Probabilistic Clock Synchronization Service in Sensor Networks

Santashil PalChaudhuri, Amit Saha, and David B. Johnson
Department of Computer Science, Rice University
Houston, Texas

Abstract

Recent advances in technology have made low cost, low power wireless sensors a reality. Clock synchronization is an important service in any distributed system, including sensor network systems. Applications of clock synchronization in sensor networks include data integration in sensors, sensor reading fusion, TDMA medium access scheduling, and power mode energy saving. However, for a number of reasons, standard clock synchronization protocols are unsuitable for direct application in sensor networks.

In this paper, we describe a probabilistic service for clock synchronization that is based on the Reference Broadcast Synchronization protocol. In particular, we use the higher precision of receiver-to-receiver synchronization, as described in Reference Broadcast Synchronization protocol. We extend this deterministic protocol to provide a probabilistic bound on the accuracy of the clock synchronization service, allowing for a tradeoff between accuracy and resource requirement. We derive expressions to convert service specifications (maximum clock synchronization error and confidence probability) to actual protocol parameters (minimum number of messages and synchronization overhead). We also extend this protocol for maintaining clock synchronization in a multihop network.

I. Introduction

Recent advances in technology have made low-cost, low-power wireless sensors a reality. Sensor networks formed from such sensors can be deployed in an ad hoc fashion and cooperate to sense and process a physical phenomenon. An important feature of sensor network is energy efficiency for extending the network's lifetime, as each sensor has a finite (non-zero) battery source.

As in any distributing computer system, clock synchronization is an important service in sensor networks. Sensor network applications can use synchronization for data integration and sensor reading fusion. A sensor network may also use synchronization for TDMA medium access scheduling, power mode energy savings, and scheduling for directional antenna reception. In addition to energy awareness, sensor networks present some unique problems that make it difficult to directly apply traditional network clock synchronization approaches.

The error in clock synchronization comes mainly from the non-deterministic random time delay for a message transfer between two nodes. Kopetz et al. [7] first characterized this message delay. In some recent work, this non-determinism has been reduced to provide tighter bounds on the clock synchronization error [13], [3]. Our work is based on the Reference Broadcast Synchronization (RBS) by Elson et al. [3], providing an analytical way to convert service specifications to protocol parameters.

The need in sensor networks, however, is to provide the best possible clock synchronization under existing circumstances, given the limited resources of the nodes and the network in the system. In various situations, for example when the system energy is extremely low, it might not be possible to provide highly accurate clock synchronization, since higher accuracy typically requires more message transfers and processing as part of the synchronization

protocol. The accuracy needed in clock synchronization is variable, depending on the higher layer application requirements. If the need for determinism can be relaxed, probabilistic guarantees often suffice for the needs of an application, while allowing optimal use of resources. Probabilistic guarantees provide better accuracy in *most* cases as compared to deterministic accuracy in *all* cases. Also the failure probability of achieving a certain accuracy can be bounded. Quality of Service (QoS) in networks makes extensive use of this concept and gives probabilistic guarantees by allowing resources to be allocated on the basis of expected aggregate demand. Previous work [2], [1] uses this concept for providing probabilistic clock synchronization in distributed systems.

Also, clock synchronization might not be necessary at all times, but only during sensor reading integration. In that case, providing clock synchronization all the time will be a waste on the limited available resources of sensors. The sensor clocks may thus be allowed to go out of sync, and then re-synchronize only when there is a need for synchronization, thus saving resources.

We use the above concepts for our probabilistic clock synchronization service, which we build by extending the deterministic Reference Broadcast Synchronization (RBS) protocol. The remainder of this paper is organized as follows. In Section II, we describe the various concepts of clock synchronization in detail and motivate the design principles used in building our algorithm. In Section III, we survey the related work in this area. Section IV describes the RBS protocol, along with our improvements. Section V evaluates the performance of the protocol. Section VI extends this protocol for multihop sensor networks. Finally, we conclude in Section VII.

II. Design Principles

In the paper describing the Network Time Protocol (NTP), Mills [10] defines the various terms used in clock synchronization. The *stability* of a clock is how well the physical clock can maintain a constant frequency. *Accuracy* refers to how well the maintained time is true to the standard time. The *offset* of two clocks is the actual time difference between them, and the *skew* is the frequency difference between them. To *synchronize frequency* means to adjust the clocks to run at the same frequency, and to *synchronize time* means to set their time at a particular epoch to be exactly the same. To *synchronize clock* means to synchronize the clocks in both frequency and time. In this paper, our algorithm will *synchronize clocks*, i.e., it will synchronize both frequency and time.

A. Traditional versus Sensor Network Synchronization

Elson and Romer [4] point out the basic features that are common to clock synchronization protocols in general. These are as follows:

- Synchronization has to be provided over a connectionless messaging protocol between nodes
- Synchronization has to be achieved with the exchange of clock information between clients and one (or a few) servers
- The protocol has methods to reduce or predict the non-determinism in message delivery and processing
- The protocol assumes the presence of an algorithm to update local clock at client based on received value

NTP [10] is scalable, robust to failures, self-configuring, and has various properties that are needed in the sensor network world. However, wireless sensor networks pose a number of challenges beyond traditional network systems. Elson and Romer [4] describe the differences quite exhaustively. We reiterate these differences, as they are important from the design point of view:

- **Energy Constraint:** Energy efficiency is very important for sensor networks as opposed to traditional networks. The basic assumptions of the nodes having steady power supply, listening to network for free, and network transmissions being inexpensive, do not hold for sensor networks. In wireless networks, listening to the network interface and sending packets take significant energy. Also, the CPU is powered down in sensor networks for much of the time, breaking the traditional assumption of the availability of CPU whenever necessary. Synchronization is not required to be maintained at all times, but only when required by the application.
  Hence, to do traditional synchronization in a sensor network would require the CPU and network interface to be powered up for significant amounts of time, thereby having a large resource overhead.
- **Tunable Accuracy:** Traditional synchronization protocols try to achieve the highest degree of accuracy possible. The higher the level of accuracy required, the higher the resource requirement. The accuracy of synchronization required depends on the application requirement. Using resources for high accuracy even if lower accuracy is enough for the application wastes the limited resources of sensor networks.
  Therefore, there is a need for a trade-off between resource requirements and accuracy, depending on the need of the application and resource availability of the system.
- **Non-determinism:** Sensor networks are dynamic systems with considerably higher rate of failures of the individual nodes than in tradition networks. The nodes can also be mobile. This gives rise to a greater non-determinism for communication delays than that present in traditional networks. Typically, NTP has the need for some manual configuration of peers and upstream nodes.
  Hence, the synchronization protocol needs to be more robust to failures and also to the greater variability in communication delay.
- **Multihop:** Most of the typical synchronization protocols have a highly accurate clock present in the LAN, such that all the nodes in the system can directly exchange messages. Sensor networks span many hops, with higher jitter.
  Hence, algorithms for sensor network clock synchronization need to have some sort of localization to reduce this error, as well as some other means of achieving multihop synchronization even in the presence of high jitter.
- **Server-less:** Traditional protocols have specified servers, with multiple accuracy levels which are sources of accurate time. Sensor networks do not have any external infrastructure present and can be large in scale. Maintaining global time scale in this network is thus harder, if no external broadcast source of global time such as GPS is present.
  Elson et al. [4] propose that each node maintain an undisciplined clock, augmented with the relative frequency and phase information of its neighbors. We also use this approach in our work.

B. Theoretical Bounds on Clock Synchronization

There have been various theoretical results that have been proven regarding clock synchronization. These analytical results and their consequences are useful when designing a clock synchronization protocol:

- From the causality property in a system, the ordering of events can be formally stated. If an event *a* occurs before another event *b*, then *a* should happen at an earlier time than *b*. Let $C_i(a)$ be the clock value of process *i* when event *a* occurs. Then it can be formally stated that:

  If *a* and *b* are events in process *i*, and event *a* occurs before event *b*, then $C_i(a) < C_i(b)$.

  Lamport [8] showed that, when the value of a clock needs to be adjusted, it always has to be set forward and never back. Setting the clock back could cause the above condition to be violated. Hence, in the ideal system, the slower clocks needs to be adjusted to the value of the fastest clock, for all clocks to be synchronized. This restriction will also maintain the partial ordering of the events.

- It is useful to have a bound on the best accuracy achievable in any system, such that no bound lower than that is specified. Srikanth et al. [12] have shown that for any synchronization algorithm, even in the absence of faults, the bound on the rate of drift of logical clocks from real time is at least as large as the bound on the rate of drift of physical clocks. In the presence of faults — such as message losses and node failures — the accuracy of logical clocks becomes even worse.

- Most clock synchronization algorithms proposed in literature try to guarantee a deterministic upper bound on the clock skew. Lundelius et al. [9] derived a theoretical limit on the best achievable clock skew that deterministic algorithms can guarantee. They show that the upper bound on clock skew that can be deterministically guaranteed by any clock synchronization algorithm can be no smaller than $(d_{max} - d_{min})(1 - 1/n)$, where n is the number of nodes in the system, and d_{max} and d_{min} are the maximum and minimum value of message delays in the system, respectively. Significantly smaller bounds on the upper bound of clock skew can be achieved if the condition of determinism is loosened. If the deterministic guarantee is replaced with a probabilistic guarantee, where the probability of failing can be bounded, it might suffice for many applications.

  This probabilistic guarantee is specially significant for sensor networks, as sensor networks inherently assume a level of redundancy present in the number of nodes and are fairly robust to failures. This approach might entail saving a significant amount of resources in the sensor network, while giving some guarantees. We thus develop a probabilistic clock synchronization protocol for sensor networks, basing our work done previously by Christian [2] and Arvind [1].

C. Sources of Time Synchronization Error

The biggest source of error in synchronization algorithms stems from non-determinism in message delivery latency. In an effort to reduce this non-determinism, we review the sources of this latency. Kopetz et al. [7] have characterized the message delivery latency into four distinct components (Elson et al [3] have adopted the same approach):

- **Send Time:** The time spent at the sender to build the message. This time includes the time for kernel processing, context switches, and system call overhead incurred by the synchronization application and is hence highly variable depending on the current system load.
- **Access Time:** This is the delay incurred when waiting in the network interface for access to the transmission channel. This time depends on the MAC protocol in use and its methods to handle congestion. Typical wireless MAC protocols like IEEE 802.11 [6] networks exchange RTS/CTS before the actual exchange of the message. Depending on the congestion in the network, this waiting time is the most significant in terms of the total delay latency.
- **Propagation Time:** This is the time needed for the message to propagate from sender to receiver over the wireless medium. If the sender and receiver are in the same broadcast region, this time is typically very small. This time can be approximately calculated by dividing the distance between the sender and receiver by the speed of light. This time is negligible compared to the other delays.
- **Receive Time:** This is the time needed for processing at the receiver's network interface. It denotes the time difference between the actual reception of the packet and actually informing the application of the packet's arrival, allowing the application to process the packet. If the time-stamp of reception can be done at a suitable low level, this delay can be made very small, and more importantly, deterministic.

Most clock synchronization algorithms go to great lengths to reduce the above non-determinism in message delivery. A significantly different approach has been taken in the CesiumSpray system [13]. CesiumSpray takes advantage of the inherent broadcast nature of the wireless medium. The Send Time and Access Time are unknown and highly variable. However, for a set of receivers listening to a common sender, those times are identical for all of the receivers. The only variable time is the Propagation Time and Receive Time, which are much smaller in value. This approach entails synchronizing a set of receivers with each other, in contrast to synchronizing with the sender. We use this idea to reduce significantly the sources of error in our clock synchronization protocol.

D. Models of Clock Synchronization

There are many different types of clock synchronization. Each type has its usage. They are as follows:

- **Global clock:** There is a precise global time, UTC, which is maintained by atomic clocks in standard laboratories. Traditional internet clock synchronization algorithms try to maintain this global time in all computer systems. Maintaining this time in sensor networks is significantly harder. Sensor networks also do not typically need this strict clock synchronization. In this paper, we can provide this service with a certain degraded accuracy if a GPS is available. But, maintaining this time is not the thrust of this work.
- **Relative clock:** This is the relative notion of time within the sensor network. Each node each synchronized with every other node with a time which might be totally different from UTC. This suffices for most of the applications of clock synchronization that we have described earlier. In this work, we provide this synchronization with an accuracy which is bounded with a tunable confidence probability.

- **Relative notion of time:** Time can also be maintained between nodes, not with real time, but with some logical notion. This logical notion need not match with physical clock. For example, two nodes might do some processing 10 units after some event has occurred, unit being not necessarily the same as seconds. This sort of time is very often enough for a variety of applications. But, providing the previous type of clock synchronization also automatically provides this type of clock synchronization. So, we indeed provide this notion of clock synchronization.
- **Physical ordering:** In many cases precise times might not be important, but what is important is the ordering of events. If the system can state whether an event occurred before or after another, that is enough. This type of synchronization simply involves ordering of events in some partial or total order. Having previous clock synchronization types automatically provide this type. However, we note that this might be significantly cheaper in terms of resources to achieve. So, if application need only this type of synchronization, alternate means should be used to preserve scarce resources.

Another classification is in terms of the initiator of synchronization procedure. They are as follows:

- **Always On:** In this model clock synchronization between nodes is always present. Many applications like TDMA scheduling might need this model. This model is the model of traditional clock synchronization protocols. But, maintaining this model might present significantly higher overhead, if the applications require clock synchronization only rarely.
- **Sensor Initiated:** In this model the sensor nodes decide whether to have synchronization or not. They synchronize between themselves or a subset of the nodes, whenever necessary. This model is useful when the need for synchronization is not required frequently. But it might entail a certain degree of latency before synchronization can be achieved. So the applications will need to tolerate this latency.
- **Outsider Initiated:** This is similar to the previous model. But here the initiator of clock synchronization is somewhere outside the sensor network, for example a control center. This model degrades to the previous model though, once the message to start synchronization has reached from the outsider to the necessary sensor nodes. So, this is essentially same as the previous one and we will not treat this as different.

In this work, we provide all three types of synchronization above, depending on the system need. The Always On model has higher overhead than the other models.

III. Related Work

A. Traditional Clock Synchronization Protocols

As mentioned earlier, most traditional clock synchronization protocols share the same basic design: a connectionless messaging protocol, exchange of clock information between client and server(s), methods to reduce the effects of random non-deterministic communication delay, and a method to upgrade the client time based on the information from the server. NTP [10] is widely deployed in the internet, being scalable, robust and having good performance. It consists of various levels (or stratums) of servers in a hierarchy providing synchronization to the clients which are

leaves in the tree. These protocols cannot be applied directly to sensor networks because of the differences pointed out in Section II-A.

B. Wireless Clock Synchronization Protocols

There have been a few synchronization protocols which are specifically for wireless or ad hoc networks. Romer [11] proposed a scheme for sparse ad hoc networks. The algorithm does not synchronize the computer clocks of the nodes, but generate time-stamps using which the unsynchronized clocks transform the message time-stamp. As a message moves from hop to hop, each node transforms the message timestamp to it's local time-stamp with some introduced error. This error increases with the number of hops. Also, the protocol uses round trip delays, the calculation of which will have the typical errors associated with estimating round trip time.

Huang et al. [5] showed that the 802.11 MAC time synchronization protocol is not scalable for large number of nodes. They proposed a simple modification to the MAC protocol which maintains synchronization among nodes in a single broadcast region.

C. Receiver-Receiver Synchronization

A couple of previous works use a completely different approach than the traditional approaches. They synchronize a set of receivers amongst themselves. This reduces much of the message delivery latency non-determinism associated with traditional protocols. CesiumSpray [13] was the first work to use this idea. It is a hybrid external/internal synchronization protocol. It uses a two-level hierarchy to improve scalability.

Reference Broadcast Synchronization (RBS) [3] is the work on which our paper is based. Least-squares linear regression is used to find the relative frequency of the clocks. RBS uses post-facto synchronization to synchronize two nodes clocks by extrapolating backwards to estimate the phase offset at any previous time. It extends the work to do multihop clock synchronization. Our paper is essentially similar to this work in terms of protocol in the single-hop case, other than some minor improvements.

However, their paper does not contain any analysis on the number of reference broadcasts necessary, or the frequency of reference broadcasts. We analyse these issues and provide a probabilistic bound on the maximum error. We also relate this probability with the number of reference broadcasts required. RBS also has more overhead in terms of exchanging information between the receivers. It assumes a single broadcast region for the sender and all the receivers, which is not the case. Two receivers, lying in the broadcast range of a sender, might not be able to exchange messages since they might be our of range of each other. For multihop synchronization, unlike RBS, our protocol does not require all sensor nodes to be within one hop of at least one sender.

D. Probabilistic Clock Synchronization

There are a few synchronization protocols which are probabilistic in nature. The clock skew that a probabilistic protocol guarantees has a probability of invalidity associated with the guarantee. However, the probability of invalidity can be bounded. All the probabilistic algorithms proposed are for server-client architecture, where the clients try to synchronize with the clock of the server. This is fundamentally different from our approach where we synchronize

amongst receivers. However, we use the idea of the probability analysis techniques proposed in the following two papers.

The idea of probabilistic protocol was proposed by Christian [2]. He realized that guarantee cannot be provided when unbounded message delays are possible or messages can be lost. Hence, his principle is to retry sufficient number of times to read the clock of another process with a given precision with probability as close as desired. However, there are some fundamental limitations to the accuracy that can be achieved. His algorithm uses an algorithm to read the remote server clock within the specified precision. It repeats this reading attempts till a reply comes back within a certain desired interval of time. The lower the round trip time for a reading attempt and its reply to come back, the higher the accuracy achieved in reading the clock of the remote server. The average number of messages to reach synchronization is $2/(1-p)$, where p is the probability of failure of message delivery within a fixed period of time. This process is repeated k times, such that the probability of reaching synchronization is $1-p^k$. By choosing a large enough k, the probability can be made arbitrarily close to 1.

Arvind [1] proposed another probabilistic synchronization protocol. It has two main parts — the Time Transmission Protocol (TTP), by which the clock value of the sender is read by the receiver, and Probabilistic Clock Synchronization (PCS), which uses this value to adjust the receivers clock. In TTP, the sender sends a sequence of n synchronization messages, each having the sender timestamp. The receiver adjusts the clock value using these n timestamps. The synchronization procedure is repeated every interval of time. He analytically showed the minimum number of messages necessary for achieving a given maximum synchronization error. This paper has less overhead than Christian's paper, if the physical medium is considered to be broadcast.

IV. Single-Hop Clock Synchronization

In this section, we describe the technique of receiver-receiver synchronization behind our protocol. Next, we present the extension of this technique to probabilistic synchronization for use within a single broadcast region. We then analyse it mathematically to find the probabilistic bounds.

A. Receiver-Receiver Synchronization Error

The sources of non-deterministic error in message latency have been described in Section II-C. In CesiumSpray [13] and RBS [3], this non-deterministic error was significantly reduced by synchronizing among the receivers, instead of synchronizing between the sender and receivers. In receiver-receiver synchronization, the only remaining sources of non-determinism are the Propagation Time and Receive Time. Propagation Time is very small, considering the range of the sensors and the speed of radio waves. Receive Time is much more deterministic and can be bounded by time-stamping the receiving time at the hardware level [7], [3]. If the total error previously was ϵ_{sr}, for sender-receiver synchronization, and the error in this case is ϵ_{rr}, for receiver-receiver synchronization, then $\epsilon_{sr} >> \epsilon_{rr}$. These two different ways of synchronization can be seen in Figure 1, showing the reduction of error in receiver-receiver synchronization.

The way receiver-receiver synchronization works is as follows. The sender sends a reference pulse at any time. Each receiver marks when it received the reference pulse according to its local clock. All the receivers exchange with

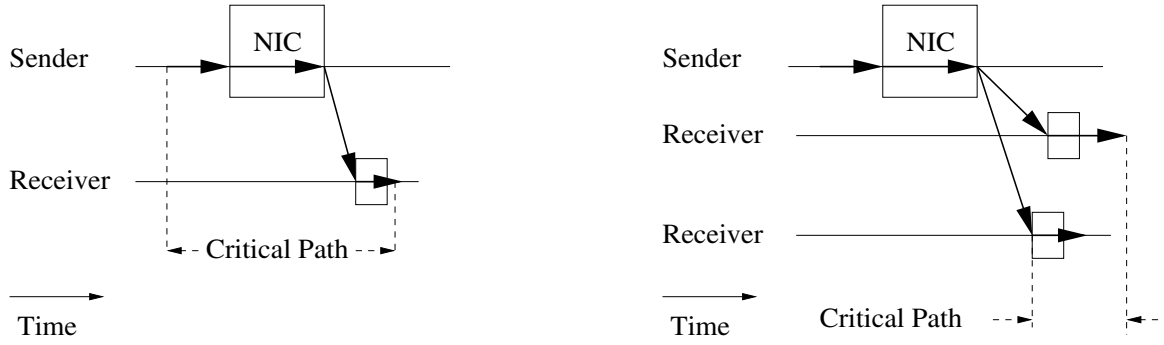

Fig. 1. Types of Clock Synchronization [3]: Sender-Receiver Synchronization and Receiver-Receiver Synchronization

each other the time of reception of the reference pulse. Since each receiver assumes that the pulse should have been received by all other receivers at approximately the same real time, a receiver *A* is able to estimate the offset of its clock with respect to another receiver *B* which has exchanged information with *A*. The number of reference packets can be increased in order to get better synchronization.

Elson et al. [3] have found the distribution of the synchronization error among a receiver set. Multiple pulses are sent from the sender to the set of receivers. The difference in actual reception time at the receivers is plotted. As each of these pulses are independently distributed, the difference in reception times gives a Gaussian (or normal) distribution with zero mean. In this figure, ϵ_{rr} would be the any point in the x-axis, with the pdf of the distribution given by the corresponding y-value. The probability of ϵ_{rr} being 0 is maximum, it being the mean. The probability falls exponentially as the value of ϵ_{rr} increases.

Given a Gaussian probability distribution for the synchronization error, we can easily calculate the relation between a given maximum error in synchronization and probability of actually synchronizing with an error less than the maximum error. If the maximum error that we allow between two synchronizing nodes is ϵ_{max}, then the probability of synchronizing with an error $\epsilon \leq \epsilon_{max}$ is given from Gaussian distribution property.

$$P(|\epsilon| \leq \epsilon_{max}) = \frac{\int_{-\epsilon_{max}}^{\epsilon_{max}} e^{-\frac{x^2}{2}}\, dx}{\sqrt{2\pi}}$$

So, as the ϵ_{max} limit is increased, the probability of failure $(1 - P(|\epsilon| \leq \epsilon_{max}))$ decreases exponentially. We use this observation in the analysis below.

B. Description of the Protocol

In this section, we present our protocol, extending the deterministic RBS protocol to provide probabilistic clock synchronization. The frameworks for providing external synchronization with UTC and for providing relative synchronization among the nodes are different. In this paper, we concentrate on providing relative synchronization, as it often is sufficient for most sensor network applications.

- For external synchronization, we assume the availability of GPS in a subset of the nodes. These nodes will be senders of synchronization messages. The sensor nodes will synchronize with these GPS receivers using any of

the many sender-receiver probabilistic algorithms proposed previously [1], [2]. Obviously, the synchronization error will be more than receiver-receiver synchronization because of the reasons pointed out in the previous section.

- For relative synchronization, we use receiver-receiver algorithm. The basic framework for providing the *Always On* or *Sensor Initiated* model is the same. So, we present and analyse them together, pointing out the differences as and when necessary. The steps in this protocol are described next. The subset of nodes chosen as senders is random among the set of sensor nodes. If the relative density of sensor sender nodes and all sensor nodes is above a threshold, and if we assume uniform distribution of the sensor nodes, we can assume the presence of sender sensors in every broadcast region. But, this assumption is not necessary, as we will see in section VI, when we extend our protocol for multihop synchronization.

The following happen for every sender sensor in a single-hop broadcast region. A particular sensor being in the broadcast region of two senders will do all of the steps below separately for each sender. When synchronization is necessary in a sensor-initiated model, the sensors needing synchronization send out a *REQUEST*. This request is broadcasted till it reaches a sender sensor, which starts a cycle of the algorithm. In the Always On model, each sender sensor periodically (period is determined analytically in the next sub-section based on requirements) starts this cycle.

1. A Sender broadcasts n reference packets to its neighbors. Each packet contains two counters, one showing a cycle number, and another the reference packet number in the current cycle. The interval between each packet is fixed and greater than some minimum, such that they are independent of each other.
2. Each receiver records the time according to its own local clock, when each of these reference packets are received. Using these time-stamps, the receiver uses linear regression to fit a line on these data. The slope of the line will approximate the relative clock skew between the receiver and the sender.
3. Each receiver sends back to the sender, a packet containing the slope of the line and one point on that line. The sending back of these packets are jittered over an interval so that the packets sent back by different receivers have less chance of colliding with each other.
4. The sender composes all these slopes together, and broadcasts a packet containing its relative clock skew slope to all the receivers who have replied back.
5. Each receiver after receiving this packet, can now calculate its own slope relative to all the receivers in the broadcast region of a a particular sender. So, for every pair of receivers, within the broadcast region of the sender, the clock skew and clock offset is now known with some synchronization error. The Send Time and Access Time error is factored out when calculating this relative slope, as that error is the same for any two receivers. The only error present will be that due to propagation time and receive time.

This ends the working of the protocol for one cycle. It is repeated periodically for the Always On model.

C. Mathematical Analysis

The preceding section shows how our algorithm keeps the clocks of sensor nodes synchronized. In this section we will analyse the probabilistic guarantee of achieving the desired synchronization skew. We will derive how to convert the service specifications (maximum clock synchronization error and confidence probability) to actual protocol parameters (minimum number of messages and synchronization interval).

- **Synchronization overhead:** The error among the receivers is a normal distribution, as described in Section IV-A. Normal distribution makes it much easier to study a distribution statistically. The following theorem gives a relation between the synchronization error and its associated probability, with the message overhead.

  For *n* synchronization pulses from the sender, the receivers exchange their observations via the sender. As explained earlier, the slope of the skew between the receivers is found by a least square linear estimation using the *n* data points. The calculated slope of the skew has an associated error in it. This error is the difference in phase between the calculated slope and the actual slope. As the points have a Gaussian distribution, this error can be calculated. As the number of data samples, *n*, increases, this error reduces.

  To prove the theorem, the following lemmas will be used.

  In Lemma 1, it is proved that the characteristic function of the sum of independent random variables is a product of the characteristic function of each random variable.

  Lemma 1: Let $Z = X_1 + X_2$ with $f_{X_1}(x), f_{X_2}(x)$ and $f_Z(z)$ denoting pdf's of X_1, X_2, and Z respectively, X_1 and X_2 are independent random variables. Then, $\Phi_Z(\omega) = \Phi_{X_1}(\omega)\Phi_{X_2}(\omega)$.

  Extending the result by induction, $\Phi_Z(\omega) = \Phi_{X_1}(\omega)\Phi_{X_2}(\omega)\cdots\Phi_{X_n}(\omega)$.

  In Lemma 2, the characteristic function of the sum of independent random variables is derived, if the random variable's are Gaussian distributions. This derivation uses Lemma 1.

  Lemma 2: The characteristic function of the summation of *n* Gaussian distributions is:

  $$\phi_Z(t) = e^{i\mu t - \frac{\sigma^2 t^2}{2n}}$$

  Proof : Given that, $Z = \frac{X_1+X_2+\cdots+X_n}{n}$

  $$= \frac{X_1}{n} + \frac{X_2}{n} + \cdots + \frac{X_n}{n}$$

Now, $f(x) = \frac{e^{-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2}}{\sqrt{2\pi}\sigma}$

Therefore, $f(y) = f\left(\frac{x}{n}\right)$

$$= \frac{ne^{-\frac{1}{2}\left[\frac{\left(\frac{x}{n}-\frac{\mu}{n}\right)n^2}{\sigma}\right]^2}}{\sqrt{2\pi}\sigma}$$

$$= \frac{e^{-\frac{1}{2}\left[\frac{y-\frac{\mu}{n}}{\frac{\sigma}{n}}\right]^2}}{\sqrt{2\pi}\left(\frac{\sigma}{n}\right)}$$

The characteristic function of a Gaussian distribution X_i is,

$$\phi_{X_i}(t) = e^{i\mu t - \sigma^2\frac{t^2}{2}}$$

So, the characteristic function of $\frac{X_i}{n}$ is,

$$\phi_{\frac{X_i}{n}}(t) = e^{i\frac{\mu}{n}t - \left(\frac{\sigma}{n}\right)^2\frac{t^2}{2}}$$

From the previous lemma, the characteristic function of the summation of the independent random variables will by the product of the characteristic function of the individual random variables. Therefore, the characteristic function of Z is

$$\phi_Z(t) = \left[e^{i\frac{\mu}{n}t - \left(\frac{\sigma}{n}\right)^2\frac{t^2}{2}}\right]^n$$

$$= e^{(i\frac{\mu}{n}t - \frac{\sigma}{n^2}\frac{t^2}{2})n}$$

$$= e^{(i\mu t - \sigma^2\frac{t^2}{2n})}$$

$$= e^{(i\mu t - (\frac{\sigma}{\sqrt{n}})^2\frac{t^2}{2})}$$

Using the Gaussian distribution $N(\mu, \frac{\sigma^2}{n})$ of Z, the synchronization error function can be found. Theorem 1 gives the relation between the synchronization bound with associated probability, and the minimum number of messages necessary. As the number of messages is the overhead of the protocol, this also gives the resource requirement to achieve a specified synchronization bound. This is the error in synchronization at the time when

synchronization is done.

Theorem 1: $P(|\epsilon| \leq \epsilon_{max}) = 2\, erf\left(\frac{\sqrt{n}\,\epsilon_{max}}{\sigma}\right)$

where ϵ is the clock skew at synchronization, ϵ_{max} is the maximum specified clock skew at synchronization point, n is the minimum number of synchronization messages to guarantee the specified error, and σ is the variation of the distribution.

Proof: For a Gaussian distribution $N(\mu, \sigma^2)$, the pdf, $f(x) = \frac{e^{-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2}}{\sqrt{2\pi\sigma^2}}$

Hence, for the Gaussian distribution of Z, $f(z) = \frac{e^{-\frac{1}{2}\left[\frac{x-\mu}{\frac{\sigma}{\sqrt{n}}}\right]^2}}{\sqrt{2\pi(\frac{\sigma}{\sqrt{n}})^2}}$

The probability that the error, ϵ lies within ϵ_{max} is given by:

$$P(|\epsilon| \leq \epsilon_{max}) = \frac{\int_{-\epsilon max}^{\epsilon max} e^{-\frac{(x-\mu)^2}{2(\frac{\sigma}{\sqrt{n}})^2}} dx}{\sqrt{2\pi(\frac{\sigma}{\sqrt{n}})^2}}$$

Putting $\mu = 0$,

$$P(|\epsilon| \leq \epsilon_{max}) = \frac{2\sqrt{n}\int_0^{\epsilon max} e^{-\frac{nx^2}{2\sigma^2}} dx}{\sqrt{2\pi\sigma^2}}$$

$$= \frac{2\sqrt{n}\sigma\int_0^{\frac{\epsilon max}{\sigma}} e^{-\frac{ny^2}{2}} dy}{\sqrt{2\pi\sigma^2}} \quad \text{[Putting } y = \tfrac{x}{\sigma}\text{]}$$

$$= \frac{2\int_0^{\frac{\sqrt{n}\epsilon max}{\sigma}} e^{-\frac{z^2}{2}} dz}{\sqrt{2\pi}} \quad \text{[Putting } z = y\sqrt{n}\text{]}$$

$$= 2\, erf(\tfrac{\epsilon max\sqrt{n}}{\sigma}) \quad \text{[Where, } erf(x) = \tfrac{\int_0^x e^{-\frac{t^2}{2}} dt}{\sqrt{2\pi}}\text{]}$$

- **Synchronization interval:** The previous theorem specified the minimum synchronization error, given the number of messages. That was the error in synchronization precisely when synchronization for done. In the next theorem, the relation between the synchronization period and the maximum specified clock skew is shown. Given a maximum value for clock skew, a time period is derived within which re-synchronization has to be done.

Theorem 2: $\gamma_{max} = \epsilon_{max} + (T_{sync} + \sigma_{max})\rho$

where γ_{max} is the maximum allowable synchronization at any point in time, T_{sync} is the time period between synchronization points for the Always On model (time period of validity for Sensor Initiated model), ρ is the

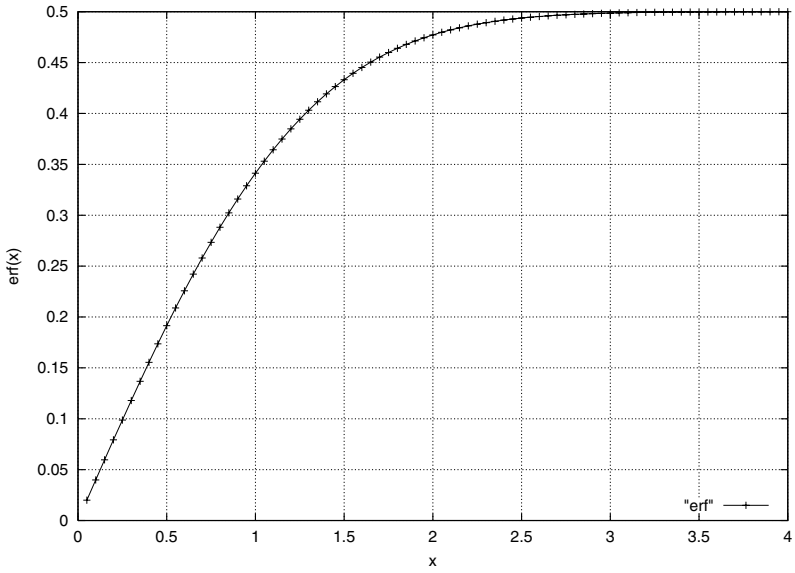

Fig. 2. Error function of x versus x

maximum drift of the clock rate, and σ_{max} is the maximum delay (after the synchronization procedure has been started) in the time values of one receiver reaching another receiver.

Proof : In our protocol, successive re-synchronization attempts are never spaced more than $T_{sync} + \sigma_{max}$ time units, since once the re-synchronization has started, a receiver might receive the vector corresponding to the times reported by another receiver after, at most σ_{max} time. Thus, due to clock drift, the maximum clock skew that can arise between two receiver clocks, in this time interval is $(T_{sync}+\sigma_{max})\rho$. Moreover, our synchronization protocol guarantees that after a synchronization attempt, the maximum skew between two receivers is ϵ_{max} with an invalidity probability of $1 - P(|\epsilon| \leq \epsilon_{max})$. Hence, the total maximum skew that can develop between two receiver clocks is given by $\gamma_{max} = \epsilon_{max} + (T_{sync} + \sigma_{max})\rho$.

V. Evaluation

The error function $erf(x)$ is a standard function and has been plotted in Figure 2. The function is defined for negative values of x, yet since $erf(-x) = -erf(x)$, we plot the function only for positive values of x. As shown in Figure 2, the function asymptotically reaches the value of 0.5. By Theorem 1 in the previous section, we have the *probability of achieved error being less than a maximum specified error* as

$$P(|\epsilon| \leq \epsilon_{max}) = 2\ erf\left(\frac{\sqrt{n}\,\epsilon_{max}}{\sigma}\right)$$

We can derive a relationship between n i.e. the number of messages and the achieved probability P. This relationship is plotted in Figure 3. Each of the three different curves corresponds to the $\frac{\epsilon_{max}}{\sigma}$ ratio of 0.5, 1.0, and 2.0. Thus, given an ϵ_{max} and the probability P of achieving an error $\epsilon \leq \epsilon_{max}$ we can find the required number of messages. As expected, if the ratio increases we need lesser number of messages in order to achieve a desired probability i.e.

$\frac{\epsilon max}{\sigma}$	Probability	Number of messages
0.5	0.95	16
0.5	0.99	28
0.5	0.999	44
1.0	0.95	4
1.0	0.99	7
1.0	0.999	11
2.0	0.95	1
2.0	0.99	2
2.0	0.999	3

TABLE I

VARIATION IN PROBABILITY AND NUMBER OF MESSAGES FOR DIFFERENT VALUES OF $\frac{\epsilon max}{\sigma}$

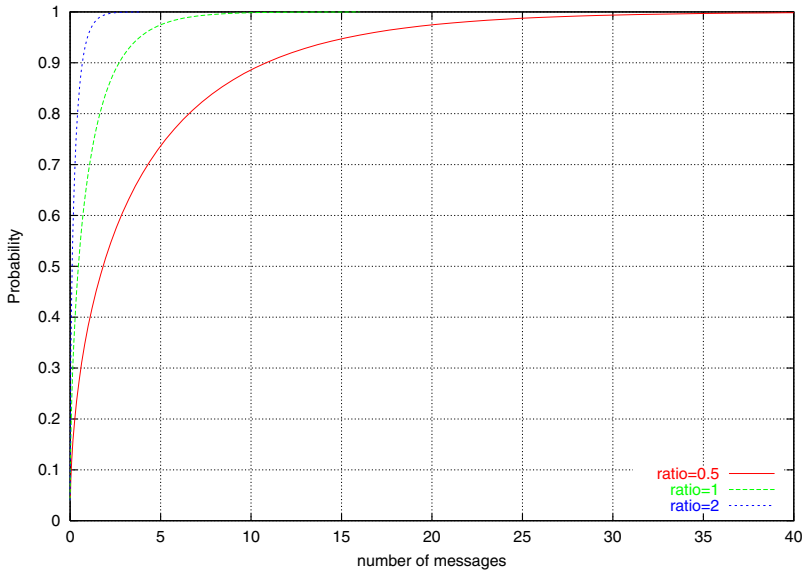

Fig. 3. Probability of achieved error being less than a maximum

$n \propto \left(\frac{\sigma}{\epsilon_{max}}\right)^2$. This proportionality suggests that once the ϵ_{max} has been specified, the number of messages required is very sensitive to the standard deviation.

From Table I, it is clear that for lower values for the ratio of $\frac{\epsilon max}{\sigma}$ the number of messages required gets quite large. For example, to achieve a probability of 0.99% with $\frac{\epsilon max}{\sigma} = 1$, the algorithm requires 7 messages whereas for $\frac{\epsilon max}{\sigma} = 2$, the algorithm requires only 2 messages.

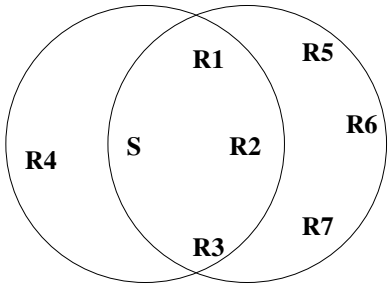

Fig. 4. Multihop clock synchronization

VI. Multihop Synchronization

In the previous section, we presented our algorithm for achieving probabilistic clock synchronization among all receivers that are within a single wireless hop of a sender. In this section, we extend our algorithm to create a probabilistic clock synchronization service between receivers that may be multiple hops away from a sender. This multihop extension is in contrast to the multihop extension proposed by Elson et al. [3] that assumes that all sensor nodes are always within a single hop of at least one sender. Also, in order for two sensor nodes (present in the broadcast regions of two different senders) to be synchronized, their algorithm requires the existence of a node that is within the broadcast region of both the senders. Our algorithm does not make any such assumptions, and sensor nodes in our algorithm are allowed to be multiple hops away from a sender and still be synchronized with all other nodes within the nodes transmission range.

A. Protocol Description

For this protocol, we consider senders at various levels. A sender which does not need any synchronization (like the sender in Section IV) is called a sender at *level 0*. A sensor node which is within the broadcast region of a sender at level 0 can behave as a sender in order to synchronize sensor nodes which are two hops away from the sender at level 0. Such a sender is called a sender at *level 1*. This can be extended for multiple hops from the sender at level 0.

The receivers which are within the broadcast region of the sender at level 0 get synchronized in the same way as described in Section IV. Once these receivers get synchronized among themselves, each receiver starts behaving as a sender at level 1 and starts sending n reference broadcast packets. In order to avoid collision of reference broadcast packets, a sender at level 1 delays the transmission of its n reference packets until it does not hear the reference packets of any other sender at level 1. These packets are received by all sensor nodes which are within the broadcast region of a sender at level 1.

Consider the scenario presented in Figure 4. Nodes R1, R2, R3 and R4 are within the broadcast region of the sender S. Using the single hop synchronization protocol nodes R1, R2, R3 and R4 are synchronized among themselves. Now suppose R2 gets to be the first node to send the reference broadcast, that is R2 starts behaving as a sender at level 1. By a similar synchronization procedure, R1, R3, R5, R6 and R7 get synchronized among themselves. Now suppose R6 needs to send a message to R4. The message would have to be routed through a node which is synchronized with R6, say R3. The assumption here is that due to the relative high density of sensor nodes, a node, such as R3 as shown

in Figure 4, will exist in the broadcast region of two senders; the two senders might be at at the same level or they might be separated by a single level. Now since R3 is synchronized with R4 R3 can transform the time reported by R6. Finally since R3 is synchronized with R4, R4 can transform the time reported by R3. Hence, all along the routing path of the message suitable time transformations can be performed.

However, from the description of the protocol it seems that the protocol will have a very high overhead since the protocol essentially floods the entire network with reference packets. This can be easily fixed by causing time synchronization to be sensor initiated so that a sender (at any level) broadcasts reference packets only if the sender receives a request for synchronization from a sensor node in the local broadcast region of the sender. This ensures that a node does not broadcast reference packets if there is no sensor node in its local broadcast region that can listen to the reference packets.

B. Mathematical Analysis

The mathematical analysis of this protocol is similar to the mathematical analysis presented in Section IV-C. If ϵ_{max} is the maximum error between two receivers present in the broadcast region of a sender then the maximum error possible between two sensor nodes which are k hops apart is $k \cdot \epsilon_{max}$. This can be proved as follows:
We shall prove by mathematical induction on the number of time transforms performed on an actual reported time.

Base Case: If there is a single time transform performed on the message then by the analysis of Section IV-C the maximum error possible between the actual time reported and the transformed time is ϵ_{max}. Thus, if the actual time is t_0 then the reported time will lie in the interval $[t_0 - \frac{\epsilon_{max}}{2},\ t_0 + \frac{\epsilon_{max}}{2}]$.

Induction Hypothesis: After k time transformations on the actual reported time t_0, the transformed time can lie in the interval $[t_0 - k \cdot \frac{\epsilon_{max}}{2},\ t_0 + k \cdot \frac{\epsilon_{max}}{2}]$ since by induction hypothesis the maximum error after k time transformations is $k \cdot \epsilon_{max}$.
Suppose we perform one more time transform on the time $t_r \in [t_0 - k \cdot \frac{\epsilon_{max}}{2},\ t_0 + k \cdot \frac{\epsilon_{max}}{2}]$. Then the transformed time will lie in the interval

$$\left[t_0 - k \cdot \frac{\epsilon_{max}}{2} - \frac{\epsilon_{max}}{2},\ t_0 + k \cdot \frac{\epsilon_{max}}{2} + \frac{\epsilon_{max}}{2}\right] = \left[t_0 - (k+1) \cdot \frac{\epsilon_{max}}{2},\ t_0 + (k+1) \cdot \frac{\epsilon_{max}}{2}\right]$$

Thus, the error is $2(k+1) \cdot \frac{\epsilon_{max}}{2} = (k+1) \cdot \epsilon_{max}$. This proves our induction hypothesis.

Moreover, if p is the probability that the maximum error between two receivers within broadcast region of a sender is ϵ_{max}, then p^k is the probability that $k \cdot \epsilon_{max}$ is the *maximum error* between two receivers which are k hops apart. Since $p < 1 \Rightarrow p_k < p$, the larger the number of time transformations, the lesser the probability of staying within an error bound. However, if we consider the *average error* over a single hop to be $err_{avg} = \frac{\epsilon_{max}}{2}$ then the average error over k hops will be $err_{avg} \cdot \sqrt{k}$ (since variance is additive). This implies that the error propagation is sublinear.

VII. Conclusion

In this paper, we have presented and analyzed a probabilistic service for clock synchronization in sensor networks. This protocol is based on the earlier deterministic RBS protocol [3]. The RBS protocol uses the concept of *receiver-*

receiver synchronization to achieve better synchronization bounds than traditional synchronization protocols. Our protocol, being a probabilistic one, is not bound by the limit given by Lundelius et al. [9]. In our protocol, we periodically send n messages to probabilistically keep the clocks of the sensor network within a specified error bound.

Our contributions in this paper are three-fold:

- We extended the deterministic RBS protocol to provide probabilistic clock synchronization, allowing the synchronization service to trade off the synchronization accuracy and the resources used by the protocol.
- We analyzed the protocol to derive expressions for synchronization overhead (in terms of number of messages) and synchronization interval. For example, we show that to achieve a synchronization error less than σ of the distribution with a confidence probability of 99%, the protocol requires 7 synchronization messages per interval. We show how to convert service specifications (maximum clock synchronization error and confidence probability) to actual protocol parameters (minimum number of messages and synchronization interval).
- We also extend the RBS protocol to handle multihop clock synchronization in which all nodes need not be within single-hop range of a clock synchronization sender. Those nodes not within single-hop range will receive synchronization with some degradation of the bounds, when compared to single-hop case.

In future work, we intend to implement our probabilistic clock synchronization service in a testbed of sensors to show its performance experimentally. Finally, we would like to build actual applications for sensor networks that utilize our clock synchronization service.

REFERENCES

[1] K. Arvind. Probabilistic Clock Synchronization in Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):474–487, May 1994.

[2] Flaviu Cristian. Probabilistic Clock Synchronization. *Distributed Computing*, 3:146–158, 1989.

[3] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, Massachusetts, December 2002.

[4] Jeremy Elson and Kay Romer. Wireless Sensor Networks: A New Regime for Time Synchronization. In *First Workshop on Hot Topics in Networks*, Princeton, New Jersey, October 2002.

[5] Lifei Huang and Ten-Hwang Lai. On the Scalability of IEEE 802.11 Ad Hoc Networks. In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Lausanne, Switzerland, June 2002.

[6] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, New York, 1997.

[7] Hermann Kopetz and wilhelm Ochsenreiter. Global Time in Distributed Real-Time Systems. Technical Report 15/89, Technische Universitat Wien, Wien Austria, October 1989.

[8] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[9] Jennifer Lundelius and Nancy Lynch. A New Fault-Tolerant Algorithm for Clock Synchronization. In *Proceedings of the Third annual ACM Symposium on Principles of Distributed Computing*, pages 75–88, Vancouver, Canada, 1984.

[10] David Mills. Internet Time Synchronization: The Network Time Protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[11] Kay Romer. Time Synchronization in Ad Hoc Networks. In *Proceedings of the Second ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Long Beach, California, October 2001.

[12] T. K. Srikanth and Sam Toueg. Optimal Clock Synchronization. *Journal of the ACM*, 34(3):626–645, July 1987.

[13] P. Verissimo, L. Rodrigues, and A. Casimiro. CesiumSpray: A Precise and Accurate Global Time Service for Large-Scale Systems. *Journal of Real-Time Systems*, 12(3):243–294, May 1997.