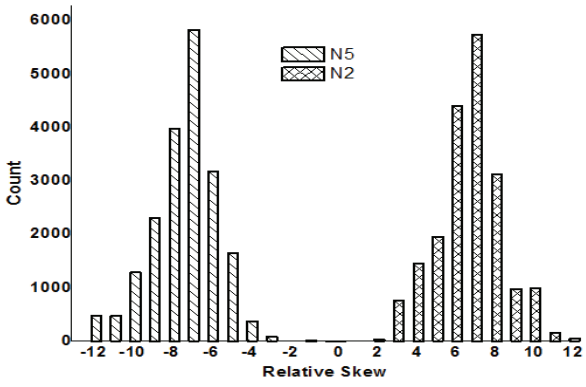

Figure 3.10. Distribution of relative skew with temperature.

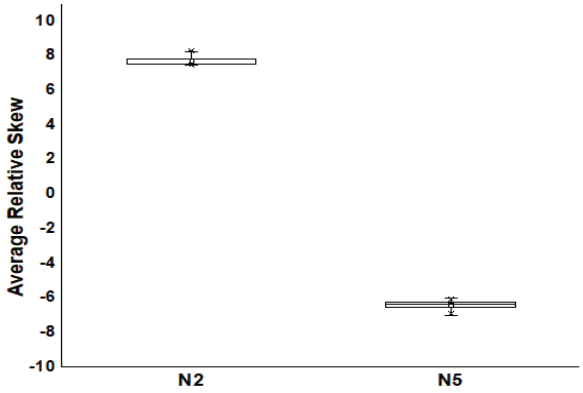

Figure 3.11. Distribution of average relative skew with temperature.

minutes. As the nodes are required to communicate only at an interval of few minutes, the time synchronization is energy efficient and permits the wireless bandwidth to be free for other primary activities of the robot like task allocation, navigation, data routing etc.

Frequency offset calibration: An accurate method of error free relative skew measurement, over a wireless medium is described in this section. The method accounts for any uncertainities introduced in timestamping as explained in Section 3.3. The proceedure for estimating the average relative skew fingerprint is as follows. After time offset compensation, reference node broadcast a control frame of the 'Message_type'- "frequency_offset_sync" to initiate frequency offset or relative skew calibration. The format of the control and data frame utilized for relative skew calibration is same as in Figure 3.2. The relative skew calibration is optional. If 'M' bit in the command field of control frame is set, relative skew calibration is mandatory for nodes. If 'M' bit is reset the slave nodes can optionally participate in the skew calibration process or utilize the past estimate of α'_{avg} or α'_{avg} obtained aprior to network deployment for reference clock estimation. The reference node, along with the poller node and the slave node which receives the "frequency_offset_sync" control frame switch to "frequency offset calibration mode" based on the control message. In frequency offset calibration mode, the nodes disable interrupts other than systick, disable task switching and poll on GD02. Poller node transmits 8 polling messages [1] at an interval of 1 second as depicted in Figure 3.12. Reference and slave nodes on reception of polling messages record their timestamps at rising edges of GD02. The recorded timestamps of the reference node are then transmitted to the slave node as a single broadcast message to the slave node. The broadcast 'Message_type' is configured as 'frequency_offset_sync' and the 'C/D' bit of command field is 'reset' to indicate that payload carries 'data'. The eight timestamps (40 bytes) in the format, hours : minutes : seconds : milliseconds : microseconds are embedded in the payload and broadcast to slave nodes by the reference node. Slave nodes on reception of the reference node timestamps, calculate their α'_{avg} with the 8 pairs of timestamps as per equation (3.16) after the removal of outliers if any.

The proposed wireless relative skew calibration scheme completely eliminates the receiver and transmitter uncertainties in periodic timestamping as mentioned in Section 3.3 as follows. The timestamping of reference node and slave node is initiated by a poller node, hence the uncertainties in the transmitter timestamping as mentioned in equation (3.6) is completely eliminated. Propagation delay for distances less than 300 meters is $\leq 1\mu s$ and since the granularity of RTC is $1\mu s$, error

[1] The reason for selecting '8' timestamps for wireless relative skew fingerprinting is elaborated later in this section.

due to variation in propagation delay is negligible. The receiver uncertainty is caused by two factors, the interrupt jitter and the receive time as in equation (3.6). Since polling based timestamping is implemented, the receiver interrupt jitter is eliminated. As task switching is eliminated (interrupts

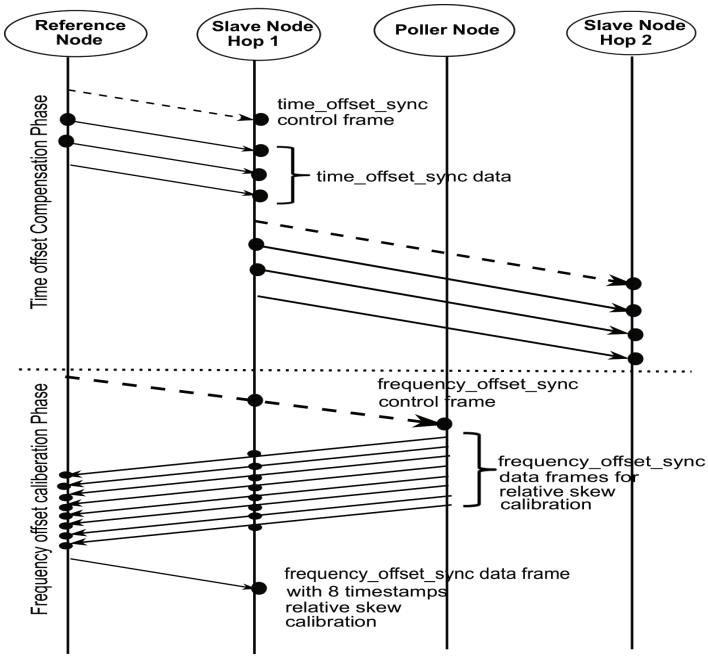

Figure 3.12. Overall Swarm-Sync Framework. (Dotted lines indicate control frames, solid lines indicate the data frames, Bulleted line ends indicate the timestamping instants)

other than systick are disabled) during relative skew calibration, the receive time is deterministic. Since the timestamping for relative skew calibration is initiated by the poller node, the reference and slave nodes are receivers of the polling messages and the reception time delay will be the same for all receiver nodes. Thus, the uncertainty in the receiver timestamping as represented in equation (3.6) is eliminated and the timestamps recorded at the receivers (reference and slave nodes) during relative skew calibration can be considered to be recorded simultaneously. If timestamping was initiated by the reference node, then the slave node should eliminate the transmitter and receiver uncertainties from the timestamps. When relative skew of nodes is in the order of less than $10\mu s$/s as indicated in Figure 3.7, the timestamping errors can be very critical. Hence if the relative skew calibration of slave nodes is initiated by the reference node without using the poller, relative skew estimation and calibration will not be accurate. As the proposed relative skew calibration scheme is error free, the same proceedure can be utilized for wirelessly fingerprinting the robots/nodes prior to their dispersion or deployment,as in Section 3.4.3.1 instead of using wired timestamping. It is recommended that nodes are fingerprinted prior to deployment either wirelessly or through wired

connection and the relative skew may later be optionally re-calibrated post their deployment if necessary. The relative skew calibration is essential when which new reference node is introduced in the system.

Reference Clock Estimation: After time offset compensation, the nodes switch to frequency offset compensation mode. If nodes receive a "frequency_offset_sync" message, then the slave nodes may temporarily switch to frequency offset calibration mode and measure the average relative skew using wireless messages. In frequency offset compensation mode, nodes enable interrupts, enable task switching and perform reference clock estimation as is explained in Algorithm 1. The slave nodes utilize the α'_{avg} measured prior to deployment or the value measured during the relative skew calibration phase for reference clock estimation between reference node and slave node till the next resynchronization interval as follows. The default value of the resynchronization interval, T_{resync} is set to '1' minute for the slave nodes. 'Count' field in the paylod during "$time\_offset\_sync$" control packet can be used by the reference node to configure the resynchronization interval. Depending upon the desired synchronization accuracy, the resynchronization interval of slave node can be adaptively configured as $Count * T_{resync}$.

Depending upon whether the application demands continuous synchronization or post-facto synchronization, slave nodes can estimate the reference node time as follows. For continuous synchronization, the slave node which has the average relative skew α'_{avg} with respect to the reference node can estimate the reference clock time after every second as

$$\hat{T}_{ref} = T_{slave} + \alpha'_{avg} \tag{3.18}$$

where T_{slave} is the RTC time of the slave and $\hat{T}_{ref}$ is the estimated reference node time. If continuous synchronization is not required for the application, then post-facto synchronization can be implemented on the offset compensated nodes as follows. For any event, at time T_{slave}, a slave node which has α'_{avg} with respect to the reference node can estimate $\hat{T}_{ref}$ as

$$\hat{T}_{ref} = T_{slave} + S * (\alpha'_{avg}) \tag{3.19}$$

where 'S' is the number of seconds elapsed after the last offset compensation. 'S' is derived from the RTC. $\hat{T}_{ref}$ is utilized by the slave node when time information of node is required. Since $\Delta\alpha'_{avg}$ is bounded, as is explained in Section 3.4.3.1, for a given resynchronization interval the maximum

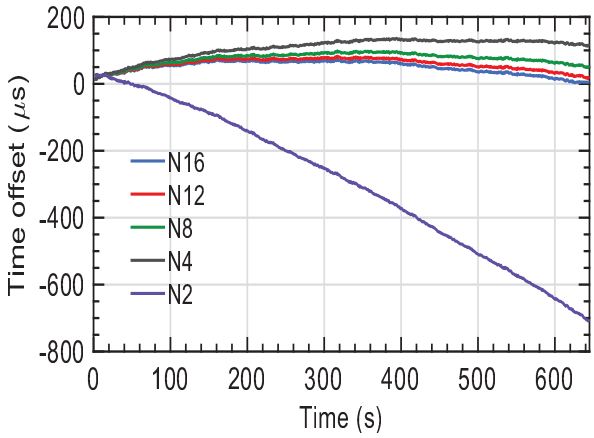

Figure 3.13. Synchronization error (time offset) for different values of 'n'

error in reference clock estimation is bounded and can be estimated as

$$max\{\hat{T}_{ref} - T_{ref}\} = T_{resync} * max\{\Delta\alpha'_{avg}\}\mu s. \tag{3.20}$$

where T_{resync} is the resynchronization interval in seconds, T_{ref} is the reference node clock and $\Delta\alpha'_{avg}$ is the maximum variation in average relative skew for the given set of nodes.

To calculate an appropriate number of timestamps 'n' required for relative skew fingerprinting, synchronization of one of the slave nodes with reference node using equation (3.18) was attempted. The α'_{avg} calculated using different values of 'n' through wireless relative skew fingerprinting was utilized for reference clock estimation. The time offset of the slave node with reference node after reference clock estimation according to equation (3.18) is shown in Figure 3.13. It can be observed that as 'n' increases, the time offset decreases. However, as a trade-off between accuracy and incurred communication overhead, the value of n=8 was chosen. This was validated through repeated experiments.

For the given set of nodes N_1-N_5, the maximum $max\{\Delta\alpha'_{avg}\}$ was measured to be .5, as explained in Section 3.4.3.1. The resynchronization interval can be configured according to the required bound of synchronization error as follows. For example, if the synchronization error, $max\{\hat{T}_{ref} - T_{ref}\}$ has to be bounded to 300 μs, the resynchronization interval can be as high as $\approx$ 10 minutes. Since the maximum change is clock frequency of a crystal is bounded it can be assured that for any set of nodes, $max\{\Delta\alpha'_{avg}\}$ is also bounded if accurate timestamping is implemented .

The overall Swarm-Sync framework is depicted in Figure 3.12. Only 4 broadcast messages are required for synchronization of nodes in a hop, if relative skew calibration is not required. If

relative skew calibration is required, then additional 9 broadcast messages are required per hop for synchronization within a given resynchronization interval. The proposed framework is appropriate for scalable networks as the number of messages required for synchronization is independent of the number of nodes in the network. In Swarm-Sync framework, once a control frame is received by the slave nodes, the nodes switch to the corresponding operational mode and wait for the data frames from reference node. However to accommodate for the possible issues which can occur due to loss of packets due to collision or due to the fact that nodes are mobile and have moved out of the coverage zone, it is advisable to use a timer to monitor the wait period of node. If data frames are not received within the wait period, the node continues the reference clock estimation with the previous estimate of average relative skew available with the node.

3.4.4 Experimental Validation of Swarm-Sync Framework

3.4.4.1 Validation for Single-hop Networks

To evaluate the accuracy of proposed Swarm-Sync framework for single-hop networks, two test scenarios were designed. For both scenarios, nodes $N_2 - N_5$ were programmed to navigate randomly within one-hop distance with a velocity of 50-80cm/sec. For the first scenario, frequency offset compensation was performed on nodes after time offset compensation using α'_{avg} measured offline prior to deployment (wired timestamping, n=600) using equation (3.18). For second scenario, frequency offset compensation was performed after time offset compensation using α'_{avg} [2] calculated from wireless timestamping (n=8) post deployment on mobile nodes. T_{resync} is configured as 10 minutes for both scenarios. 40 rounds of experiments were conducted for both scenarios- spread over a duration of 1 month in uncontrolled indoor environmental conditions. The maximum error in synchronization measured during 40 rounds of experiments and the error bound obtained for 90% of the trials for Scenario 1 and Scenario 2 is indicated in Table 3.3. The representative results obtained during one of the experiments for Scenario 1 and Scenario 2 is depicted in Figure 3.14 and Figure 3.15 respectively which indicate that the synchronization error and the variance in synchronization error among nodes is bounded. It can be observed that even

[2] α'_{avg} with precision of 2 decimal digits is used for skew compensation. $\hat{T}_{ref}$ is rounded off to the nearest integer.

with wireless relative skew fingerprinting, the synchronization error is bounded ($\leq 300\mu s$) for a resynchronization interval of 10 minutes.

3.4.4.2 Validation for Multi-hop Networks

For a multihop scenario, the components of Swarm-Sync framework can be utilized in three different ways for synchronization, depending on the characteristics of the network.

Option 1: Hop-wise time offset compensation of the network, followed by frequency offset compensation with average relative skew fingerprints calculated prior to network deployment.

Option 2: Time offset compensation, immediately followed by frequency offset compensation with relative skew calibration, repeated one hop after the other.

Option 3: Hop-wise time offset compensation of the network, followed by frequency offset compensation with relative skew calibration.

Swarm-Sync framework was tested in a 4-hop scenario with node deployment as shown in Figure 3.16. Nodes were configured to move at a velocity of 80 cm/second within each zone. For Option 1 and 2, once the time offset compensation of nodes at one-hop level is completed, selected offset compensated nodes serve as the reference node for the time offset compensation of the next hop nodes. The selection of intermediate reference node is not covered within the scope of this work as the selection criteria will vary based on the application and its constraints. In our deployment scenario, N_2, N_3, N_4 serve as the reference node for the next hop nodes during time offset compensation. N_2 compensate its time offset with respect to $N1$ after which it serves

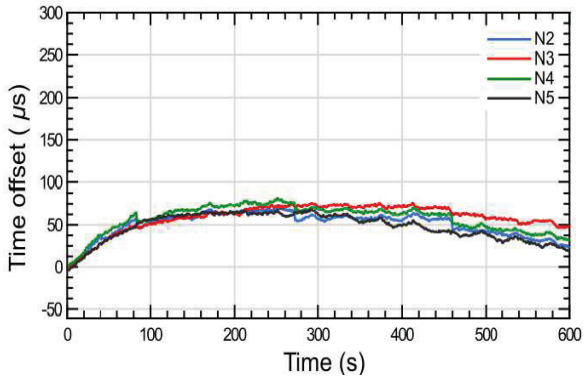

Figure 3.14. Time offset between reference and slave nodes post synchronization, α'_{avg} calculated prior to deployment using wired relative skew fingerprinting.

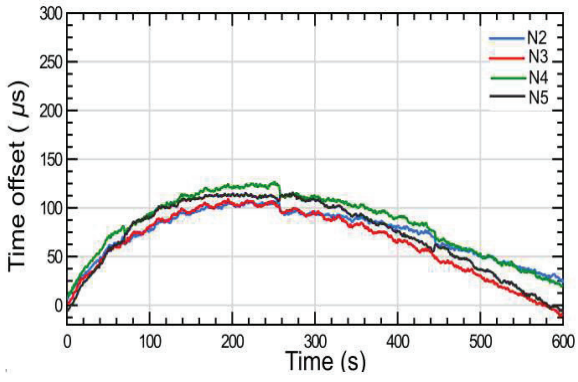

Figure 3.15. Time offset between reference and slave nodes post synchronization, α'_{avg} calculated using wireless relative skew fingerprinting post the deployment.

Algorithm 1: Swarm-Sync Algorithm

Output Estimated reference clock $\hat{T}_{ref}$ by slave node

:

start:
if ((*Message_type* == *time_offset_sync*) & ($C/D == 1$)) **then**
  $T\_resync = Count$;
  $RN_{Address} = Sender\ Address$;
  mode = *time offset compensation mode*
end if
while (*time offset compensation mode* == *true*) **do**
  disable interrupts ();
  disable task switching ();
  reset RTC();
  start polling GDO2 ();
  for $i = 0$ to 2 **do**
    if (*EOP detected at GDO2*) **then**
      if ($i == 0$) **then**
        enable systick timer() and *systick interrupt*();
        enable RTC();
      end if
      $i = i + 1$;
      t_{i+1} = *record time stamp* ();
      if ($Sender\ Address != RN_{Address}$) **then**
        discard time stamp ;
        decrement i;
      else if ($Count <= 2$)&($Sender\ Address == RN_{Address}$)) **then**
        obtain t_{i-1} *from received message*
      end if
    end if
  end for
  if ($Count >= 2$) **then**
    $i = 0$;
    perform time offset compensation using equation(3.13);
  end if
end while
if ((*Message_type* == *frequency_offset_sync*) & ($C/D == 1$)) **then**
  $RN_{Address} = Sender\ Address$;
  mode = *frequency offset calibration mode*
end if
while (*frequency offset calibration mode* == *true*) **do**
  disable interrupts except systick();
  disable task switching ();
  start polling GDO2 ();
  for $j = 0$ to 7 **do**
    if (‘*sync*′ *detected at GD*02) **then**
      t_j = *record time stamp* ();
    end if
  end for
  enable all interrupts();
  enable task switching ();
  for ($j == 8$) **do**
    $l = 0$;
    if (*GD*01 == 1) **then**
      for $k = 0$ to 7 **do**
        t_k = *obtain time stamp of reference node from received messages*();
        $\theta_l = t_k - t_j, l = l + 1$;
      end for
    end if
  end for
  remove outliers if any
  $n =$ (8 − *number of outliers*)
  Calculate α'_{avg} *from equation* (3.16)
  mode = *frequency offset compensation mode*
end while
if ((*frequency offset compensation mode* == *true*)) **then**
  $j = 0$;
  enable all interrupts();
  enable task switching ();
  perform skew compensation with last updated α'_{avg} *as per equation* (3.19) *or equation* (3.20) ;
end if
repeat from start

Table 3.3. Synchronization error between slave nodes and reference node N_1 utilizing - Swarm-Sync framework

	Synchronization Error (in μs)		
	Wired, relative skew fingerprinting (scenario 1)	Wireless relative skew fingerprinting (scenario 2)	Wireless relative skew fingerprinting (4-hop scenario)
Error Bound for 90% trials	110	175	210
Maximum Error	127	192	243

as the reference node for offset compensation of N_3. After time offset compensation of N_3, N_2 switches to 'frequency offset compensation mode'. Similarly, N_4 and N_5 correct their time offsets and switch to 'frequency offset compensation mode'. After switching to frequency offset compensation mode, nodes enable interrupts, enable task switching and compute reference clock $\hat{T}_{ref}$ with the α'_{avg} calculated prior to node deployment for option 1 or wait for the relative skew calibration message from the reference node for option 3. Mobility of reference and poller node is utilized to ensure that the frequency offset or relative skew estimation/calibration is achieved with reasonable accuracy in option 3. Poller node and reference node will move to each zone/hop to facilitate the average relative skew estimation of time offset compensated nodes. When poller node and reference node are available for relative skew estimation, the reference node broadcast "frequency_offset_sync" control message. When poller node receives the control message, 8 polling messages are transmitted by poller. Once the 8 timestamps from reference nodes are transmitted to the slave nodes, the reference and the poller nodes move to the next hop to facilitate the relative skew estimation of the next hop nodes. Option 1 is ideal for large scale networks whereas option 3 is suitable only if the coverage area of the network is limited. In Option 3, after time offset compensation, frequency offset compensation is performed on the availability of the reference and poller node. Hence even before the frequency offset caibration is initiated by the reference node, significant time offset may manifest after the time offset compensation due to the skew exhibited by the nodes. Hence even if frequency offset compensation is performed, time represented by the slave nodes may deviate significatly for the first global synchronization cycle, if the coverage area is large. However once relative skew calibration is performed, for subsequent

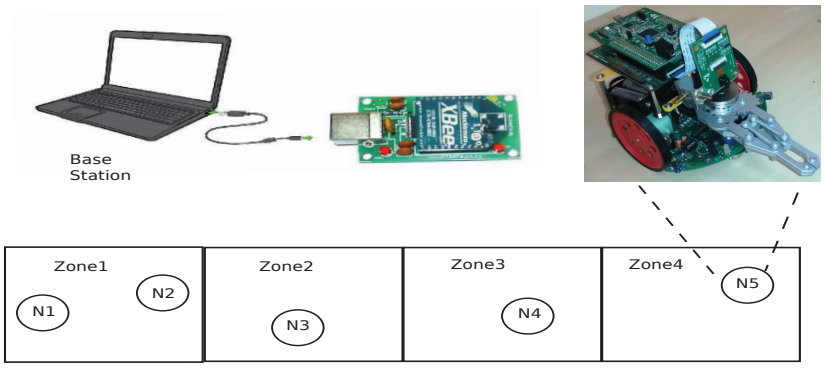

Figure 3.16. Deployment of nodes in multi-hop scenario.

synchronization cycles, the robots can utilize the average relative skew estimated in previous cycle for frequency compensation till the reference and poller node are in vicinity. In applications like land mine detection, airborne robots can be utilized as reference and poller nodes and hence, the synchronization can be completed quickly using option 3. The airborne robots can also be utilized for surveillance and be used to provide supplemetary path planning inputs to the other swarm robots. In Option 2, time and frequency offset compensation of each hop is repeated one hop after the another. The poller node and reference node will move to each hop perform time offset compensation, followed by frequency compensation with relative skew calibration. Option 2 is suitable for small and medium sized networks.

The error distribution for multi-hop network deployment is as follows. The framework utilizes intermediate reference nodes only for time offset compensation. Intermediate reference nodes are not utilized for frequency offset compensation. The accuracy of proposed time offset compensation is elaborated in Section 3.4.2.1. However, time offset compensation of n+1th hop should be initiated as soon as the time offset compensation of nth hop is complete, to avoid any cumulative error propagation due to skew. WSN nodes are typically clocked using crystals which exhibit a skew of 1- 50ppm (e.g. 50ppm implies a maximum variation of 50μs per second from ideal value). In the presented work, once the time offset compensation of nth hop is complete, the control frame for time offset compensation of next hop nodes is transmitted by the intermediate reference node after a delay as follows

$$Delay_{tx} \approx T_{rx-tx} + T_{micro\_w} + T_{tansmit} \tag{3.21}$$

where T_{rx-tx} is the time required for CC2500 to switch from receive to transmit mode ($\approx 21\mu$s [33]), T_{micro_w} is the time taken by microcontroller to write payload data on to CC2500 register

(SPI communication between CC2500 and STM32F407VGT6 microcontroller is at 10Mhz. Hence $\approx 7.2\ \mu$s is incurred for writing data according to format mentioned in Figure 3.2), $T_{transmit}$ is the time taken by CC2500 for transmitting a packet of format as mentioned in Figure 3.2. ($\approx 650\mu$s when CC2500 is configured for data rate of 250Kbps [86]). Similarly, the delay after which a node at $n + 1^{th}$ hop receive the control message can be determined as

$$Delay_{rx} \approx T_{prop} + T_{recp} + T_{micro\_r} \tag{3.22}$$

where T_{prop} is the propagation delay (which is $<< 1\mu$s), T_{recp} is the time for receiving a packet bit-bit at the physcial layer ($T_{recp} \approx T_{transmit}$) and T_{micro_r} is the time taken by the microcontroller to read data from CC2500 registers ($T_{micro\_r} = T_{micro\_w}$= 7.2$\mu$s). The time for transmitting and receiving 4 messages required for the time offset compensation is

$$Delay_{total} \approx T_{micro\_w} * 4 + T_{rx-tx} + T_{transmit} * 4 + T_{receive} * 4 + T_{micro\_r} * 4 \tag{3.23}$$

which is $\approx 6ms$ (for a node with 50 ppm , skew developed during this time is negligible). The skew in the transmitter node and receiver node during time offset compensation is negligible, leading to minimal cumulative error propagation during time offset compensation. It has to be noted that time offset compensation of nodes at one hop distance is completed using broadcast messages unlike pairwise two-way message exchanges as utilized in TPSN or RBS protocol. Hence errors due to accumulation of skew during time offset compensation is negligible. Frequency offset compensation is achieved using relative skew fingerprinting technique and is non-cumulative as $n + 1^{th}$ hop node does not communicate with n^{th} hop node for frequency offset compensation.
For option 1, the performance of synchronization is similar to that obtained for single-hop networks as relative skew fingerprints are calculated a priori and there is no communication between neighbour nodes for frequency offset compensation. Time offset compensation will cause negligible cumulative error as explained above. For Option 2 also, the performance is same as that of single-hop networks as relative skew calibration is performed immediately after time offset compensation in each hop. It can be analyzed that option 3 may lead to higher synchronization error over multiple hops when compared to other two options for the first synchronization cycle due to time difference between time offset compensation and relative skew calibration. To examine the performance of Swarm-Sync network for multi-hop scenarios, experiments were repeated for

40 times, spread over a period of 1 month for Option 3. The reference and poller nodes move to the next hop (in ≈ 3 seconds for our deployment) to facilitate the frequency offset calibration as soon as the calibration of previous hop is complete. The nodes after synchronization (time and frequency offset compensation) were polled for 10 minutes. The time offset after synchronization for one of the experiments is depicted in Figure 3.17. The representative result shown in Figure 3.17 indicates that even in a 4-hop network all nodes can remain synchronized within an error bound of $300\mu s$. The worst-case synchronization error for a 4-hop scenario (for 40 attempts) with wireless relative skew fingerprinting was $243\mu s$. The maximum error in synchronization measured during 40 rounds of experiments and the error bound obtained for 90% of the trials for 4-hop scenario is indicated in Table 3.3

Swarm robots mostly utilize local communication strategies instead of global communication and since the cumulative error over neighbouring hops is negligible, the robots can move across hop or clusters without restriction on its mobility. This also leads to easier implementation of other layers of the protocol stack like medium access control or routing. The proposed scheme is energy efficient as the resynchronization interval can be increased to an order of several minutes. The protocol is scalable as the number of robots required for the synchronization is independent of the number of nodes in the system. Framework can not only provide a bounded synchronization error but also provide a lower variance in the error among nodes even for multi-hop scenarios. This technique does not impose any restriction on the movement of robots and is also robust to loss of robots or nodes. The proposed framework eliminates the sender and receiver side uncertainty in timestamping and at the same time is energy efficient as the number of communication messages are reduced drastically due to the improvement in resynchronization interval.

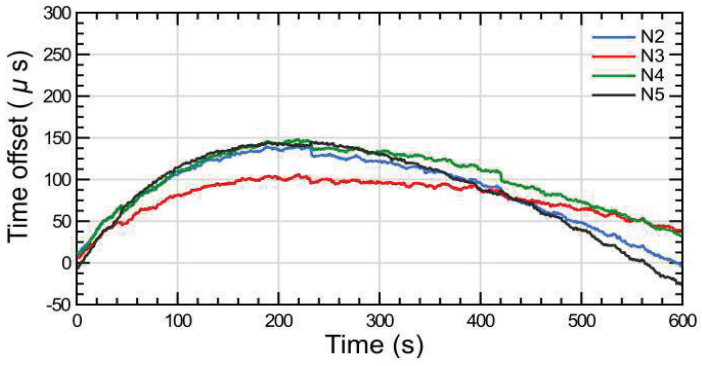

Figure 3.17. Time offset between reference and slave nodes for 4-hop deployment, α'_{avg} calculated using wireless relative skew fingerprinting as in Option 3.

3.5 Comparison of Swarm-Sync Framework with LR, LP and KF and Consensus Based Techniques

Table 3.4 provides a comparison of the popular state-of-the-art time synchronization protocols. These protocols are based on fundamental techniques such as Linear regression (LR), Linear prediction (LP), Kalman filter (KF) or consensus theory. In this section, an analysis of these different techniques is experimentally performed and their suitability for synchronization of dynamic networks like swarm robotic systems is evaluated. These techniques are evaluated based on the following crucial aspects.

1. Can the resynchronization interval be extended to an order of few minutes?.
2. Can the technique provide reasonable time synchronization accuracy when performed on nodes with single precision floating point unit (FPU), which is the common type of FPU available in most of the microcontrollers based nodes?.
3. Can the technique provide synchronization with bounded error over a multi-hop network?.
4. Is the technique scalable and robust to node failures?.

3.5.1 Comparative Analysis with LR, LP, KF Techniques

LR, LP and KF techniques are predictive whereby the slave nodes predict the reference node clock using the pairwise timestamps obtained from reference and slave nodes at periodic intervals of time. The protocols based on these techniques recommend resynchronization intervals in the order of few seconds as indicated in Table 3.4. A common experimental set up was designed with the objective to estimate the best-case prediction accuracy provided by LR, LP and KF techniques for a given resynchronization interval on a given set of nodes. Nodes, $N_1 - N_6$ were kept stationary at one hop distance from each other and were switched-on one by one, after random delay to introduce initial offset among the nodes. A periodic pulse of 1s was generated by the poller node N_6 and at the rising edge of pulse, timestamps were recorded at the reference and slave nodes simultaneously. Polling was done using a wired connection between poller and other nodes to avoid uncertainties

associated with wireless timestamping. Pairwise timestamps from reference node (N_1) and slave nodes ($N_2 - N_5$) were fed into MATLAB tool (computations using double precision floating point data type) and the same set of timestamps were utilized to predict the reference clock using each of the three techniques. The prediction error (T_{error}), i.e the difference between the measured and predicted reference clock was calculated for the three techniques. The pairwise timestamps were also fed onto robots (computations using single precision floating point data type) and prediction accuracy of the three techniques when computations were performed on nodes was evaluated.

Linear Regression (LR) FTSP and PulseSync protocols utilize linear regression to estimate the reference node clock. These protocols do not incorporate explicit initial time offset (θ_0) compensation and recommend the use of most recent, 8 pairwise timestamps obtained at resynchronization intervals (T_{resync}) from reference and slave node using one-way wireless timestamping for linear regression. Frequent updation of regression table at an interval of 30 s and 10 s as recommended by FTSP and PulseSync respectively is not feasible in mobile networks. Hence, we maintained a regression table with 8 timestamps obtained at an interval of 1 s and these 8 timestamps were used to calculate the regression coefficients 'α' and 'β'. The regression coefficients thus obtained were utilized to predict reference node time for 10 min duration (T_{resync} =10 min) as in Algorithm 2. The experiment was also repeated for T_{resync}=1 min. For T_{resync}=1 min, the regression table was updated after every minute, with 8 new timestamps obtained at an interval of 1s. The prediction error for two representative nodes N_2 and N_5 obtained during one of the representative experiments, provided in Figure 3.18.a and Figure 3.18.b indicate that the error is significant for T_{resync} = 10 min when compared to T_{resync} = 1 min. It can be observed that even with double precision data type computations (MATLAB), LR method leads to significant error in prediction due to the initial offset among nodes.

Pairwise timestamps were obtained from reference and slave nodes after performing time offset compensation on slave nodes as suggested by Swarm-Sync framework in Section 3.4.2. LR based reference clock prediction was performed using timestamps for T_{resync}=10 min on nodes as well as on MATLAB. The prediction error for representative nodes N_2 and N_5 when the prediction is performed offline (MATLAB) and on nodes is depicted in Figure 3.18.

Table 3.4. Comparison of Swarm-Sync with other time synchronization protocols

Protocol	Resync interval (s)	Average Accuracy (μs)	No of Messages per hop/cluster with 'n' nodes	No of iterations for Convergence	Synchronization Technique	Cumulative error over multi-hop
RBS[90]	-	29.1	$n(n-1)/2$	- -	Pairwise 2-way messaging	High
TPSN [91]	-	16.9	$2(n-1)$	- -	Pairwise 2-way messaging	High
FTSP [48]	30	1.48	n	-	LR	High
PulseSync[33]	10	2.06	n	-	LR	High
ALPS [49]	2	19.33	n	-	LP	High
AMKF[50]	1000	1000	n	-	KF	High
ATS[51]	30	600	n	120	Consensus	High
MTS[52]	1	100	n	212	Consensus	High
CCTS[51]	1	30.2	$n+3$	31	Consensus	High
CMTS[53]	1	≈ 5	$n+2$	126	Consensus	High
Sywarm-Sync	600	110	13	-	Skew Fingerprinting	Negligible

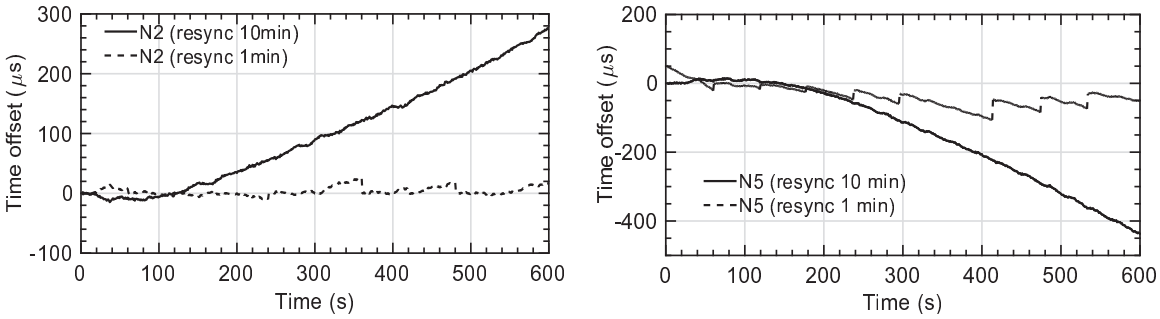

Figure 3.18. Time offset (Prediction error) for LR based reference clock prediction for different T_{resync} periods a) Initial offset between between N_1 and N_2 was 60ms b) Initial offset between N_1 and N_5 was $10\mu s$

The results indicate that the prediction error has reduced significantly even for $T_{resync} = 10$ min when compared to Figure 3.18.a and Figure 3.18.b. From repeated experiments it was concluded that frequent resynchronization as pointed out in FTSP or PulseSync can be avoided if time offset among nodes is compensated. Moreover, it was observed that there is significant difference in prediction accuracy when prediction is performed offline, when compared to accuracy when the prediction is performed on nodes as shown in Figure 3.18. STM32F407VGT6 microcontroller used in our experiments has a FPU which supports single precision floating point operations. Hence the values of $\hat{\alpha}$ and $\hat{\beta}$ (eg: +/- 1.00000865394978 i.e. with 13 or more digits of precision) are truncated to 7 decimal digitals resulting in error in prediction when compared to the prediction computed on MATLAB. Commonly used nodes in WSN like TelosB or Micaz do not support FPU and the LR computations on them will result in significant truncation and error in prediction. LR technique was also tested for a 4-hop scenario with node deployment as shown in Figure 3.16. Nodes N_2, N_3, N_4 serve as intermediate reference nodes for LR prediction of the next hop nodes after time offset compensation. The timestamps were collected from nodes at an interval of 1s

Algorithm 2: Reference clock prediction using Linear Regression

Input : Timestamps from slave node $x_{i-1}, x_{i-2}, ..x_{i-8}$ and reference node $y_{i-1}, y_{i-2}, ..y_{i-8}$ obtained at intervals of $i = 1$ second at the start of resynchronization interval

Output : Predicted reference clock $\hat{y}_i$

1 Calculate $\sum_1^8 x_i$, $\sum_1^8 x_i^2$,$(\sum_1^8 x_i)^2$,$\sum_1^8 y_i$, $\sum_1^8 x_i y_i$;

2 $n = 8$; $\beta = \frac{n\sum x_i y_i - (\sum x_i \sum y_i)}{n\sum x_i^2 - (\sum x_i)^2}$

3 $\alpha = \frac{\sum y_i}{n} - \beta\frac{\sum x_i}{n}$;

4 $y_i = \beta * x_i + \alpha$; calculate y_i for i ≥9 till end of resynchronization interval;

5 Repeat from step 1 after T_{resync} with new 8 pair of timestamps

using wired timestamping and the reference clock prediction was performed on MATLAB. The measured prediction error for one of the representative experiment is shown in Figure 3.19. It can be observed that when T_{resyc} =10 minutes, the prediction error between the reference and slave nodes increase to an order of few milliseconds even for a 4 node scenario under ideal conditions. It can be observed that the LR technique lead to cumulative error over a multi-hop scenario, thus leading to large variance in the time of nodes throughout the network as is indicated in Figure 3.20. The LR prediction is highly sensitive to the timestamps and prediction error over a multihop scenario will vary with the deployment pattern of nodes and hence the prediction error is not deterministic.

In Swarm-Sync framework, the skew compensation by slave nodes is independent of other slave nodes in the system, thus eliminating cumulative error propagation. The framework also guarantees a bounded synchronization error irrespective of the number of nodes in the network. From Figure 3.17 and Figure 3.18, it can be observed that the variance of error among nodes is much lesser for Swarm-Sync framework when compared to LR even in multi-hop scenarios, thus making Swarm-Sync protocol ideal for situations which are prone to dynamic cluster formations/changes. Our analysis indicates that LR based technique is not ideal for synchronization of multi-hop, scalable and dynamic network like swarm of robots.

Linear Prediction (LP) LP is widely applied in signal estimation problems in which the future values of a signal is predicted based on the past signal samples. ALPS [49], predicts the instantaneous offset between a reference node and a given node based on past measurements of their offsets as follows.

$$\hat{\theta}_{i+1} = a_1\theta_i + a_2\theta_{i-1} .... + a_p\theta_{i-p} \tag{3.24}$$

where a_j $(j = 1, 2, ...p)$ are the prediction coefficients, 'p' is the prediction order, θ_i.. θ_{i-p} are the measured offset between the reference node and slave node at fixed intervals and $\hat{\theta}_{i+1}$ is the predicted offset at $i + 1^{th}$ instant. LP is computationally complex (Algorithm 3), Levinson-Durbin algorithm requires a nested loop with a complexity of O(p^2). [49] suggests prediction order of 3 to strike a balance between computational complexity and prediction error. Using wired polling, timestamps were obtained from representative nodes N_2 & N_5 after time offset compensation. LP based prediction (Algorithm 3) was performed with $p = 3$ as recommended by ALPS. The prediction coefficients were updated on every iteration by minimizing mean square error between

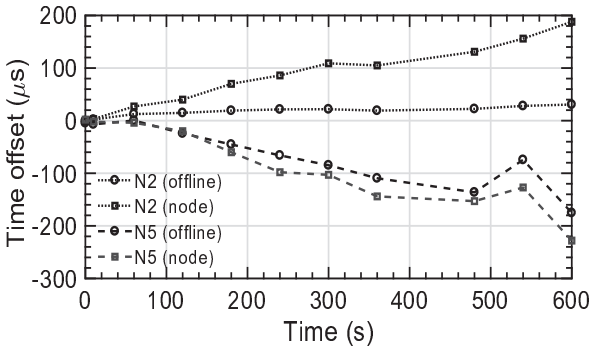

Figure 3.19. Time offset after LR prediction, when performed on nodes (hardware) Vs offline (MATLAB) for T_resync =10 minutes

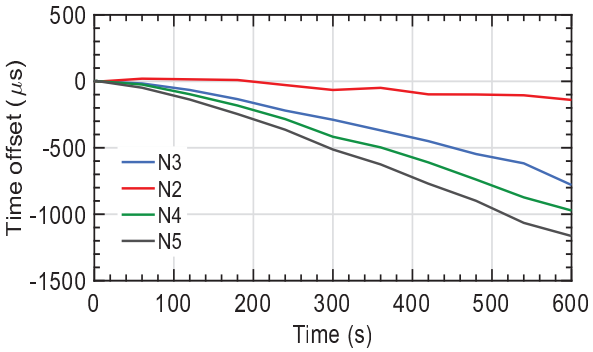

Figure 3.20. Time offset after LR prediction for 4-hop scenario (LR performed offline (MATLAB).

measured and the predicted offset. It can be observed from Figure 3.21.a, that the average error in prediction (computed using double precision floating point data type in MATLAB) is approximately $20\mu s$ and $5\mu s$ for N_2 and N_5 respectively, for timestamps taken at an interval of 1 second (T_{resync}). The prediction error increases drastically (Figure 3.21.b) when timestamps obtained at T_{resync} of 30 seconds is used for the prediction. From repeated experiments, it was also observed that LP technique produces a non-zero steady-state error which increases with the sampling interval (Figure 3.21.b). When LP is conducted on nodes and with uncertainties introduced by the wireless timestamping, the prediction error will increase substantially even for smaller resynchronization intervals in the order of few seconds. Hence LP technique is not suitable for synchronization if the desired resynchronization interval is greater than few seconds even for single hop scenarios.

Kalman Filter (KF) Researchers [50] have proposed a multi-model Kalman filter for dynamic environments, where as a constant-skew model is suggested for constant environmental conditions. A constant skew model based Kalman filter (Algorithm 4) was implemented to evaluate the performance of clock synchronization under constant environment conditions. Timestamps were obtained from reference node N_1 and the time offset compensated slave nodes N_2 and N_5 at T_{resync} and the instantaneous offset between the reference and slave nodes was input to the Kalman filter.

The process covariance (Q) and noise covariance (R) of input data was estimated at each iteration using a smoothed version of input data, as a surrogate for the true process state. KF based offset prediction for T_{resync} =1 and T_{resync} =30 s was performed offline on MATLAB and the prediction

Algorithm 3: Reference clock prediction using Linear prediction

Input: : Time offset between the reference and slave node $\theta_i, \theta_{i-1}, \theta_{i-2}, ..\theta_{i-p}$ measured from the corresponding timestamps

Input: : Prediction order p

Output : Predicted reference clock offset $\hat{\theta}_{i+1}$

1 Initialize array for storing previous offsets. Array size is equal to the size of prediction order.
2 Update the array with previous 'p' offset samples.
3 Calculate autocorrelation function of clock offsets.
4 Compute prediction coefficients $a_1, a_2, ...a_p$ using Levinson-Durbin iterative algorithm.
5 Calculate $\hat{\theta}_{i+1} = a_1\Delta T_i + .. + a_p\Delta T_{i-p}$
6 repeat from step 2 after T_{resync}

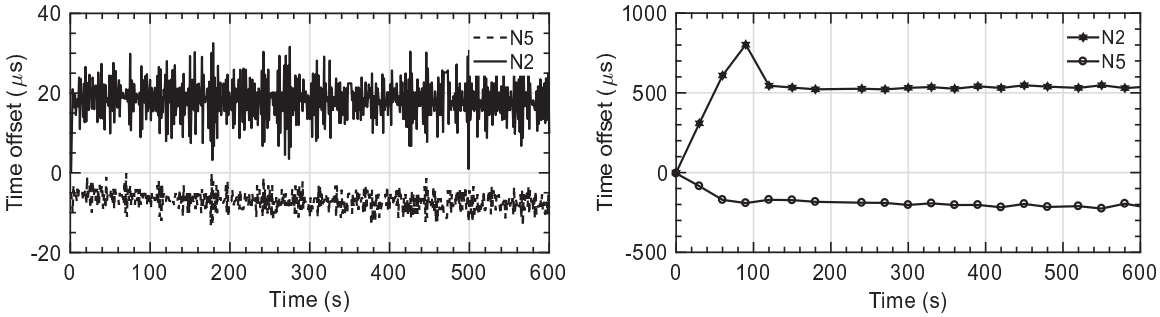

Figure 3.21. a) Prediction error (Measured time offset- Predicted time offset) for LP based prediction, $T_{resync} = 1second$. b) Prediction error for LP based prediction, $T_{resync} = 30seconds$.

error (Figure 3.22) was calculated. Similar to LP based prediction, the prediction error increases when T_{resync} is 30 s. The magnitude of prediction error is lesser for KF based prediction when compared to LP. KF technique also produce a non-zero steady state error proportional to the sampling interval like LP technique. When KF is implemented on nodes, it was observed that approximately 8 timestamps were required to obtain a converged result (Figure 3.22.b). For T_{resync}

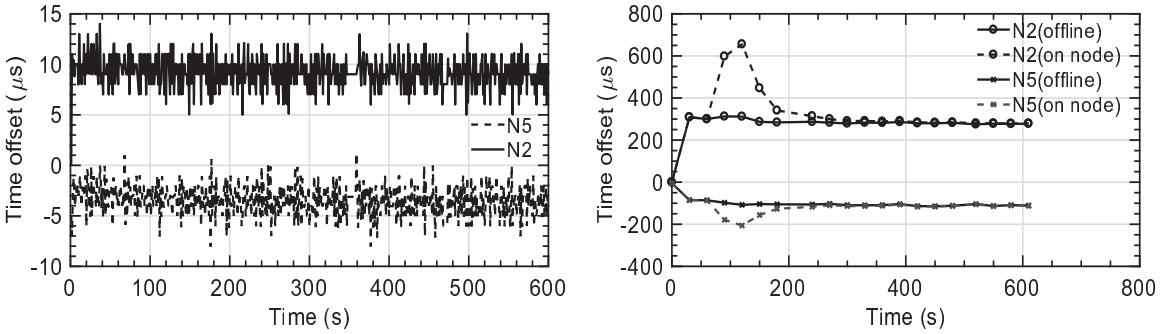

Figure 3.22. a) Prediction error (Measured time offset- Predicted time offset) for KF based prediction with T_{resync}=1 second. b) Prediction error for KF with T_{resync}=30 seconds.

= 30 s, converged result was obtained only after 4 minutes when the prediction was computed on nodes (Figure 3.22.b). When implemented on nodes without FPU, the convergence time can be in the order of several minutes even for single hop scenarios. Hence KF based techniques cannot be used for multi-hop, mobile networks where the convergence time is expected to be minimal.

Having experimentally verified the LR, LP and KF based prediction techniques it can be concluded

Algorithm 4: Reference clock prediction using Kalman Filter

Input: : Time offset between the reference and slave node, θ_{i-1} measured from the corresponding timestamps

Output : Predicted reference clock offset $\hat{\theta}_i$

1 $\theta_{predicted} = A * \theta_{i-1}; A = [1\ 0; 0\ 0]$.
2 $P_{predicted} = A * P_{n-1} * A^T + Q$
3 Innovation $\tilde{y} = Z_n - H * \theta_{predicted}; H = 1$; Z_n is the observation measurement;
4 Innovation Co-variance $S = H * P_{predicted} * H^T + R$
5 Kalman Gain $K = P_{predicted} * H^T * S^{-1}$
6 Update $\hat{\theta}_i = \theta_{predicted} + K * \tilde{y}$
7 Covariance update $P_i = (I - K * H)P_{predicted}$
8 repeat from step 1 after T_{resync}

that these techniques cannot be utilized for global synchronization, if the resynchronization interval is higher than few tenths of seconds even for static, single hop environments. With errors introduced by the wireless timestamping, the prediction error can be very significant. For the prediction, these techniques require timestamps at fixed intervals and hence the robustness of LP, KF algorithms to loss of timestamps needs to be further investigated. Moreover, the three techniques are prone to cumulative error in a multi-hop network and not suitable for a scalable mobile network. The variance in the prediction error is also higher for LR, LP and KF techniques (Figure 3.20, 3.21 and 3.22) when compared to the Swarm-Sync Framework (Figure 3.14). The convergence time of LP and KF techniques when implemented on nodes can be in the order of several minutes which is not appropriate for a mobile network.

3.5.2 Comparison with Consensus Based Algorithms

Popular consensus based synchronization algorithms are summarized in Table 3.4. CCTS and CMTS protocols are adaptations of ATS and MTS protocols respectively for clustered networks [51, 52]. CMTS protocol proposed in 2017 is based on maximum consistency theory. [54]

suggest that CMTS protocol is superior to CCTS in terms of accuracy, convergence speed and communication overhead. Hence, CMTS protocol is compared with Swarm-Sync framework in this section. In CMTS, the cluster heads initiate synchronization with a broadcast message. Cluster members respond to the cluster head with their clock compensation parameters and the recorded timestamps on the message reception. The cluster head, then adjusts its logical clock to the node with maximum skew compensation parameter as in Algorithm 5 and then broadcast its updated logical clock and clock compensation parameters to its cluster members. The cluster members, then update their logical clocks as in Algorithm 5. The cluster member nodes in general, can be considered as two-hops away from the node with maximum skew compensation parameter. The associated synchronization error for intra-cluster and inter-cluster synchronization is discussed in Section 3.3.

To verify the suitability of CMTS protocol for dynamic networks like swarm of robots, a test case was designed and simulated in MATLAB. Dispersion is a fundamental activity to be performed by robots in applications like mine detection, precision agriculture etc. A swarm of 30 robots were assigned the task of uniformly dispersing in an area of 10m x 20m starting from an initial position of (x=0, y=0). Each robot is randomly assigned a skew ranging from 1-10ppm and offset of 1-100μs. It is assumed that the robots can perform accurate MAC level timestamping. To manage the communication traffic, clustering is implemented. The network is divided into four clusters such that each cluster encompass the robots in an area of 5m x 10m. The number of robots in each cluster, the cluster heads and the overlapping nodes will vary from time to time untill uniform dispersion is achieved. For e.g., Figure 3.23.a depicts the scenario in which the dispersion is in progress whereas, Figure 3.23.b depicts the scenario in which the robots have reached their destination. At the start of dispersion, when all nodes belong to the same cluster, perfect synchronization is achieved for both CMTS and Swarm-Sync under the ideal simulation conditions presented above. Resynchronization of network is performed using CMTS and Swarm-Sync at an interval of 100 s. In Swarm-Sync, the frequency offset compensation is independent of other slave nodes or the network topology. Hence, for the ideal simulation conditions presented here, perfect synchronization is maintained even after several iterations of resynchronization and topological changes in network using Swarm-Sync framework. For CMTS, the variance in time represented by nodes increase with each iteration, even for ideal simulation conditions. The logical time represented by nodes for CMTS and Swarm-Sync after fourth and eighth iteration is depicted

Algorithm 5: Reference clock prediction using CMTS

Input: : Current ($\tau(t)$), and previous ($\tau(t-1)$) clock reading of nodes 'l' and 'j' of which one is cluster head and other is cluster member, skew $\hat{\alpha}(t_k)$ and offset $\hat{\beta}(t_k)$ compenation parameter

Output : Skew $\hat{\alpha}_l(t_{k+1})$ and Offset $\hat{\beta}_l(t_{k+1})$ compensation parameter

$\Delta\tau_j = \tau_j(t_k) - \tau_j(t_{k-1})$; $\Delta\tau_l = \tau_l(t_k) - \tau_l(t_{k-1})$.
if $(\hat{\alpha}_j\Delta\tau_j > \hat{\alpha}_l\Delta\tau_l)$ **then**
  $\hat{\alpha}_l(t_{k+1}) = \frac{\hat{\alpha}_j(t_k)*\Delta S_j}{\Delta S_l}$
  $\hat{\beta}_l(t_{k+1}) = \hat{\alpha}_j(t_k)\tau_j(t_k) + \hat{\beta}_j(t_k) - \hat{\alpha}_l(t_{k+1})\tau_l(t_k)$
else if $(\hat{\alpha}_j\Delta S_j = \hat{\alpha}_l\Delta S_l)$ **then**
  $\hat{\beta}_l(t_{k+1}) = max_{i=l,j}(\hat{\alpha}_j(t_k)\tau_j(t_k) + \hat{\beta}_j(t_k))_- \hat{\alpha}_l(t_{k+1})\tau_l(t_k)$
Update logical clock based on $\hat{\beta}_l(t_{k+1})$ and $\hat{\alpha}_l(t_{k+1})$

in Figure 3.24. The communication overhead for CMTS for intra-cluster synchronization with 'n' cluster members is presented in Table 3.4. Our studies indicate that despite large number of message exchanges, CMTS cannot provide accurate synchronization of mobile networks, even under ideal scenarios due to following reasons. Initially, when all nodes belong to the same cluster, the cluster head selects the node with maximum logical clock and adjust its logical clock with the node with maximum logical clock. Cluster members, then adjust their logical clock with the cluster head. As the dispersion progress, the nodes move out of communication range of initial cluster head and hence new clusters and cluster heads are formed. The node which had maximum skew with respect to initial cluster head may not be the one with maximum skew, when skew is measured with respect to new cluster head. This results in emergence of variance in time leading to cumulative error across several iterations. CMTS utilize two-way message exchanges and hence, the time taken for intra-cluster synchronization is dependent on the number of nodes in each cluster.

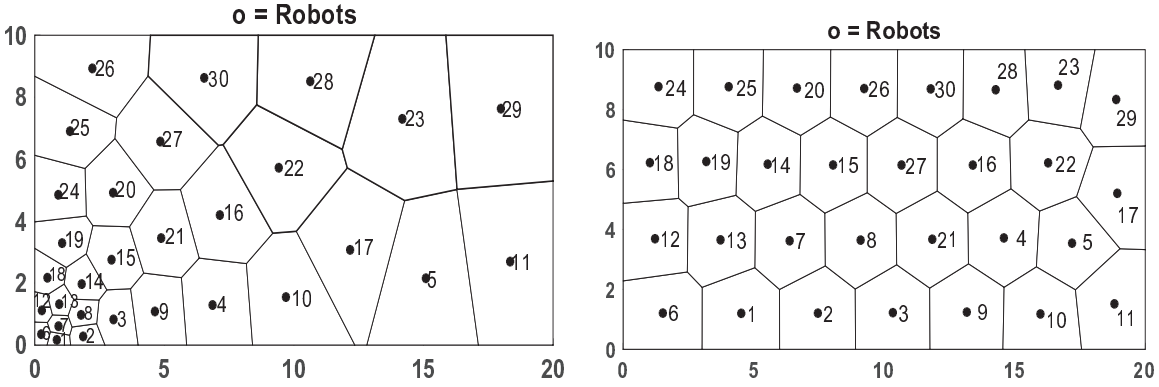

Figure 3.23. a) Robot dispersion- Intermediate Stage. b) Robots at their destination after dispersion.

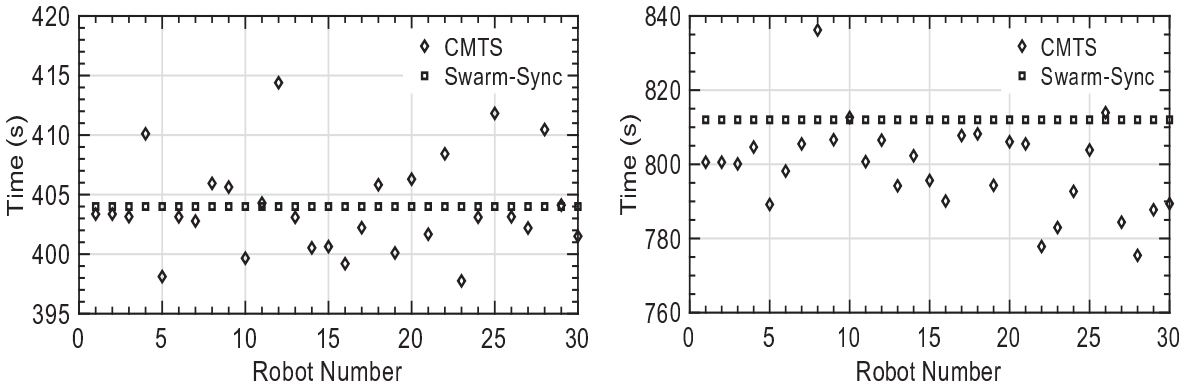

Figure 3.24. a) Time reported by robots after 4th synchronization iteration. b) Time reported by robots after 8th synchronization iteration.

In addition to this, in CMTS, convergence time of entire network depends upon the relative position of the cluster with maximum clock and other clusters in the network. Moreover, implementation of techniques to detect the cluster head changes, addition and removal of cluster members is required if CMTS protocol has to be utilized for mobile networks.

In Swarm-Sync protocol, each node in the network perform frequency offset compensation with respect to the reference node of the whole network, independent of the other slave or intermediate reference nodes. Slave nodes depend on intermediate reference nodes only for time offset compensation. The analysis on the accuracy of the proposed time offset compensation in Section 3.4.2 indicate that the time offset compensation is accurate. Hence the cumulative error is negligible or the time synchronization in Swarm-Sync framework can be considered as non-cumulative. The framework is flexible as it supports periodic or event based synchronization. Swarm-Sync framework can be utilized for continuous or post-facto synchronization with adaptive resynchronization intervals depending upon the application requirements. Since all nodes perform skew compensation with respect to a global reference node, loss of slave nodes in network will have negligible effect on the performance of time synchronization.

3.6 Conclusions

In this chapter, major desired characteristics of a time synchronization framework for scalable and dynamic networks such as swarm robotic systems or multi-robot systems is presented. The

sources of error in popular time synchronization techniques which utilize MAC level timestamping are identified, which justifies the need and significance of developing a new technique for time synchronization. A novel global time synchronization framework- 'Swarm-Sync' which utilizes the wireless network for communication among members of the swarm is presented. Important characteristics of the Swarm-Sync framework are as follows.

- Swarm-Sync framework implements time synchronization in two phases, 1) One-way time offset compensation and 2) Relative skew fingerprinting based frequency offset compensation. A unique characteristic of the proposed framework is that the framework utilizes only one-way messages for time and frequency offset compensation, as is desired for a scalable network.
- The synchronization framework can provide a bounded synchronization error throughout the network, thus leading to the easier design of other layers of the protocol stack like medium access control, routing, localization and task allocation.
- The Swarm-Sync framework is decentralized and topology independent, hence the robots can navigate freely across clusters or hops without necessitating resynchronization of the network. The framework does not impose any restrictions on the movement of the robots.
- One of the unique and the important feature of the Swarm-Sync framework is that it can provide a synchronization accuracy in the order of few hundreds of microseconds, for a resynchronization interval in the order of several minutes (average synchronization error of 110μs for resynchronization interval of 10 minutes) whereas the time synchronization protocols available in literature support a resynchronization interval in the order of few seconds only. The increase of resynchronization interval improves the energy efficiency of the system drastically and the communication bandwidth can be utilized for other robotic activities.
- The protocol is scalable as the number of message exchanges required/hop for synchronization is independent of the number of robots in the hop.
- The framework is experimentally validated on robots for single hop, multi-hop and for dynamic environments.

Comparison of Swarm-Sync framework with other two major class of protocols- consensus and prediction based protocols is presented in Section 3.5 and Table 3.4. The results obtained during experimental validation of Swarm-Sync framework as presented in Section 3.4.4 show that the framework is appropriate for the synchronization of swarm robotic systems and multi-robot systems.