A Dynamic Wireless Sensor Network Synchronisation Protocol

Jonathan Shannon, Hugh Melvin
College of Engineering and Informatics
NUI Galway
Galway, Ireland
shannon.jonathan@gmail.com, hugh.melvin@nuigalway.ie

Abstract *— This paper provides the reader with a comprehensive description of time synchronization within the wireless sensor network domain and details an alternative dynamic synchronization protocol which could result in reduced energy usage when compared to current static protocols.*

1. INTRODUCTION

Time synchronisation plays a critical role within the area of real-time distributed systems. While this has always been the case, the continuous development of more advanced and diverse systems place an ever-increasing demand on time synchronisation facilities in terms of reliability, precision requirements and energy consumption. Advances in integrated circuit design and fabrication techniques have given rise to the development of miniature computing devices which, with respect to size, have relatively sophisticated capabilities such as environmental sensing and radio communication. This technology in turn has spawned a new area in distributed data collection termed wireless sensor networks (WSN). While wireless sensor networks are nothing more than a number of dispersed miniature sensing devices connected via some wireless communication protocol, their applications are endless.

With regard to time sensitive WSN applications such as industrial automation, fire tracking or environmental monitoring applications, data collected within the WSN is generally of little use if the point in time it was acquired at is unknown. Time is crucial when it comes to the analysis of sensor data and in order to make accurate judgements or predictions about a system as a whole the issue of time synchronisation must be dealt with. While data time-stamping might be viewed as the main motive for time synchronisation, some other incentives include the time division multiple access (TDMA) and duty cycling techniques used by MAC protocols. In relation to duty cycling, the miniature computing devices used in WSNs are constrained in terms of energy and effective synchronous duty cycling techniques such as S-MAC and T-MAC provide a means to per long battery life.

In contrast to the internet domain, WSNs place extra constraints on time synchronisation techniques due to their composite node's limited processing, memory, communication and energy resources. As such, time protocols that employ frequent or large radio transmissions or complicated statistical techniques like those used by NTP are not practical. In addition a WSN is tightly coupled to its application meaning the application will have a large influence on the optimal synchronisation protocol employed.

At present the Flooding Time Synchronisation Protocol (FTSP) is the de-facto time synchronisation protocol employed by WSN applications. FTSP uses an elegant approach to implicitly build an ad-hoc synchronisation tree that can effortlessly adjust to changes in network topology. This together with its precision capabilities has led to its popularity. However, one limitation of FTSP is its un-optimal transmission rate. To achieve a specific level of accuracy FTSP transmits time messages at a periodic pre-calculated rate. In general, the number of message transmissions is based on worst case synchronisation performance of WSN clocks thus resulting in needless transmissions and energy wastage. In this paper we propose a dynamic synchronisation protocol that dynamically alters the transmission rate of nodes based on the clock stability of neighbouring nodes with the aim of reducing message transmissions and, thus, conserving energy.

The remainder of this paper is structured as follows. Section 2 outlines core synchronisation concepts. Section 3 describes some current static synchronisation protocols. Section 4 outlines a possible dynamic synchronisation protocol. Section 5 outlines current work and section 6 concludes the paper.

2. CONCEPTS

2.1 Sources of Clocks Error

To understand the primary source of clock error one most understand the workings of a clock oscillator. Oscillators used to drive computer clocks typically come in two varieties, namely, resistor-capacitor (RC) oscillators and crystal-controlled oscillators both of

which generate a sinusoidal output signal with a constant frequency. RC oscillators are composed of a network of resistors and capacitors, the structure and composition of which determines the frequency of the output signal. Crystal-controlled oscillators employ a crystal as the frequency-determining device within the oscillator. Crystals exhibit a characteristic known as the *piezoelectric effect* whereby mechanical forces produce electrical charges and vice versa. Some crystal substances exhibit this effect to a greater degree than others, namely quartz and Rochelle salt, and so they are more commonly employed in oscillators, the former more so than the latter.

With respect to time keeping, crystal-controlled oscillators provide superior frequency stability when compared to their counterparts and, as such, are commonly employed as the driving force of a computer system's real time clock while RC oscillators are generally used to generate the clock signal in integrated circuits. Crystals used in oscillators are cut and ground into thin wafers with a specific dimension and thickness that allows them to resonate at a desired frequency or *natural resonant frequency*. This frequency dictates the maximum resolution of the clock it will drive. Subsequently, the crystal is mounted in holders within the oscillator circuitry and voltage applied to it, the result of which is mechanical vibrations which in turn produce an output voltage at the crystal's natural resonant frequency.

The desired natural resonant frequency of a crystal is highly dependent on its manufacturing process. Given this fact, it is safe to assume that a crystal's actual frequency may differ from its stated frequency. Hence, with respect to a perfect crystal, most crystals will have a frequency offset or frequency error. This is generally true and in most cases manufactures will state the maximum error, which is typically less than 10 parts per million (ppm). The problem with frequency error becomes apparent when one considers the timer system the oscillator drives. If a crystal is stated as having a frequency of 32,768 Hz then 1 oscillation and likewise 1 clock tick will be interpreted by the system as an interval of 30.5 μs. If however the crystal has a frequency offset of -10 ppm then for every million oscillations of the crystal an extra 305 μs will have passed in real time. While this may not seem significant, it results in a time error of just under a second a day which is undesirable in many distributed applications.

The frequency error of an oscillator represents an important characteristic of a clock, namely, the *precision*. Another important characteristic of a clock is its *stability*. Stability is the ability of the clock to sustain a constant frequency over time. To understand how the frequency of a clock could change one must recognise that the frequency-determining device within the oscillator is influenced by physical conditions. Temperature and pressure changes cause materials to expand and contract and since we stated previously that the natural resonant frequency of a crystal is determined by its dimensions and thickness then it becomes apparent why. Naturally, voltage too has an impact but so too do long term processes such as material aging.

There are a number of approaches of ensuring good clock stability. The simplest of these is to select a crystal's cut such that it has the lowest temperature coefficient for its operating environment. A more elaborate approach is to control a clock's primary source of instability, that is, temperature, by placing it in a controlled environment such as an oven (termed OCXOs). The most effective approach is to replace the crystal clock with an atomic clock, albeit a very expensive solution. All of these approaches deal with the sources of clock error directly and entail altering the low level clock hardware. This is not always a feasible solution and so a cheap yet effective approach is required. A synchronisation algorithm offers just that, a software based procedure that alters the clock of a host based on a reference clock, thus ensuring two or more computing systems adhere to a common timescale.

2.2 Synchronisation Techniques

The majority of synchronisation protocols share the same basic design, that is, a server/reference distributes its understanding of the current time to a client/host via some communication protocol. The most simplistic technique, *unidirectional synchronisation*, which follows this basic design, has a host set its clock to the value received in a time message from a reference. Since the message latency is unknown and not factored in, this technique is only sufficient if the message latency from the reference to the host is lower than the accuracy requirement of the host. For increased accuracy, one must use *round-trip synchronisation*, (see Fig. 1) whereby a host wishing to be synchronised sends a time request to a reference node and records the transmission time ***ti***. The reference node replies with the reception time of the request message and transmission time of its reply message, ***ti+1*** and ***ti+2*** respectively. The host node uses these three timestamps along with the reception time of the reply message, ***ti+3***, to calculate the one-way message delay and, thus, determine its clock offset from the reference.

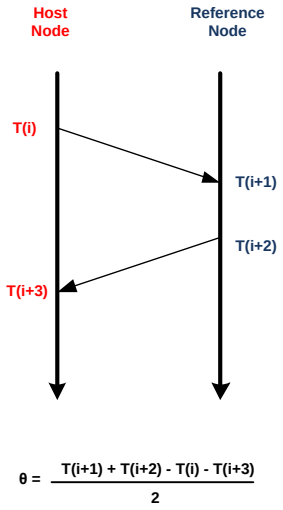

Figure 1: Round-trip synchronisation

Round-trip synchronisation operates on the assumption that the round-trip times of a message between a sender and receiver and vice versa are symmetric. This is unrealistic for many networks due to the non-deterministic delays associated with message traversal from a sender to a receiver (particularly with multi hops) and, as such, the result of the calculation of a host's time offset from its reference will differ from the true offset leading to a clock error.

A third technique, termed "*Reference Broadcasting*", takes advantage of the broadcast nature of certain networks. It involves three or more nodes, all of which lie within the broadcast range of each other. One node takes on the role of a *reference broadcast node* and broadcasts a beacon message. Each node records the reception time of the beacon message and subsequently exchanges its recorded timestamp with all other nodes. This approach results in the formation of relative time scales whereby each node can transform its timescale into that of every other node.

2.3 Sources of Synchronisation Error

All of the aforementioned synchronisation techniques are susceptible to different sources of synchronisation error. To recognise these sources one must understand the different components of a synchronisation message's latency which are categorised in [1] and [2]. These components are the *send time*, *access time*, *propagation time* and *receive time*.

The *send time* represents the time interval between the recording of a timestamp by the host and the delivery of a synchronisation message, containing that timestamp, to the network interface for transmission. The time taken to construct the message will be influenced by the underlying operating system. For instance, the process that constructs the message will be subject to some scheduling algorithm which may block it numerous times during its operation. Furthermore, additional delays may be incurred due to system call overheads.

The *access time* represents the delay incurred by the network interface while waiting to gain access to the communication medium. This is dictated by the Medium Access Control (MAC) protocol in use. For instance, the traditional wired Ethernet and wireless Ethernet 802.11 protocols use contention based approaches meaning a node may not transmit until the channel is clear, thus, high traffic loads will most likely lead to large access delays.

The *propagation time* represents the time taken for a message to traverse the communication link between sender and receiver. In a single hop network where the sender and receiver are connected directly by the same physical medium, the propagation time is dictated by the speed of light and as such is deterministic. If, however, the sender and receiver are connected via multiple network nodes, this time will be non-deterministic since the message will be subject to multiple queue and access delays at each intermediate node.

The *receive time* represents the time taken for the receiver's network interface to receive the message from the communication medium, decode it and notify the host application that it has arrived.

While all of the above delays are comprised of non-deterministic components, except for propagation time in the case of a single-hop network, much of this non-determinism can be eliminated by time stamping message transmission and reception at lower levels in the communication hierarchy. The non-deterministic delays associated with the send time and access time can be almost eliminated by recording the transmission time when the network interface gains access to the medium and placing this timestamp in the message as its been transmitted. Likewise, non-determinism associated with the reception time can be significantly reduced by having the network interface initiate the recording of a timestamp at the beginning of the reception of a message. This is possible with many modern sensor platforms which are comprised of microcontrollers with on board capture/compare hardware that captures the value of a timer when signalled to do so by a transceiver. Non-determinism associated with propagation time in multi-hop networks can be significantly reduced by ensuring intermediate nodes incorporate physical layer time stamping. This is performed by the *Precision time Protocol* (PTP) Transparent Clocks [3] which use specialised PTP time-stamping hardware installed at each node in the PTP hierarchy. While these approaches are effective at reducing or eliminating certain types of non-deterministic delays, they are not

always practical and, as such, techniques for mitigating the effects of these delays must be employed. The distinguished Network Time Protocol (NTP) [4] is a good example of a protocol that employs such techniques. It use data filtering and statistical techniques to mitigate errors and as such provides a practical alternative to other more elaborate approaches.

2.4 Dealing with Skew and Drift

As stated previously, a clock typically has a frequency that differs from its stated frequency, that is, a frequency offset/error which signifies its precision. Hence, two clocks generally run at different rates. Given this fact, it is obvious that a single synchronisation round is inadequate to keep two nodes synchronised. As such, nodes must periodically exchange time messages at a rate dependent on the accuracy requirement of the nodes and their relative frequency offset. If a node i wishes to stay synchronised within an error bound ε of a reference node r, then the interval Δ between synchronisation rounds must satisfy:

$$\epsilon i \leq \frac{\Delta}{fr - fi}$$

If the accuracy requirements of an application are quite stringent then the number of message exchanges required to keep a node's clock within the error bound may result in significant communication overhead. To minimise message exchanges a node can estimate the frequency offset between its clock and its reference and use this to determine its clock offset at any point in the future. The most widely used technique for estimating skew entails the use of regression analysis, or more specifically, linear regression. By obtaining multiple timestamps from a reference, a node can define a linear relationship between its clock and the reference clock thus allowing it to determine its clock offset relative to its reference at any particular point in time. This linear relationship is defined by the expression:

$$tr(t) = \alpha + \beta . ti(t)$$

where $tr(t)$ represents the time at the reference node r at true time t, $ti(t)$ represents the time at node i at true time t, α represents the time offset between the nodes at $ti=0$ and β represents the frequency offset between the nodes.

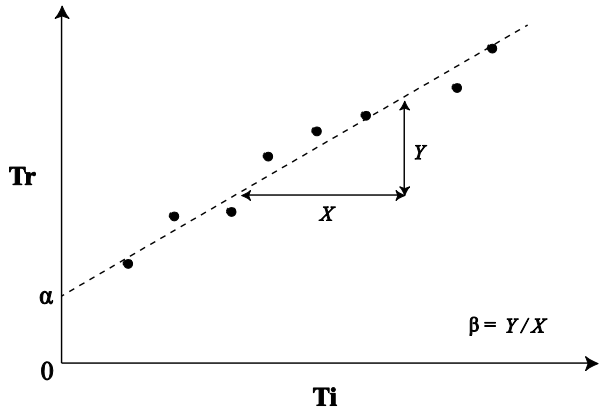

Figure 2: Linear Regression

The accuracy of this relationship is heavily influenced by the data that is used to postulate it. As such, non-deterministic latencies in a time message's traversal through a network will degrade the accuracy of this relationship. To remedy this, data filtering techniques can be used such that only credible data is used to define the relationship. NTP uses a data filtering algorithm that filters data based on the round-trip time of time messages. Data that originates from messages with lower round-trip times is regarded as been more accurate since it is more probable that the message did not encounter large asymmetric latencies.

Linear regression provides a simplistic yet effective means of defining a relationship between two clocks which in turn permits a reduction in the number of message exchanges between nodes. Nevertheless, linear regression models the relationship as a linear one which is an approximation. Referring to the previous section detailing the operation of an oscillator, it was stated that the frequency-determining device, namely the crystal, is susceptible to frequency changes caused predominantly by environmental factors and the extent of these frequency changes indicate the stability of a clock. These frequency changes imply that the relationship between two clocks is highly complex and cannot be accurately modelled as a linear one. Due to the difficulties that lie in accurately modelling this relationship and given that, in general, frequency changes occur slowly, it's is easier to model this relationship as a linear one over short time intervals and continue synchronisation rounds at an appropriate interval as long as synchronicity is required.

3. WSN SYNCHRONISATION PROTOCOLS

3.1 RBS Protocol

The RBS protocol uses the *reference broadcast synchronisation* technique to synchronise a cluster of

nodes located within the broadcast range of each other. Within a cluster, a particular node is elected the beacon node and periodically transmits a beacon message, the reception time of which is recorded by all nodes within the cluster. These nodes subsequently exchange their reception times and this enables them to determine the relationship between their clocks and, thus, construct relative time scales. Ultimately, a node can transform its time scale into that of every other node within the cluster.

Since RBS is only concerned with the reception time of messages, it eliminates the two sources of synchronisation error dominant with other synchronisation techniques, namely, the send time and access time. Consequently, the two leading sources of error with RBS are the propagation time and the receive time of a beacon message. Since nodes are all within broadcast range, the delays associated with the propagation of a beacon message are dictated by the speed of light together with the differences in the distance between nodes and are, thus, deterministic. While RBS does not directly determine this delay, it is considered negligible due to its insignificant contribution to the final clock error. The leading source of error, thus, becomes the receive time or more specifically those non-deterministic delays associated with the receive time of a beacon message.

Assuming the receive error can be reduced by having a node timestamp the reception of messages at the physical layer, the authors of RBS performed a study to characterise the receive error and, thus, verify their assumption that the maximum error would unlikely be greater than 1 bit time. Their assumption was based on the fact that a receiver must be synchronised to within 1 bit of a message in order to interpret it. The results confirmed their assumption, revealing a Gaussian distribution with a mean of 0, a standard deviation of 11.1 μs and a maximum of 53.4 μs, with 1 bit time being equal to approximately 52 μs given that the motes' transceivers operated at 19,200 baud. These results indicated that the accuracy achieved by a basic form of RBS could be improved further by having a beacon node broadcast more than one beacon message and have nodes average multiple offset estimations to determine the final offset relative to a neighbour. This procedure, as claimed by the authors, resulted in an improved accuracy from 11.1 μs to 1.6 μs in the case of an RBS network with 2 receivers utilising information from 30 beacon messages. While certainly an effective approach, it did not deal with clock skew and so the protocol was further refined by having nodes use multiple messages and linear regression to estimate their clock skew relative to their neighbours.

To extend RBS to operate in a multi-hop network comprised of nodes not within broadcast range of each other, nodes are organised into multiple clusters each of which contains one or more nodes that lie within one or more neighbour clusters. Thus, these nodes act as gateways translating time scales between clusters. The authors verify their assumption that the synchronisation errors introduced at each hop are independent and therefore the total path error should be equal to the sum of independent Gaussian variables or $\sigma\sqrt{n}$. Hence, the experiment they perform reveals a mean error of 3.68 μs in a 4 hop network.

An interesting scheme proposed by the authors is that termed "*post-facto synchronisation"*. This technique was born out of a desire to harmonise synchronisation protocols with the energy constraints inherent in sensors networks. Rather than continuously synchronises nodes, it is more efficient to synchronise them only when an event of interest occurred. Thus, after an event of interested occurs, synchronisation rounds commence to determine the offset and skew of a clock. Subsequently a node can extrapolate backward to determine its offset when the event occurred. While it is proposed, its effectiveness is not yet confirmed through experimentation.

3.2 TPSN Protocol

The *Timing-sync protocol for sensor networks* (TPSN) organises a network into a tree hierarchy the root of which acts as the source of time. Nodes are organised into levels based at the root and nodes at higher levels synchronise with nodes at lower levels using the round-trip synchronisation technique outline earlier.

Construction of the tree hierarchy is termed the *"Level Discovery phase"*. Initiation of this phase is performed by the root, a node that has been chosen explicitly because of its capabilities or by some automatic election process. The root assigns itself a level of zero in the hierarchy and then constructs a *level_discovery packet* containing its identity and level which it subsequently broadcasts. Reception of a level-discovery packet by a node allows it to determine its level and, in addition, initiates the broadcast of its own *level_discovery* packet. To deal with special cases such as when a node fails to determine its level due to packet collisions or when a new node joins a network after the level discovery phase, a node may transmit a *level_request* packet which provides the same functionality. Ultimately, this phase constructs a synchronisation hierarchy allowing the next phase to commence, that is, the *"Synchronisation Phase"*.

The synchronisation phase is initiated by the root when it broadcasts a *time_sync* packet. Reception of this packet by a node at level 1 triggers a synchronisation round between that node and the root. A synchronisation round complies with the round-trip synchronisation technique described earlier and sees a sender transmit a *synchronisation-pulse* packet to a receiver which replies with an *acknowledgement* packet. This procedure allows the sender to collect the information required to determine its offset relative to

the receiver and, thus, correct its clock. Synchronisation rounds at lower levels initiate synchronisation rounds at higher levels since message exchanges are overheard at higher levels.

Since TPSN uses the round-trip synchronisation technique it is vulnerable to all the sources of synchronisation error detailed earlier. To mitigate the effects of uncertainty associated with the send, access and receive times of a message, TPSN employs MAC layer time-stamping allowing the transmission and reception times of message to be recorded at the physical layer. In the TPSN study the authors compare TPSN to RBS with their own test-bed that consists of a zero hop network of Berkley motes. They claim a 2x better performance and attribute this to the 2 way message exchange involved in round-trip synchronisation that reduces the error. Their experiments reveal that the distribution of errors associated with the receive time is indeed Gaussian as claimed by the authors of RBS but the mean of the distribution for TPSN is half that of the RBS distribution. In relation to multi-hop networks, as with RBS, the errors at each hop are independent and Gaussian but since TPSN achieves a 2x performance the worst case n-hop mean will be 2*n more in RBS compared to TPSN.

With regards to clock skew, TPSN does not employ regression analysis to determine its value rather it relies on resynchronisation. Through numerous experimentations the authors fail to find motes that have a relative frequency offset of more than 4.75μs/s and so argue that resynchronisation would be relatively infrequent in the case of an application that requires an error bound in the vicinity of tens of milliseconds.

3.3 FTSP

FTSP organises a network into an ad-hoc synchronisation tree whereby an elected root acts as the source of time. Synchronisation of nodes is performed using the *unidirectional synchronisation* technique detailed earlier and operates such that senders distribute time to receivers.

The initial phase of FTSP involves the election of a root node which acts as the source of time for the network. The root election process is based on a simple algorithm whereby the node with the lowest ID is elected the root. Election of the root is followed by synchronisation rounds which are initiated by the root and occur at periodic intervals represented by ***P***. The choice of ***P*** is dictated by the accuracy requirements of the higher level application since lower values of ***P*** will result in more accurate estimates of the root node's time.

Synchronisation messages contain 3 key fields, namely, the *timestamp, rootID* and *seqNum* fields. The *timestamp* field represents the sender's time, the *rootID* field represents the address of the root as recognised by the sender and the *seqNum* is a sequence number, incremented solely by the root at the beginning of each synchronisation round.

The root broadcasts a synchronisation message which is received by all nodes within range. These nodes determine their offset and skew, correct their clock and rebroadcast the message placing their own time in the *timestamp* field. Thus, the time is effectively flooded through the network.

To manage redundant messages and ensure only the most recent messages are utilised by nodes, FTSP dictates that a node only accept messages if it contains a lower rootID than that recognised by the node or if it has a higher sequence number than the previously received message. In addition to managing redundant messages and increasing accuracy this technique also implicitly forms the resulting ad-hoc synchronisation structure.

The authors of FTSP describe in detail the constituents of non-deterministic message delay within a WSN. In the case where MAC time-stamping is employed, they decompose the send, access and receive time further in order to identify the main culprits of error associated with unidirectional synchronisation. The description entails the transfer of an idealised point in the message through the communication layers. They identify -

- *Interrupt handling time* – The time between when a transceiver raises an interrupt and when the microcontroller handles the interrupt by recording the time. This is non-deterministic but may be eliminated using capture registers.
- *Encoding time* – The time between the raising of an interrupt by the transceiver indicating the reception of a byte of data from the microcontroller and the encoding of this byte into electromagnetic waves. The encoding time and *Interrupt handling time* overlap since a timestamp must be generated before the message can be completely encoded. This time is deterministic.
- *Decoding time* – The time interval between the decoding of electromagnetic waves into binary data and the raising of an interrupt to notify the microcontroller. This is mostly deterministic but bit synchronisation errors can introduce jitter.
- *Byte alignment time* – This is a deterministic delay caused by the differences in the byte alignment between a sender and receiver.

By deconstructing the core message latencies further, the authors are able to identify the specific processes that cause the greatest error. These are the jitter and encoding times. The FTSP method of reducing the errors associated with these processes entails the use of timestamps recorded at each byte boundary of a message. A sender records these

timestamps at transmission time and normalises them by taking an appropriate multiple of the nominal byte transmission time from each one. By taking the minimum timestamp, those errors associated with the interrupt handling time are eliminated with high probability. The minimum timestamp provides a basis to deduce the interrupt error associated with all other timestamps, thus, allowing them to be corrected. By averaging these corrected timestamps, the error associated with the jitter of encoding can be reduced. This averaged timestamp is transmitted in the message and a similar process occurs at the receiver at message reception time, thus, reducing those errors associated with the decoding time. The final correction is that associated with the byte-alignment time. Since this delay is deterministic it is calculated directly using the transmission time and bit offset. Ultimately, FTSP mitigates most of the errors associated with message delays with the exception of propagation delay. This can be attributed to the synchronisation technique employed, or more specifically, the use of a single message for synchronisation. However, in this scenario, propagation time contributes to a very small proportion of error (less than 1 μs for up to 300 meters).

Similar to RBS, FTSP employs regression analysis, namely, linear regression to estimate the skew of a node. Knowledge of the skew reduces the communication overhead required to achieve a desired level of accuracy. In relation to protocol operation, each node contains a regression table that holds reference points related to the last N valid messages received. A node's skew value is updated with each new message reception using the newly received reference point and those contained in the table. The protocol also dictates that a node may only broadcast synchronisation messages when it has at least M entries in the regression table, thus, ensuring stability throughout the network.

	RBS	**TPSN**	**FTSP**
Topology	Cluster	Tree	Ad-hoc
Broadcast	Yes	Yes	Yes
Uni-directional	No	No	Yes
Bi- directional	No	Yes	No
Reference Broadcast	Yes	No	No

Table 1: Protocol characteristics

4. A DYNAMIC TIME SYNCRONISATION PROTOCOL

This paper proposes a dynamic time synchronisation protocol which in contrast to a static time synchronisation protocol automatically adjusts its transmission interval (δ) in order to achieve an application specific level of precision defined by an error bound (ε_{max}). A static synchronisation protocol such as FTSP requires initial configuration of the transmission interval before deployment in order to meet an application's time precision requirements. This transmission interval is influenced by the clock stability of the nodes and the conditions of their operating environment, as such, for optimal results the determination of an ideal transmission interval should be performed via an empirical analysis of the WSN's future operating environment. In general, the ambient temperature of a sensor is the biggest influencer and as such an environment subject to occasional temperature fluctuations will require that FTSP use an unnecessarily high transmission interval resulting in, at best, sub-optimal energy usage.

In general, with static time synchronisation algorithms, a node obtains a time message from a remote node, via a 2-way message exchange or a single broadcast, and uses this information to calculate its offset (θ). As explained in section 2.4, a node's clock has a frequency which differs from the nominal clock frequency and as such the offset between the two nodes will increase over time. Thus, a node must continuously obtain time messages from its reference node in order to keep its time within a specific threshold of the reference. If the precision requirements of an application are quite stringent then communication overhead between nodes can become unacceptably high.

The frequency offset or "clock skew" (λ) of a node from another can be corrected for directly. With two timestamps a node can estimate λ but, generally, to increase the accuracy of the estimate, a node obtains multiple timestamps from its reference and uses regression analysis to estimate λ. Once the node has an estimate of its skew relative to its reference it can calculate θ at any point in time. Ideally, this would allow a node to be permanently synchronised with little more than a few message exchanges, however, as discussed in section 2.4, the frequency of a clock does not remain constant and tends to change over time resulting in "clock drift" (φ).

To account for φ a node must continuously receive time messages from its reference at a rate dictated by the WSN application's precision requirement and the WSN operating environment. Subsequently, the node can correct for changing λ, and thus, produce a more accurate estimation of θ.

In contrast to a static time synchronisation protocol, a dynamic time synchronisation protocol accounts for

φ directly. A dynamic time synchronisation protocol observes the drift of a node and given an application's precision requirement determines when the node's clock will cross the acceptable error bound (ε_{max}). Thus, an optimal value for δ can be determined and the reference node notified so that a node receives a time message at the appropriate time to recalculate and update its changing skew.

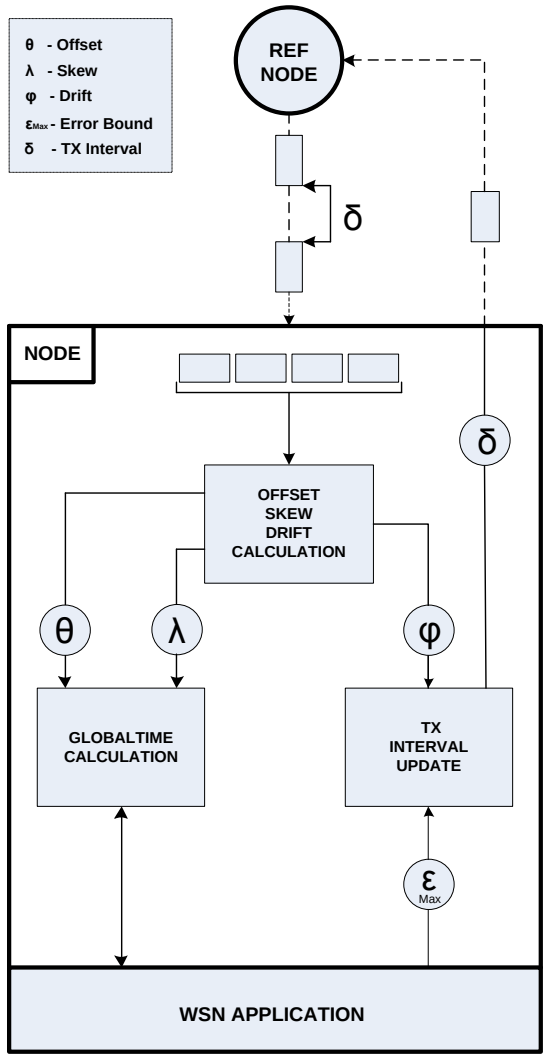

Figure 3: Dynamic Synchronisation Algorithm Operation

Fig. 3 outlines the general flow of control of a dynamic time synchronisation protocol. A node wishing to synchronise receives a time message from its reference node (***r***) which contains a time stamp T_{ri}. Two sequential time stamps (T_{ri}, T_{ri+1}) are used to determine the offset and skew of the node's clock with respect to the reference clock. A third timestamp, T_{ri+2}, is used to determine the rate of change of skew, that is, **φ**. Knowledge of **φ** allows one to estimate the point in time at which the current skew value will lead to an estimation of the reference time with an error that exceeds the error bound of the WSN application. The node must receive a new time message from its reference node before the aforementioned point in time so that it can update λ. Thus, the ideal time between message receptions, that is, the transmission interval of the reference node, δ, is determined and the reference node is notified via a notification message.

5. CURRENT STATUS

The aforementioned protocol is currently being Implemented and tested on the telosb platform. Its performance, in terms of precision and communication overhead, relative to current static protocols is being deduced.

6. CONCLUSIONS AND FUTURE WORK

This paper has provided the reader with a comprehensive explanation of time synchronisation and its relevance to the wireless sensor network domain. A detailed explanation of synchronisation concepts and techniques has been provided. In addition, a number of current wireless synchronisation protocols have been described. This information has served as a foundation for the description of a dynamic synchronisation protocol. The latter aims to reduce message transmissions, and thus energy usage (relative to similar static protocols) by setting an informed transmission interval, based on the required synchronisation level for the network.

References

[1] H. Kopetz, W. Schwabl. Gloab *Time in distributed real-time systems.*

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

[3] IEEE-1588 - *Standard for a Precision Clock Synchro-nization Protocol for Networked Measurement and Control Systems.*

[4] Network Time Protocol (NTP).

[5] S. Ganeriwal, R. Kumar, M.B. Srivastava, *Timing-sync protocol for sensor networks,* in Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys), Los Angeles, CA, November 2003.

[6] M. Maroti, B. Kusy, G. Simon, A. Ledeczi, *The flooding time synchronization protocol*, In Proceedings of the Second ACM Conference on

Embedded Networked Sensor Systems (SenSys), November 2004.

[7] J. Shannon, H. Melvin, *Synchronisation Challenges for Wireless Networks*, Digital Technologies (DT), November 2009.