A Survey of Time Synchronization Techniques for Sensor Networks
Aashish K. Manucha
Binghamton University
amanuch1@binghamton.edu

Abstract
Time synchronization is key for any distributed systems like 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. Sensor networks show some unique characteristics in the scope and lifetime, as well as the time and energy required to achieve precision synchronization, that make it difficult to directly apply traditional network clock synchronization approaches. Therefore, existing methods may not be suitable and need to be amended for these needs. The goal here is to help to discuss various synchronization regimes presently available according to the application requirement.

1. Introduction
Time Synchronization is vital for Wireless sensor networks (WSNs), which consist of large populations of wirelessly, connected nodes capable of computation, communication and sensing. Sensor nodes cooperate in order to merge individual sensor readings into a high-level sensing result, such as integrating a time series of position measurements into a velocity estimate. The physical time of sensor readings is a key element in this process called data fusion.

The need in sensor networks is to provide the best possible clock synchronization under existing circumstances, given the limited resources of the nodes and the network in the system, for example a limited battery source. A global clock synchronization method is also needed in a sensor system to help process and analyze the data correctly and predict future system behavior.

The process gets complicated because if we want to achieve accurate clock synchronization, many message transfers and processing may be required as a part of the synchronization protocol draining on the battery.

The clocks in a sensor network can be inconsistent due to several reasons. The clock may drift due to environment changes, such as temperature, pressure, battery voltage, etc. The nodes in a sensor network may not be synchronized well initially, when the network is deployed. The sensors may be turned on at the different times and their clocks may be running according to different initial values. The results of events on specific sensors may also affect the clock. For example, a sensor may be busy handling message transmission or sensing tasks, and miss clock interrupts.

Time synchronization methods for sensor networks must therefore also be mindful of the time and energy that they consume. Various methods have been presented over the recent time period for Time synchronization. This paper will present a survey of some of those, and present a comparison of the regimes and their applicability.

2. Problem Definition
Clock synchronization in ad-hoc (and thus wireless) sensor networks pose challenges that are significantly different from those in infrastructure-based networks. The clocks in a sensor network can be inconsistent due to several reasons. The clock may drift due to environment changes, such as temperature, pressure, battery voltage, etc. This has been a research topic in the operating system and Internet communities for many years. The nodes in a sensor network may not be synchronized well initially, when the network is deployed. The sensors may be turned on at the different times and their clocks may be running according to different initial values. The results of events on specific sensors may also affect the clock. For example, the Berkeley Mote sensors may miss clock interrupts and the chance to increase the clock time value when they are busy handling message transmission or sensing tasks.

‘Non-determinism’ is a primary source of time synchronization error. Kopetz and Schwabl characterize it as having four distinct components [11]:

- Send Time. It is the time spent to construct the message.
- Access Delay. It is the delay incurred in waiting to access the transmit channel and is specific to the MAC protocol.
- Propagation Time. This is the transit time of the message from the sender to the receivers. It is dominant in wide-area networks.
- Receive Time. This is the time for the receiver’s network interface to receive the message from the channel and notify the host.
3. Various Approaches

Existing time synchronization algorithms vary primarily in their methods for estimating and correcting for these sources of error. Here in, are discussed some of the methods.

3.1 Simple Synchronization

Sichitiu et al.[1] have presented a synchronization algorithm for drifting clocks with a unique method for processing collecting data. The two closely related algorithms in this paper are called mini-sync and tiny-sync. The main contribution of the paper is the development of a simple algorithm, which delivers accurate offset and drift information together with tight, deterministic bounds on them. The other features are accuracy, Low computation and storage complexity and insensitivity to communication errors.

The Algorithm: Consider two wireless nodes 1 and 2, with their hardware clocks $t_1(t)$ and $t_2(t)$ respectively, where $t$ is the Coordinated Universal Time (UTC). The oscillator’s frequency depends on the ambient conditions, but for relatively extended periods of time (minutes -hours) can be approximated with good accuracy by an oscillator with fixed frequency:

$$t_i(t) = a_i t + b_i;$$

(1)

where $a_i$ and $b_i$ are the drift and the offset of node i’s Clock. This algorithm is suitable for any type of network. It is especially useful in wireless sensor networks, which are typically extremely constrained.

The available computational power and bandwidth and have special requirements for high precision synchronization. In general $a$ and $b$ will be different for each node and approximately constant for an extended period of time. From (1) it follows that $t_1$ and $t_2$ are linearly related:

$$t(t) = a_{12} t + b_{12};$$

(2)

The parameters $a_{12}$ and $b_{12}$ represent the relative drift and the relative offset between the two clocks respectively. If the two clocks are perfectly synchronized, the relative drift is equal to one and the relative offset is equal to zero. Assume that node 1 would like to be able to determine the relationship between $t_1$ and $t_2$. Node 1 sends a probe message to node 2. The probe message is time stamped right before it is sent with $t_o$. Upon receipt, node 2 timestamps the probe $t_b$ and returns it immediately (we will shortly relax this requirement) to node 1 which timestamps it upon receipt $t_r$. Fig. 1 depicts such an exchange. The three time-stamps ($t_o; t_b; t_r$) form a data-point, which effectively limits the possible values of parameters $a_{12}$ and $b_{12}$ in (2). Indeed, since $t_o$ happened before $t_b$ and $t_b$ happened before $t_r$ the following inequalities should hold:

$$t_o(t) < a_{12} t_b(t) + b_{12};$$

(3)

$$t(t) > a_{12} t_b(t) + b_{12};$$

(4)

The measurement described above will be repeated several times and each probe which returns will provide a new data point and thus new constraints on the admissible values of $a_{12}$ and $b_{12}$. To decrease the overhead of this data-gathering algorithm the probes can be piggybacked on data messages. Since most MAC protocols in wireless networks employ an acknowledgment (ACK) scheme, the probes can be piggybacked on the data and the responses on the ACKs. After acquiring a few (at least two) data-points, the offset and the drift can be estimated using inequalities (3) and (4). An existing solution for finding the optimal bounds on the drift and offset involves solving two linear programming problems with twice as many inequalities as data points.
The experimental results show that the simpler of the two algorithms, called tiny-sync, produces results very close to the optimum (within 0.1%) and thus is preferable. The above two methods can be extended to any number of nodes and thus used for synchronizing an entire network.

3.2 Global Synchronization

Global synchronization is crucial to many sensor network applications that require precise mapping of the collected sensor data with the time of the events, for example in tracking and surveillance. It also plays an important role in energy conservation in MAC layer protocol. Li et al. [2] have presented three methods to achieve global synchronization in a sensor network: (1) a node based approach, (2) a hierarchical cluster based method and (3) a fully localized diffusion-based method. The first method assumes the transmission time of a packet across a hop is the same for all nodes. It uses a packet to go around a cycle that is composed of all the nodes in the network and amortizes the packet transmission time on the cycle to each hop. This technique has scalability issues because it requires the nodes in the whole network to participate in the synchronization process at the same time. However this can be addressed by using clusters. In this approach cluster header nodes are synchronized by using the first method and in each cluster the members are synchronized using the cluster head. The last method is a fully localized diffusion based method in which each nodes exchanges and updates information locally with all neighbors. This method achieves full scalability. It can choose various global values to synchronize the network provided that each node in the overall network agrees to change its clock reading to the consensus value. An easy and possible way is to choose the highest or lowest reading over the network. To make the algorithms more robust and reasonable, the following algorithms use the global average value as the ultimate synchronization clock reading.

The authors also present asynchronous version of the rate-based algorithm. It is nice that the synchronous algorithm is localized. However, it requires the node operations to be done in a set order. No node can perform the operation without waiting for all the nodes to finish the current round of operations. The asynchronous version of the algorithm is more practical and does not have this constraint. All the nodes can perform operations in any order as long as each node is involved in the operations with non-zero probability. This method can be shown to converge. Our proposed algorithms can be extended to other sensor network applications, such as data aggregation.

3.3 Fine Grained Network Time Synchronization using Reference Broadcasts

This scheme proposed by Elson et al.[3] sends reference beacons to their neighbors using physical-layer broadcasts. A reference broadcasts does not contain an explicit timestamp; instead receivers utilize the arrival time as a reference point for comparing their clocks. The authors show from implementations that removing the sender’s non-determinism from the critical path results in a dramatic improvement in synchronization over using Network Time Protocol (NTP). An algorithm is presented that allows time to be propagated across broadcast domains without losing the reference-broadcast property. In this way, nodes in a multi-hop network can form a highly precise relative timescale, or maintain microsecond-level synchronization to an external timescale such as UTC. However the greatest drawback of this approach is that it requires a network with a physical broadcast channel and cannot be used in a point-to-point network.

The advantages of RBS include:
- The largest sources of non-deterministic latency can be removed from the critical path by using the broadcast channel to synchronize receivers with one another. This results in significantly better precision synchronization than algorithms that measure round-trip delay.
- Multiple broadcasts allow tighter synchronization because residual errors tend to follow well-behaved distributions. In addition, multiple broadcasts allow estimation of clock skew and thus extrapolation of past phase offsets. This enables post-facto synchronization, saving energy in applications that need synchronized time infrequently and unpredictably.
- Outliers and lost packets are handled gracefully.
- RBS allows nodes to construct local timescales. This is useful for sensor networks or other applications that need synchronized time but may not have an absolute time reference available.
An extension to basic RBS can be used to synchronize a group of nodes that lie beyond the range of a single broadcast. Consider the example topology shown in Figure 2 above. The lettered nodes, A and B, both send a sync pulse. A and B can not hear each other, but each of them are heard by 4 receivers. Receivers that are in the same neighborhood (i.e., have heard the same sync pulse) can relate their clocks to each other, as described in previous sections. However, notice that receiver 4 is in a unique position: it can hear the sync pulses from both A and B. This allows receiver 4 to relate the clocks in one neighborhood to clocks in the other. The authors have implemented RBS on a variety of hardware platforms, where it has proven to be robust and reliable for both performance measurement and in support of real applications.

### 3.4 Improved Interval-Based Clock Synchronization in Sensor Networks

Blum et al [4] in a paper have proposed a model for synchronization in ad-hoc, sporadic communication scenarios. The model allows us to identify the worst and the best case in terms of achievable time uncertainty and to show the worst-case optimality of the discussed algorithms. Simulations suggest that in the average case, this algorithm can significantly reduces the time uncertainty.

This improvement is achieved by exploiting the typical drift diversity of the nodes’ clocks. The authors present and analyze an improved version of the algorithm IM, the Back-Path Interval Synchronization Algorithm (BP-ISA). The BP-ISA is worst-case-optimal like the algorithm IM, but achieves better results in non-worst-case executions. In BP-ISA for each neighbor, nodes store the time bounds of the last communication event with this neighbor. Whenever a node can improve its current bounds through communication, it tries to use them to also improve bounds in its history. At communication events, the nodes exchange their current bounds and the bounds of the previous encounter (if it exists). Each node then uses both bounds to improve all bounds in its history and introduces additional paths in the timing graph. The algorithm introduces new and possibly shorter paths from a source event to a destination event. By considering these paths, the BP-ISA can achieve better time uncertainties than the algorithm IM.

The simulation results show that the BP-ISA provides substantial improvements percents if the sensor nodes communicate relatively often among each other, but only rarely with anchor nodes. The BP-ISA performs best in networks where the average length of paths to anchor nodes is large. For a given node density, this is the case at those small transmission ranges that lead to (almost) connected networks. The graph below shows the improvement for a simple case. If event b provides uncertainty zero, and occurs immediately after event a, 100% improvement can be achieved.
3.5 The Flooding Time Synchronization Protocol

Maroti et al. [6] describe the Flooding Time Synchronization (FTSP) especially designed for applications requiring high precision on resource limited wireless platforms. The FTSP achieves its robustness by utilizing periodic flooding of synchronization messages, and implicit dynamic topology update. The unique high precision performance is reached by utilizing MAC-layer time stamping and comprehensive error compensation including clock skew estimation. The sources of delays and uncertainties in message transmission are analyzed in detail and techniques are presented to mitigate their effects. Implementation results on Berkeley Mica2 platform and evaluated in a 60-node multi-hop setup produced an average per-hop synchronization error in one microsecond range, which is better than RBS and TPSN algorithms. The main idea here is to achieve a network wide synchronization of the local clocks of the participating nodes. The FTSP synchronizes the time of a sender to possibly multiple receivers utilizing a single radio message time-stamped at both the sender and the receiver sides. It also uses linear regression to compensate for clock drift. The root of the network—a single, dynamically (re)elected node—maintains the global time and all other nodes synchronize their clocks to that of the root. The nodes form an ad hoc structure to transfer the global time from the root to all the nodes, as opposed to a fixed spanning-tree based approach. This saves the initial phase of establishing the tree and is more robust against node and link failures and dynamic topology changes.

The FTSP utilizes a radio broadcast to synchronize the possibly multiple receivers to the time provided by the sender of the radio message. The broadcasted message contains the sender’s time stamp, which is the estimated global time at the transmission of a given byte. The receivers obtain the corresponding local time from their respective local clocks at message reception. Consequently, one broadcast message provides a synchronization point (a global-local time pair) to each of the receivers. The difference between the global and local time of a synchronization point estimates the clock offset of the receiver. As opposed to the RBS protocol, the time stamp of the sender must be embedded in the currently transmitted message. Therefore, the time stamping on the sender side must be performed before the bytes containing the time stamp are transmitted. During the transmission of the preamble bytes the receiver radio synchronizes itself to the carrier frequency of the incoming signal. From the SYNC bytes the receiver can calculate the bit offset it needs to reassemble the message with the correct byte alignment. The message layout is shown in figure 3.

![Figure 3. Data packets transmitted over the radio channel. Solid lines represent the bytes of the buffer and the dashed lines are the bytes of packets.](image)

![Figure 3. Data packets transmitted over the radio channel. Solid lines represent the bytes of the buffer and the dashed lines are the bytes of packets.](image)

3.6 Lightweight Time Synchronization for Sensor Networks

Greunen et al. [8] have presented a paper motivated by Pico-radio project. Their algorithms are designed to work with generic low cost sensor nodes and focus on minimizing overhead (energy) while being robust and self-configuring. In particular, the algorithms operate correctly in the presence of node failures, dynamically varying channels and node mobility. The single-hop, pair-wise synchronization scheme requires the exchange of only three messages and has Gaussian error properties. The authors extend the approach to a centralized multi-hop synchronization method.

Multi-hop synchronization consists of pair-wise synchronizations performed along the edges of a spanning tree. Multi-hop synchronization requires only n-1 pair-wise synchronizations for a network of n nodes. In addition, the authors show that communication
complexity and accuracy of multi-hop synchronization is a function of the construction and depth of the spanning tree; several spanning-tree construction algorithms are described. Further, the required refresh rate of multi-hop synchronization is shown as a function of clock drift and the accuracy of single-hop synchronization. Finally, a distributed multi-hop synchronization is presented where nodes keep track of their own clock drift and their synchronization accuracy. In this scheme, nodes initialize their own resynchronization as needed.

The multi-hop component of the level-based synchronization scheme differs from the scheme presented in this paper. In the level-based scheme each node is assigned a logical level indicating its distance from the chosen leader node. This level assignment is fixed for the lifetime of the leader. When new nodes join the synchronization they are required to initiate a “level discovery” phase. Pair-wise synchronizations are then performed between nodes in adjacent levels. The static nature of the level hierarchy reduces the robustness of this solution.

A basic scheme to synchronize pairs of nodes is described below. Nodes j and k can synchronize their local time by exchanging two packets with the following procedure:

1. Node j transmits the first packet with a timestamp t1 with respect to its local time. Node k records the time t2 when it receives the first packet. Time t2 is equal to t1 plus the transmission time D from node k to j plus the offset d between node j and k’s clocks. Generally the transmission time D is unknown and is a function of the distance between the nodes and signal propagation characteristics.

2. Next, node k transmits a second packet to j that contains t1 and t2. This packet is also time stamped by k at time t3.

3. Node j receives the second packet at time t4 = t3 + D – d.

4. The offset d can be calculated at node j by subtracting t4 from t2.
   \[ t2 - t4 = t1 - t3 - D + D + 2d \]
   \[ d = 0.5 * (t2 - t4 - t1 + t3) \]

5. The two nodes are synchronized once node j has calculated the offset d. However, a third message is required if the offset d must also be communicated to node k.

The underlying assumption is that the transmission time is the same from j to k and k to j, that is D1 = D2.

Figure 4 above shows the graphical depiction of the exchange.

Simulations results indicate that when all nodes participate the centralized scheme is more efficient than the distributed scheme but when a portion of the nodes need frequent synchronization the distributed scheme can result in less pair-wise synchronization. The scheme is generally robust and works well in the presence of dynamic links and fading.

3.7 Internal Synchronization of Drift-Constrain clocks in Ad-Hoc Sensor Networks

Meier et al. [7] have proposed a new algorithm that needs less computation and no more communication or memory than the original algorithm. It always yields equal or better results and thus outperforms the original algorithm. The authors argue that better synchronization can be obtained, possibly at the cost of additional computation, communication, and memory. To this end, they have introduced a model for internal synchronization. This model allows us to find an algorithm which makes use of all the data a node can obtain from the network for a given communication pattern and thus provides optimal synchronization.

The problem of internal synchronization as follows:

Given a trace with an event occurring at node \( N_1 \) at local time \( H_1(t) \), we are interested in tight bounds \( H'_1(t) \) on the local time \( h_1(t) \) of another node \( N_2 \) at time \( t \), such that \( H'_1(t) \leq h_1(t) \leq H'_2(t) \). Consider the example of sensor network in Figure 5 below, the sensor nodes \( N_2 \) and \( N_3 \) sense a vehicle at local times \( h_2(t1) \) and \( h_3(t2) \) which show that \( t_1 < t_2 \), it can conclude that the vehicle entered the sensing area of \( N_2 \) first and then that of \( N_3 \).
The algorithm presented in this paper uses all the data that can be obtained for a given communication pattern, at the expense of additional computation, communication and memory.

However more work is required in simulation and quantitative analysis of the gains in synchronization quality and of the additional requirements for a large network; this allows to find a trade-off between synchronization quality and required resources.

### 3.8 Locating tiny sensors in time and space: A case study

Girod et al. [9] have described a system based on commercial off-the-shelf (COTS) components, which is capable of automatic localization and time synchronization with sufficient precision (on the order of 10cm and 10 micro sec) to support distributed, coherent signal processing. In their system synchronization is implemented using Reference Broadcast synchronization (RBS). The RBS daemon simultaneously acts in both “sender” and “receiver” roles. Every 10 seconds (slightly randomized to avoid unintended synchronization), each daemon emits a pulse packet with a sequence number and sender ID. The daemon also watches for such packets to arrive; it timestamps them and periodically sends a report of these timestamps back to the pulse sender along with its receiver ID. The pulse sender collects all of the pulse reception reports and computes clock conversion parameters between each pair of nodes that heard its broadcasts. These parameters are then broadcast back to local neighbors. The RBS daemons that receive these parameters make them available to users. (RBS never sets the nodes’ clocks, but rather provides a user library that converts UNIX timevals from one node ID to another.) The authors have used a very simple and effective algorithm to correct skew: a least-squares linear regression on the time series of phase differences between nodes, after automatic outlier rejection. This offers a fast, closed-form method for finding the best-fit line through the phase error observations over time. The frequency and phase of the local node’s clock with respect to the remote node can be recovered from the slope and intercept of the line. In addition to RBS functionality, the time service daemon also supports synchronization between components within a system. The system’s device drivers periodically inject pairs of time values into the time daemon, where each pair represents the value of two of the system’s clocks at the same instant. This allows the clocks to be related.

The authors show that it is possible on COTS hardware by making using of novel techniques, including Reference-Broadcast Synchronization and wideband acoustic ranging to achieve a 10 micro sec time synchronization and 10cm spatial localization on a low-cost, low-power, ad-hoc deployable sensor network.

### 3.9 Adaptive Clock Synchronization in Sensor Networks

Pal Chaudhuri et al. [10] in paper have introduced the concept of adaptive clock synchronization based on the need of the application and the resource constraint in the sensor networks. It is a probabilistic method for clock synchronization that uses the higher precision of receiver-to-receiver synchronization, as described in Reference Broadcast Synchronization (RBS) protocol. This deterministic protocol is extended to provide a probabilistic bound on the accuracy of the clock synchronization, allowing for a tradeoff between accuracy and resource requirement.

Expressions to convert service specifications maximum clock synchronization error and confidence probability) to actual protocol parameters (minimum number of messages and synchronization overhead) are derived. The authors also extend the approach for a multi-hop network.

The authors describe different synchronization methods:
1. Global clock
2. Relative clock
3. Relative notion of time
4. Physical Ordering

Another classification is in terms of the initiator of synchronization procedure. They are as follows:
1. Always On: In this model, clock synchronization between nodes is always present. Many applications like TDMA scheduling might need this model.
2. 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 and is useful in infrequent synchronization.

3. 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.

The main idea of probabilistic protocol is that that guarantee cannot be provided when unbounded message delays are possible or messages can be lost. Hence, it makes sense to retry client 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. 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 \(1 - p^k\).

The receiver-receiver synchronization used in this paper works in the following manner - 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 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 in formation with A. The number of reference packets may be increased in order to get better synchronization.

The protocol can be described as follows - 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 starts the cycle and repeats it periodically.

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 timestamps, 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 is 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 particular sender. So, for every pair of receivers, within the broadcast region of the sender, the clock skew and clock offset are now known with some synchronization error. The Send Time and Access Time errors are 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. The relationship between the number of messages and the achieved probability \(P\) of error is depicted in the graph below.

![Figure: Probability of achieved error being less than a maximum](image-url)
4.0 Conclusion
Time synchronization is a complicated and an active area of research within the domain of sensor networks. The goal here was to analyze and present some regimes for time synchronization in sensor networks that have been proposed over the last few years. Some approaches maybe more suitable depending on applicability, that is where or which environment the sensor network will be deployed in.

5.0 Acknowledgements
I would like to thank Prof. KD Kang and fellow students of the sensor network class in supporting this study.

6.0 References
[2] Qun Li, Daniela Rus, Global Clock Synchronization in Sensor Networks, INFOCOM’04, March 2004