Clock Synchronization in Sensor Networks(Distributed System)

Vinit Kamboj
Computer Science Department
Illinois Institute of Technology
Chicago, USA
vkamboj@hawk.iit.edu

Ritika Kumari
Computer Science Department
Illinois Institute of Technology
Chicago, USA
rkumari@hawk.iit.edu

***Abstract*—Clock Synchronization is a critical component in the operation of Distributed System. Our paper will discuss clock synchronization in Sensor Network. Wireless ad-hoc sensor networks have emerged as an interesting and important research area in the last few years. The applications envisioned for such networks require collaborative execution of a distributed task amongst a large set of sensor nodes. This is realized by exchanging messages that are time-stamped using the local clocks on the nodes. Therefore, time synchronization becomes an indispensable piece of infrastructure in such systems. For years, protocols such as NTP have kept the clocks of networked systems in perfect synchrony. However, this new class of networks has a large density of nodes and very limited energy resource at every node; this leads to scalability requirements while limiting the resources that can be used to achieve them. A new approach to time synchronization is needed for sensor networks.**

Our paper will discuss about Timing-sync Protocol for Sensor Networks (TPSN) that aims at providing network-wide time synchronization in a sensor network, Reference Broadcast synchronization, Network Time protocol for Sensor Networks

Keywords—Clock Synchronization, Distributed Systems, Sensor Networks, Time-Sync Protocol Network, Network Time Protocol, Synchronization Algorithm formatting, Reference Broadcast Time protocol.

I. INTRODUCTION

Distributed system consists of a set of processes that communicate through message passing and do not have access to global time. Clock synchronization is a fundamental service that is required in many applications distributed systems. Distributed system is one in which hardware and software components located on a network communicate and coordinate their actions only by message passing. There are two types of Distributed Systems Homogeneous Distributed Systems (HDS) & Heterogeneous Distributed Systems (HeDS).

a) Homogeneous Distributed Systems (HDS): It is a distributed system such that all nodes have identical hardware, the same type of architecture and operating system.

b) Heterogeneous Distributed Systems (HeDS): It is a distributed system such that each node has their own operating system and machine architecture.

Resources like a printer and scanner cannot be used by multiple processes simultaneously, so it must wait for one process to complete and then give chance to the next process

Time synchronization in all networks either wired or wireless is important. It allows for successful communication between nodes on the network. It is, however, particularly vital for wireless networks. Synchronization in wireless nodes allows for a TDMA algorithm to be utilized over a multi-hop wireless network. Wireless time synchronization is used for many different purposes including location, proximity, energy efficiency, and mobility to name a few.

In sensor networks when the nodes are deployed, their exact location is not known so time synchronization is used to determine their location. Also, time stamped messages will be transmitted among the nodes in order to determine their relative proximity to one another. Time synchronization is used to save energy; it will allow the nodes to sleep for a given time and then awaken periodically to receive a beacon signal. Many wireless nodes are battery powered, so energy efficient protocols are necessary. Lastly, having common timing between nodes will allow for the determination of the speed of a moving node.

The need for synchronization is apparent. Besides its many uses like determining location, proximity, or speed, it is also needed because hardware clocks are not perfect. There are variations in oscillators, which the clocks may drift, and durations of time intervals of events will not be observed the same between nodes. The concept of time and time synchronization is needed, especially in wireless networks.

II. NETWORK SYNCHRONIZATION

A. Wired Network Synchronization

For a wired network, two methods of time synchronization are most common. Network Time Protocol and Global Positioning System (GPS) are both used for synchronization. Neither protocol is useful for wireless synchronization. Both require resources not available in wireless networks.

The Network Time Protocol requires an extremely accurate clock, usually a server with an atomic clock. The client computer wanting to synchronize with the server will send a UDP packet requesting the time information. The server will then return the timing information and as a result the computers would be synchronized. Because of many wireless devices are powered by batteries, a server with an atomic clock is impractical for a wireless network.

GPS requires the wireless device to communicate with satellites in order to synchronize. This requires a GPS receiver in each wireless device. Again because of power constraints, this is impractical for wireless networks. Also, sensor networks consist of inexpensive wireless nodes. A GPS receiver on each wireless node would be expensive and therefore unfeasible. The time accuracy of GPS depends on how many satellites the receiver can communicate with at a given time. This will not always be the same, so the time accuracy will vary. Furthermore, Global Positioning System devices depend on line of sight communication to the satellite, which may not always be available where wireless networks are deployed.

The constraints of wireless networks do not allow for traditional wired network time synchronization protocols. Wireless networks are limited to size, power, and complexity. Neither the Network Time Protocol nor GPS were designed for such constraints.

B. Wireless Network Synchronization

The definition of time synchronization does not necessarily mean that all clocks are perfectly matched across the network. This would be the strictest form of synchronization as well as the most difficult to implement. Precise clock synchronization is not always essential, so protocols from lenient to strict are available to meet one's needs.

There are three basic types of synchronization methods for wireless networks. The first is relative timing and is the simplest. It relies on the ordering of messages and events. The basic idea is to be able to determine if event 1 occurred before event 2. Comparing the local clocks to determine the order is all that is needed. Clock synchronization is not important.

The next method is relative timing in which the network clocks are independent of each other and the nodes keep track of drift and offset. Usually a node keeps information about its drift and offset in correspondence to neighboring nodes. The nodes have the ability to synchronize their local time with another nodes local time at any instant. Most synchronization protocols uses this method.

The last method is global synchronization where there is a constant global timescale throughout the network. This is obviously the most complex and the toughest to implement. Very few synchronizing algorithms use this method particularly because this type of synchronization usually is not necessary.

III. TIME SYNCRHORNIZATION

A. Sensor Node Clock

Every sensor node maintains its own clock, and this is the only notion of time that a node has. The clock is an ensemble of hardware and software components; it is essentially a timer that counts the oscillations of a quartz crystal running at a frequency. Computing devices are mostly equipped with a hardware oscillator assisted computer clock, which implements an approximation C(t) of real-time t. Let us represent the clock for node A by $C_A(t)$. The difference in the clocks of two sensor nodes (i.e., A and B) is referred as the offset error between them. There are three reasons for the nodes to be representing different times in their respective clocks (Ganeriwal et al. 2008): 1) The nodes might have been started at different times, 2) the quartz crystals at each of these nodes might be running at slightly different frequencies, causing the clock values to gradually diverge from each other (termed as the skew error), or 3) the frequency of the clocks can change differently over time because of aging or ambient conditions such as temperature (termed as the drift error). These errors can be summarized as follows:

Offset: $\delta = C_A(t) - C_B(t)$
Skew: $\eta = (\partial C_A(t) - \partial C_B(t))/\partial t$
Drift: $\lambda = (\partial^2 C_A(t) - \partial^2 C_B(t))/\partial^2 t$

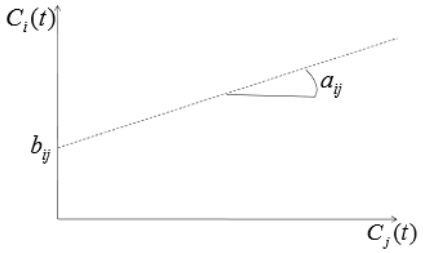
The angular frequency of the hardware oscillator determines the rate at which the clock runs. The rate of a perfect clock, which can be denoted as *dC/dt*, would equal 1, however, all clocks are subject to a clock drift; oscillator frequency will vary unpredictably due to various physical effects. Even though the frequency of a clock changes over time, it can be approximated with good accuracy by an oscillator with fixed frequency (Sichitiu & Veerarittiphan, 2003). Then, for some node i in the network, we can approximate its local clock as:

$$C_i(t) = a_i(t) + b_i$$

Where $a_i(t)$ is the clock drift, and b_i is the offset of node i's clock. Drift denotes the rate (frequency) of the lock, and offset is the difference in value from real time t. Using equation (4), we can compare the local clocks of two nodes in a network, say node i and node j as:

$$C_i(t) = a_{ij} * C_j(t) + b_{ij}$$

We call the a_{ij} relative drift, and b_{ij} the relative offset between the clocks of node i and node j. If two clocks are perfectly synchronized, then their relative drift is i (meaning the clocks have the same rate) and their relative offset is zero (meaning they have the same value at that instant). Some studies in the literature use "skew" instead of "drift", defining it as the difference (as opposed to ratio) between clock rates. Also, the "offset" may equivalently be mentioned as "phase offset". Fig. 1 shows the relationship between relative drift and offset.



Relation between relative drift and offset

Although each sensor node is equipped with a hardware clock, these hardware clocks can usually not be used directly, as they suffer from severe drift. No matter how well the hardware clocks will be calibrated at deployment, the clocks will ultimately exhibit a large skew. Since all hardware clocks are imperfect, local clocks of nodes may drift away from each other in time, hence observed time or durations of time intervals may differ for each node in the network. To allow for an accurate common time, nodes need to exchange messages from time to time, constantly adjusting their clock values. Furthermore, nodes can convert the current hardware clock reading into a logical clock value and vice versa (Sommer & Wattenhofer, 2009).

B. Challenges in sensor networking

Wireless sensor networks have tremendous potential because they will expand our ability to monitor and interact remotely with the physical world. Smart sensors have the ability to collect vast amounts of hitherto unknown data, which will pave the way for a new breed of computing applications as we showed in Table 1. Sensors can be accessed remotely and placed where it is impractical to deploy data and power lines. Nodes can be spaced closely, yielding fine-grained pictures of real-world phenomena that are currently modeled only on a large scale. However, to exploit the full potential of sensor networks, we must first address the peculiar limitations of these networks and the resulting technical issues. Evidently, sensor networks can be best exploited by applications that perform data fusion to synthesize global knowledge from raw data on the fly. Although data fusion requires that nodes be synchronized, the synchronization protocols for sensor networks must address the following features of these networks:

(a)Limited energy: While the efficiency of computing devices is increasing rapidly, the energy consumption of a wireless sensor network is becoming a bottleneck. Due to the small size and cheap availability of the sensors, sensor networks can employ thousands of sensors. This makes it impossible to wire each of these sensors to a power source. Also, the need for unmanned operation dictates that the sensors be battery-powered. Since the amount of energy available to such sensors is quite modest, synchronization must be achieved while preserving energy to utilize these sensors in an efficient fashion.

(b)Limited bandwidth: In wireless sensor nets, much less power is consumed in processing data than transmitting it. Presently, wireless communication is restricted to a data rate in the order of 10–100 Kbits/second [21]. Pottie and Kaiser [55] have shown that the energy required to transmit 1 bit over 100 meters, which is 3 joules, can be used to execute 3 million instructions. Bandwidth limitation directly affects message exchanges among sensors, and synchronization is impossible without message exchanges.

IV. REFERENCE BROADCAST SYNCRHRONIZATION

A. Reference Broadcast Synchronization

Many of the time synchronization protocols use a sender to receiver synchronization method where the sender will transmit the timestamp information and the receiver will synchronize. RBS is different because it uses receiver to receiver synchronization. The idea is that a third party will broadcast a beacon to all the receivers. The beacon does not contain any timing information; instead the receivers will compare their clocks to one another to calculate their relative phase offsets. The timing is based on when the node receives the reference beacon.

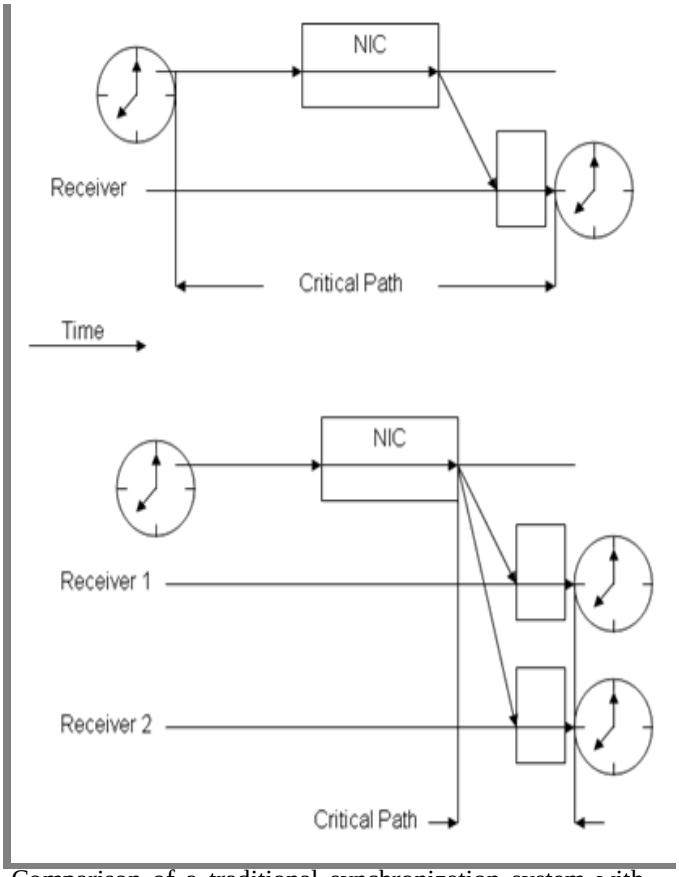
The simplest form of RBS is one broadcast beacon and two receivers. The timing packet will be broadcasted to the two receivers. The receivers will record when the packet was received according to their local clocks. Then, the two receivers will exchange their timing information and be able to calculate the offset. This is enough information to retain a local timescale.

RBS can be expanded from the simplest form of one broadcast and two receivers to synchronization between n receivers, where n is greater than two. This may require more than one broadcast to be sent. Increasing the broadcasts will increase the precision of the synchronization.

RBS differs from the traditional sender to receiver synchronization by using receiver to receiver synchronization. The reference beacon is broadcasted across all nodes. Once it is received, the receivers note their local time and then exchange timing information with their neighboring nodes. The nodes will then be able to calculate their offset.

B. Advantages of RBS

The main advantage of RBS is that it eliminates the uncertainty of the sender by removing the sender from the critical path. By removing the sender, the only uncertainty is the propagation and receive time. The propagation time is negligible in networks where the range is relatively small. It is claimed that the reference beacon will arrive at all the receiving nodes instantaneously. By removing the sender and propagation uncertainty the only room for error is the receiver uncertainty. Figure below will illustrate this concept.



Comparison of a traditional synchronization system with RBS

As seen here, the critical path in a traditional system, which is the top diagram, includes the sender. Since RBS is a receiver to receiver synchronization the sender is removed from the critical path. The critical path on contains the propagation and the receiver uncertainty. If, however, the transmission range is relatively small, then we can eliminate the propagation time and the critical path only contains the uncertainty of the receiver

V. TIMING-SYNC PROTOCOL FOR SENSOR NETWORKS

TPSN is a traditional *sender-receiver* based synchronization that uses a tree to organize the network topology. The concept is broken up into two phases, the level discovery phase and the synchronization phase. The level discovery phase creates the hierarchical topology of the network in which each node is assigned a level. Only one node resides on level zero, the root node. In the synchronization phase all *i* level nodes will synchronize with *i-1* level nodes. This will synchronize all nodes with the root node.

A. Level Discovery Phase

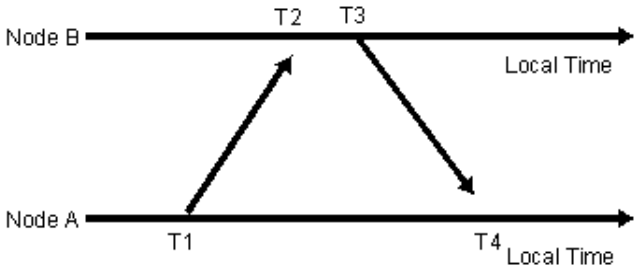
The level discovery phase is run on network deployment. First, the root node should be assigned. If one node was equipped with a GPS receiver, then that could be the root node and all nodes on the network would be synced to the world time. If not, then any node can be the root node and other nodes can periodically take over the functionality of the root node to share the responsibility.

Once the root node is determined, it will initiate the level discovery. The root, level zero, node will send out the level_discovery packet to its neighboring nodes. Included in the level_discovery packet is the identity and level of the sending node. The neighbors of the root node will then assign themselves as level one. They will in turn send out the level_discovery packet to their neighboring nodes. This process will continue until all nodes have received the level_discovery packet and are assign a level.

Once again, all nodes are assigned a level to create a tree type topology. The root node is level zero continuing down the tree with level one and so on. All nodes of level i will broadcast the level_discovery with all nodes of level i-1. This is maintained until all nodes are assigned a level.

B. Synchronization Phase

The basic concept of the synchronization phase is two-way communications between two nodes. As mentioned before this is a sender to receiver communication. Like the level discovery phase, the synchronization phase begins at the root node and propagates through the network.



Two-way communication between nodes

Above figure illustrates the two-way messaging between a pair of nodes. This messaging can synchronize a pair of nodes by following this method. The times T1, T2, T3, and T4 are all measured times. Node A will send the synchronization_pulse packet at time T1 to Node B. This packet will contain Node A's level and the time T1 when it was sent. Node B will receive the packet at time T2. Time T3 is when Node B sends the acknowledgment_packet to Node A. That packet will contain the level number of Node B as well as times T1, T2, and T3. By knowing the drift, Node A can correct its clock and successfully synchronize to Node B. This is the basic communication for TPSN.

The synchronization process is again initiated by the root node. It broadcasts a time_sync packet to the level one node. These nodes will wait a random amount of time before initiating the two-way messaging. The root node will send the acknowledgment and the level one node will adjust their clocks to be synchronized with the root nodes.

The level two node will be able to hear the level one nodes communication since at least one level one node is a neighbor of a level two node. On hearing this communication, the level two nodes will wait a random period of time before initiating the two-way messaging with the level one node. This process will continue until all nodes are synchronized to the root node.

Again, the synchronization process executes much the same as the level discovery phase. All communication begins with the root node broadcasting information to the level 1 nodes. This communication propagates through the tree until all level i-1 nodes are synchronized with the level i nodes. At this point all nodes will be synchronized with the root node.

C. Advantages of TPSN

Any synchronization packet has the four delays discussed earlier: send time, access time, propagation time, and receive time. Eliminating any of these would be a plus. Although TPSN does not eliminate the uncertainty of the sender it does, however, minimize it. Also, TPSN is designed to be a multi-hop protocol; therefore, transmission range is not an issue.

Unlike RBS, TPSN has uncertainty in the sender. They attempt to reduce this non-determinism by time stamping packets in the MAC layer. It is claimed that the sender's uncertainty contributes very little to the total synchronization error. By reducing the uncertainty with low level time stamping, it is claimed that TPSN has a 2 to 1 better precision than RBS and that the sender to receiver synchronization is superior to the receiver to receiver synchronization. [Ganeriwal2003]

RBS also is limited by the transmission range. It was stated that RBS can ignore the propagation time if the range of transmission was relatively small. If it is a large multi-hop network, this is not the case. RBS would have to send more reference beacons for the node to synchronize. TPSN on the other hand was designed for multi-hop networks. Their protocol uses the tree-based scheme, so the timing information can accurately propagate through the network.

The sender to receiver synchronization method is claimed to be more precise than the receiver to receiver synchronization. Also, TPSN is designed for multi-hop networks, where RBS works best on single hop networks. So, the transmission range is not a factor with TPSN.

D. Special Provisions

In a sensor network, the nodes are usually deployed in a random fashion. So, scenarios might exist where a sensor node joins an already established network i.e., the node might join the network when the level discovery phase is already over. Even if the node is present at the onset of the network, it might not receive any level_discovery packets owing to MAC layer collisions. In either case it will not be assigned any level in the hierarchy. However, every node needs to be a part of the hierarchical topology so that it can be synchronized with the root node. Thus, when a node is deployed, it waits for some time to be assigned a level. If it is not assigned a level within that period, it timeouts and broadcasts a level_request message. The neighbors reply to this request by sending their own level. The new node assigns itself a level, one greater than the smallest level it has received and hence, joins the hierarchy. This could be seen as a local level discovery phase.

Sensor nodes may also die randomly. A situation may arise, when a level i node does not have any neighbor at level i-1. In such scenarios, the node would not get back an acknowledgement to its synchronization_pulse. Thus, this node at level i would not be able to synchronize to the root node. It has already been explained that in order to handle collisions, a node would retransmit the synchronization_pulse after some random amount of time. After retransmitting the synchronization_pulse a fixed number of times, a node assumes that it has lost all its neighbors on the upper level and broadcasts a level_ request message. On getting back a reply, the node is assigned a new level. Assuming the network is still connected, the node will have at least one node in its neighbor set and thus it will surely be assigned a new level in the hierarchy. We consider four retransmissions to be a heuristic for deciding non-availability of a neighbor in the upper level. The validity of this heuristic has been verified via simulations. In general, choosing a large number will increase the time taken for synchronization whereas a small number will cause unnecessary flooding in the network, decreasing the synchronization accuracy. We started with the premise that a node has been designated as the root node. If an elected root node dies, the nodes in level would not receive any acknowledgement packets and hence, they will timeout following the scheme described above. Instead of broadcasting a level_request packet, they run a leader election algorithm and the elected leader takes over the functionality of the root node. This new root node starts from the beginning and reruns the level discovery phase. Note that these special provisions are essentially heuristics to take care of ambiguities in the network. Though we do not claim that these are indeed the optimal solutions, we have verified their efficacy via extensive simulations.

E. Error Analysis of TPSN

In this section, we characterize the possible sources of error and present a detailed mathematically analysis for TPSN. We concentrate on pair wise synchronization between two nodes. We compare the performance of our scheme to RBS [7], an algorithm that synchronizes a set of receivers in sensor networks. However, as will be clear from our analysis, the results can be in general extended to make a comparison between the classical approach of sender-receiver synchronization and receiver-receiver synchronization.

F. Decomposition of Packet Delay

Above figure shows the decomposition of packet delay when it traverses over a wireless link between two sensor nodes. We designate the node that initiates the packet exchange as the sender and the node that responds to this message as the receiver. Although a similar decomposition has also been presented in, we detail the various delay components

from a systems perspective. In this discussion, we will borrow terms from a typical layered architecture used in traditional computer networks.

- **Send time**: When a node decides to transmit a packet, it is scheduled as a task in a typical sensor node. There is time spent in constructing the packet at the application layer, after which it is passed to the lower layers for transmission. This time includes the delay incurred by the packet to reach the MAC layer from the application layer. This delay is highly variable due to the software delays introduced by the underlying operating system.
- Access time: After reaching the MAC layer, the packet waits until it can access the channel. This delay is specific to wireless networks resulting from the property of common medium for packet transmission. This is perhaps the most critical factor contributing to packet delay. Moreover, it's highly variable in nature and is specific to the MAC protocol employed by the sensor node.

- **Transmission time:** This refers to the time when a packet is transmitted bit by bit at the physical layer over the wireless link. This delay is mainly deterministic in nature and can be estimated using the packet size and the radio speed. The software implementation of the transmitter will have a few minor variations due to the response time for interrupts. A novel hardware-based RF transceiver has been proposed where the variations would be completely negligible.

- **Propagation time:** This is the actual time taken by the packet to traverse the wireless link from the sender to the receiver. The absolute value of this delay is negligible as compared to other sources of packet latency.

- **Reception time:** This refers to the time taken in receiving the bits and passing them to the MAC layer. This is going to be mainly deterministic in nature. The variations in reception delay would even be smaller if the sensor node employs a hardware-based RF Transceiver.

- **Receive time:** The bits are then constructed into a packet and then this packet is passed on to the application layer where it's decoded. The time taken in this whole activity refers to receive time. The value of receive time changes due to the variable delays introduced by the operating system.

The size used for different boxes is just to give an intuition about the absolute value of each component. It doesn't correspond to their actual ratio. For example, we expect that the access time (MAC delay) would completely overshadow other delays in practice. Secondly, communication takes place in bits and a node optimizes by performing events in parallel. Thus, when a bit is being coded for transmission, another bit could be in air or being received at the other end simultaneously. Thus, this decomposition is just an approximation when done at the packet level instead of the bit level.

VI. FLOODING TIME SYNCHRONIZATION PROTOCOL

A. Introduction

Another form of sender to receiver synchronization is FTSP. This protocol is similar to TPSN, but it improves on the disadvantages to TPSN. It is similar in the fact that it has a structure with a root node and that all nodes are synchronized to the root.

The root node will transmit the time synchronization information with a single radio message to all participating receivers. The message contains the sender's time stamp of the global time at transmission. The receiver notes its local time when the message is received. Having both the sender's transmission time and the reception time, the receiver can estimate the clock offset. The message is MAC layer time stamped, as in TPSN, on both the sending and receiving side. To keep high precision compensation for clock drift is needed. FTSP uses linear regression for this.

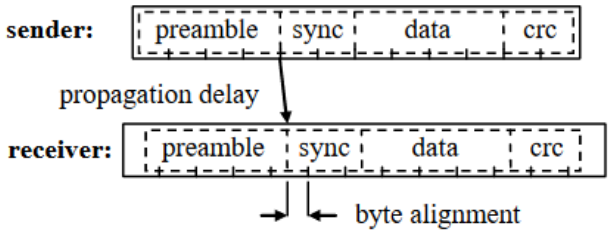
FTSP was designed for large multi-hop networks. The root is elected dynamically and periodically reelected and is responsible for keeping the global time of the network. The receiving nodes will synchronize themselves to the root node and will organize in an ad hoc fashion to communicate the timing information amongst all nodes. The network structure is mesh type topology instead of a tree topology as in TPSN.

Typical WSN operate in areas larger than the broadcast range of a single node; therefore, the FTSP provides multi-hop synchronization. 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 proposed. This saves the initial phase of establishing the tree and is more robust against node and link failures and dynamic topology changes.

B. Time Stamping

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 is transmitted.



Message broadcast starts with the transmission of preamble bytes, followed by SYNC bytes, then with a message descriptor followed by the actual message data, and ends with CRC bytes. 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 descriptor contains the target, the length of the data and other fields, such as the identifier of the application layer that needs to be notified on the receiver side. The CRC bytes are used to verify that the message was not corrupted. The message layout is summarized in Figure above

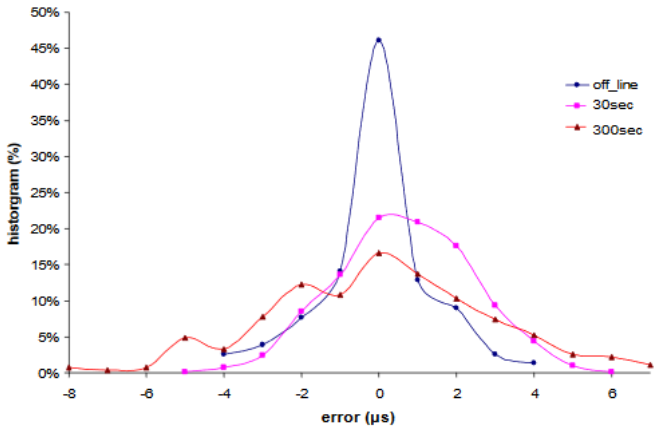
The FTSP time-stamping effectively reduces the jitter of the interrupt handling and encoding/decoding times by recording multiple time stamps both on the sender and receiver sides. The time stamps are made at each byte boundary after the SYNC bytes as they are transmitted or received. First, these time stamps are normalized by subtracting an appropriate integer multiple of the nominal byte transmission time, the time it takes to transmit a byte. The jitter of interrupt handling time is mainly due to program sections disabling interrupts on the microcontroller for short amounts of time. This error is not Gaussian but can be eliminated with high probability by taking the minimum of the normalized time stamps. The jitter of encoding and decoding time can be reduced by taking the average of these interrupt error corrected normalized time stamps. On the receiver side this final averaged time stamp must be further corrected by the byte alignment time that can be computed from the transmission speed and the bit offset. Note that, even though multiple time stamps are made, only the final error corrected time stamp is embedded into the message. The number of bytes put an upper limit on the achievable error correction using this technique. However, with only 6-time stamps, the time-stamping precision can be improved from tens of microseconds to 1.4μs on the Mica2 platform as measured by the following experiment.

Four motes were sending time-stamped messages to each other for 10 minutes, each with a 5-second sending period. The time-stamps were recorded both on the sender and receiver sides, and the pairwise clock offset and skew values were determined off-line with linear regression. The time-stamping error is the absolute value of the difference of the recorded receiver side time stamp and the linearly corrected sender side time-stamp. The average and maximum time-stamping errors were 1.4μs and 4.2μs, respectively. Since the FTSP time-stamping employs a single radio message, it does not and cannot compensate for the propagation delay. This is not a major limitation of the approach in typical WSN, however, as the propagation delay is less than 1μs for up to 300 meters.

C. Clock drift Management

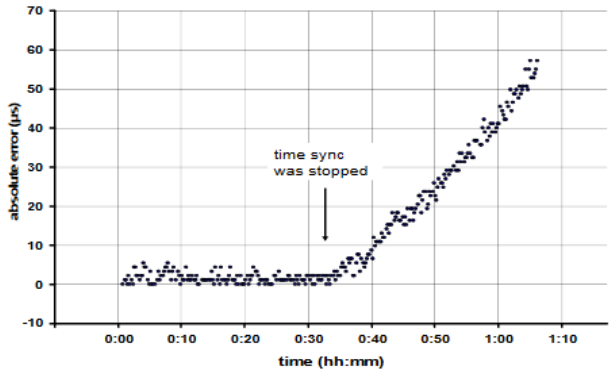
If the local clocks had the exact same frequency and, hence, the offset of the local times were constant, a single synchronization point would be sufficient to synchronize two nodes. However, the frequency differences of the crystals used in Mica2 motes introduce drifts up to 40μs per second. This would mandate continuous re-synchronization with a period of less than one second to keep the error in the micro-second range, which is a significant overhead in terms of bandwidth and energy consumption. Therefore, we need to estimate the drift of the receiver clock with respect to the sender clock.

The offset between the two clocks changes in a linear fashion provided the short-term stability of the clocks is good. We verified the stability of the 7.37 MHz Mica2 clock by periodically sending a reference broadcast message that was received by two different motes. The two motes time-stamped the reference message using the FTSP time-stamping described in the previous section with their local time of arrival and reported the time-stamp. For each transmitted message the offset of the two reported time-stamps was calculated. The offsets were further examined: linear-regression was used to find the line L best approximating the dataset and the errors were analyzed. For a data point (time, offset) and the regression line L, the error is offset-L(time). A one-hour experiment produced the following results: the average value of the absolute errors was 0.95μs and the maximum absolute error was 4.32μs. The distribution of the errors calculated off-line is shown in Figure4. This off-line regression provides the best prediction that can possibly be achieved, provided the clocks can be considered stable during the experiment. Naturally this method cannot be used online; it is used here as a reference to evaluate online solutions



We need to identify the trend of the global time relative to the local time from the data points received in the past.

Furthermore, only a limited number of data points can be stored due to the memory constraints of the platform. The following scenario was used to test our Mica2 implementation: mote A maintains the global time and sends synchronization messages to mote B with a period of T. Mote B estimates the skew and offset of its local clock from that of A using linear regression on the past 8 data points. A reference broadcaster sends a query message with period t and both A and B respond to this query by time-stamping its arrival with the global time and reporting it to the base station.



The linear regression prediction error is the difference between the global time given by A and the estimated global time given by B. Figure4 shows the distribution of these prediction errors, for (a) T=30s, t=18s, and (b) T=300s, t=93s. The length of experiment (a) was 18 hours, the average absolute error was 1.48µs, and the maximum absolute error was 6.48µs. The length of experiment (b) was 8 hours, the average absolute error was 2.24µs and the maximum absolute error was 8.64µs. An important design parameter is the required resynchronization interval to reach the desired precision. As shown in Figure4, the 30s resynchronization interval gave slightly better results than 300s. To further evaluate the behavior of the skew compensation, another experiment was carried out, the results shown inFigure5. This result shows that the resynchronization period, depending on the accuracy requirements, can go up to several minutes.

D. Multi-Hop Clock Synchronization

The linear regression prediction error is the difference between the global time given by A and the estimated global time given by B. Figure4 shows the distribution of these prediction errors, for (a) T=30s, t=18s, and (b) T=300s, t=93s. The length of experiment (a) was 18 hours, the average absolute error was 1.48µs, and the maximum absolute error was 6.48µs. The length of experiment (b) was 8 hours, the average absolute error was 2.24µs and the maximum absolute error was 8.64µs. An important design parameter is the required resynchronization interval to reach the desired precision. As shown in Figure4, the 30s resynchronization interval gave slightly better results than 300s. To further evaluate the behavior of the skew compensation, another experiment was carried out, the results shown in Figure. This result shows that the resynchronization period, depending on the accuracy requirements, can go up to several minutes. Multi-hop time synchronization (in practical WSN applications) the network radius is greater than one hop. If network-wide synchronization is required, the multi-hop FTSP protocol can be used, as described in this section. The only assumption the protocol makes is that every node in the network has a unique ID. Nodes in multi-hop FTSP utilize reference points to perform synchronization. A reference point contains a pair of global and local time stamps where both refer to the same time instant, as described in Section5.1. Reference points are generated by sending and receiving periodic broadcast messages, which are either transmitted by the synchronization-root (root, for short), or any synchronized node in the network. The root is a special node, elected and dynamically reelected by the network, to which the whole network is being synchronized. A node that is in the broadcast radius of the root can collect reference points directly from it. Nodes outside the broadcast radius of the root can gather reference points indirectly through other synchronized nodes that are located closer to the root. When a node collects enough consistent reference points, it estimates the offset and skew of its own local clock and becomes synchronized. The newly synchronized node can then broadcast synchronization messages to other nodes in the network. In the following subsections the most important aspects of the protocol will be presented.

Synchronization Message Format: Each synchronization message contains three fields: the timeStamp, the rootID, and the seqNum. The timeStamp contains the global time estimate of the transmitter when the message was broadcasted. The rootID field contains the ID of the root, as known by the sender of the message. The seqNum is a sequence number set and incremented by the root when a new synchronization round is initiated. Other synchronized nodes insert the most recent (i.e. the largest) received seqNum into the synchronization messages they broadcast. This field is used to handle redundant synchronization messages.

Managing Redundant Information: Since all synchronized nodes periodically transmit synchronization messages, in a dense network a receiver may receive several messages from different nodes in a short time interval. Due to limited resources, an appropriate subset of the messages must be selected to create reference points. In the Mica2 implementation an eight-element regression table stores the selected reference points used to calculate the regression line, and, thus, the drift of the local clock.

To achieve more accurate offset and skew estimation using the limited amount of data that can be stored in the regression table, it is more beneficial to store reference points that are distributed over a longer period. To aid message filtering, each node maintains a highestSeqNum variable. Also, each node has a myRootID variable, containing the root ID as known by the node. A received synchronization message is used to create a reference point only if the rootID field of the message is less than or equal to myRootID and the seqNum field is greater than highestSeqNumin the case when rootID = myRootID. The node's variables are updated after storing a reference point, according to the respective fields in the

message. This message filtering protocol guarantees that only the first message arrived will be used in the reference table for each rootID andseqNum pair (i.e. one per round), providing reference points distributed over a longer time period for more accurate skew and offset estimation. **The root election problem:** To perform global synchronization, obviously one and only one root is needed in the network. Since nodes may fail or the network can get disconnected, no dedicated node can play the role of the root. Thus, a robust election process is needed to provide a root after startup, and also in case of root failure. FTSP utilizes a simple election process based on unique node IDs, as follows:

```
1  event Radio.receive(TimeSyncMsg *msg)
2  {
3    if( msg->rootID < myRootID )
4      myRootID = msg->rootID;
5    else if( msg->rootID > myRootID
6          || msg->seqNum <= highestSeqNum )
7      return;
8
9    highestSeqNum = msg->seqNum;
10   if( myRootID < myID )
11     heartBeats = 0;
12
13   if( numEntries >= NUMENTRIES_LIMIT
14      && getError(msg) > TIME_ERROR_LIMIT )
15     clearRegressionTable();
16   else
17     addEntryAndEstimateDrift(msg);
18 }
```

When a node does not receive new time synchronization messages for ROOT_TIMEOUT number of message broadcast periods, it declares itself to be the root (myRootID = myID). Thus, after a ROOT_TIMEOUT period, there will be at least one, but possibly multiple roots in the network. Whenever a node receives a message with a rootID field smaller than its myRootID variable, it updates the variable according to the received rootID field. This mechanism ensures that roots with higher IDs give up their status and eventually there will be only one root—the node with the smallest ID—in the whole network. The pseudo-code describing this root election protocol is given at lines 3–4, 10–11 in Figure and 3–7 in Figure. The heartBeats variable contains the number of message broadcast periods since the last synchronization point was inserted into the regression table.

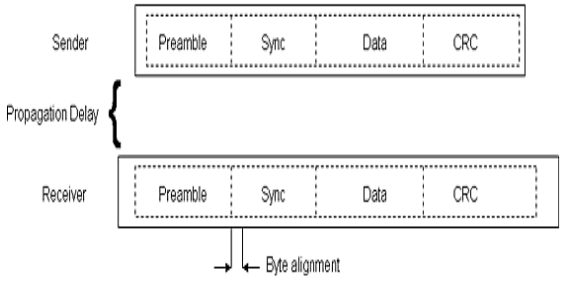
During the election process special care is taken to avoid inconsistencies and maintain the synchronized state of the network as much as possible. Only the root and synchronized nodes, those that have enough entries (at least NUMENTRIES_LIMIT many) in their linear regression table, transmit time synchronization messages. If a node receives a new reference point that disagrees with previous estimates of the global time, then it clears its regression table Finally, a newly elected root preserves its estimated skew and offset parameters (does not clear its regression table) and broadcasts the global time accordingly. This way the network will not get out of synchronization during the reelection phase. Another root related problem may arise if a new node is introduced to the network with a lower ID than the current root of the network. The new root does not start transmitting its own synchronization messages for the ROOT_TIMEOUT period, while it collects reference points to estimate its own skew and offset values. Thus, the global time broadcasted by the new root will be synchronized to the previous global time

```
1  event Timer.fired()
2  {
3    ++heartBeats;
4
5    if( myRootID != myID
6        && heartBeats >= ROOT_TIMEOUT )
7      myRootID = myID;
8
9    if( numEntries >= NUMENTRIES_LIMIT
10       || myRootID == myID ){
11     msg.rootID = myRootID;
12     msg.seqNum = highestSeqNum;
13     Radio.send(msg);
14
15     if( myRootID == myID )
16       ++highestSeqNum;
17   }
18 }
```

E. Advantages of FTSP

There are several advantages to FTSP, which it has improved on TPSN. Although TPSN did provide a protocol for a multi-hop network, it did not handle topology changes well. TPSN would have to reinitiate the level discovery phase if the root node changed or the topology changes. This would induce more network traffic and create additional overhead.

FTSP is robust in that is utilizes the flooding of synchronization messages to combat link and node failure. The flooding also provides the ability for dynamic topology changes. The protocol specifies the root node will be periodically reelected, so a dynamic topology is necessary. Like TPSN, FTSP also provides MAC layer time stamping which greatly increases the precision and reduces jitter. This will eliminate all but the propagation time error. It utilizes the multiple time stampings and linear regression to estimate clock drift and offset.



The data packets transmitted with FTSP are constructed as shown in figure 4. There is a preamble then sync bytes

followed by the data then finally the CRC. The dashed lines in the figure indicate the actual bytes in the packet and the solid line indicate the bytes in the buffer. When the sender is transmitting the preamble bytes, the receiver adjusts to the carrier frequency. Once the sync bits are received, the receiver can calculate the bit offset needed to accurately recreate the message. The time stamps are located at the boundaries of the sync bytes.

Allowing for dynamic topology changes, robustness for node and link failure, and MAC layer time stamping for precision are the major advantages of FTSP. It provides a low bandwidth flooding protocol to provide a network wide synchronization where all nodes are synchronized to the root node.

VII. NETWORK TIME PROTOCOL

The Network Time Protocol (NTP) is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks. In operation since before 1985, NTP is one of the oldest Internet protocols in current use. NTP was designed by David L. Mills of the University of Delaware.

NTP is intended to synchronize all participating computers to within a few milliseconds of Coordinated Universal Time (UTC): It uses the intersection algorithm, a modified version of Marzullo's algorithm, to select accurate time servers and is designed to mitigate the effects of variable network latency. NTP can usually maintain time to within tens of milliseconds over the public Internet, and can achieve better than one millisecond accuracy in local area networks under ideal conditions. Asymmetric routes and network congestion can cause errors of 100 ms or more.

The protocol is usually described in terms of a client-server model, but can as easily be used in peer-to-peer relationships where both peers consider the other to be a potential time source:20 Implementations send and receive timestamps using the User Datagram Protocol (UDP) on port number 123. They can also use broadcasting or multicasting, where clients passively listen to time updates after an initial round-trip calibrating exchange. NTP supplies a warning of any impending leap second adjustment, but no information about local time zones or daylight saving time is transmitted.

A. Definition

In this paper, the stability of a clock is how well it can maintain a constant frequency, the accuracy is how well its time compares to national standards and the precision is how precisely time can be resolved in a particular timekeeping system. The offset of two clocks is the time difference between them, while the skew is the frequency difference between them. The reliability of a timekeeping system is the fraction of the time it can be kept operating and connected in the network (without respect to stability and accuracy). Local clocks are maintained at designated time servers, which are timekeeping systems belonging to a synchronization subnet, in which each server measures the offsets between its local clock and the clocks of its neighbor servers or peers in the subnet. In this paper to synchronize frequency means to adjust the clocks in the subnet to run at the same frequency, to synchronize time means to set them to agree at a particular epoch with respect to coordinated universal time (UTC), as provided by national standards, and to synchronize clocks means to synchronize them in both frequency and time.

B. Performance Requirement

Internet transmission paths can have wide variation in delay and reliability due to traffic load, route selection, and facility outages. Stable frequency synchronization requires stable local-clock oscillators and multiple offset comparisons over relatively long periods of time, while reliable time synchronization requires carefully engineered selection algorithms and the use of redundant resources and diverse transmission paths. For instance, while only a few offset comparisons are usually adequate to determine local time in the Internet to within a few tens of milliseconds, dozens of measurements over some days are required to reliably stabilize frequency to a few milliseconds per day. Thus, the performance requirements of an internet-based time synchronization system are particularly demanding. A basic set of requirements must include the following:

1) The primary reference source(s) must be synchronized to national standards by wire, radio, or calibrated atomic clock. The time servers must deliver continuous local time based on UTC, even when leap seconds are inserted in the UTC timescale.

2) The time servers must provide accurate and precise time, even with relatively large delay variations on the transmission paths. This requires careful design of the filtering and combining algorithms, as well as an extremely stable local-clock oscillator and synchronization mechanism.

3) The synchronization subnet must be reliable and survivable, even under unstable network conditions and where connectivity may be lost for periods up to days. This requires redundant time servers and diverse transmission paths, as well as a dynamically reconfigurable subnet architecture.

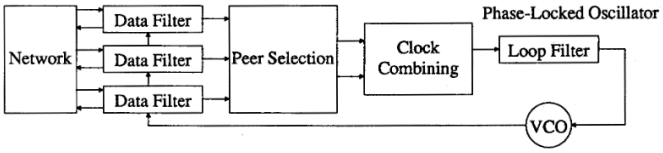
4) The synchronization protocol must operate continuously and provide update information at rates enough to compensate for the expected wander of the room-temperature quartz oscillators used in ordinary computer systems. It must operate efficiently with large numbers of time servers and clients in continuous-polled and procedure-call modes and in multicast and point-to-point configurations.

5) The system must operate in existing internets including a spectrum of machines ranging from personal workstations to supercomputers but make minimal demands on the operating system and supporting services. Time-server software and especially client software must be easily installed and configured. In addition to the above, and in common with other generic, promiscuously distributed services, the system must include protection against accidental or willful intrusion and provide a comprehensive interface for network management. In NTP' address filtering is used for access control, while encrypted checksums are used for authentication. Network management presently uses a

proprietary protocol with provisions to migrate to standard protocols where available.

C. Modes of Operation

NTP time servers can operate in one of three service classes: multicast, procedure-call, and symmetric. These classes are distinguished by the number of peers involved, whether synchronization is to be given or received and whether state information is retained. The multicast class is intended for use on high speed LAN's with numerous workstations and where the highest accuracies are not required. In the typical scenario one or more-time servers operating in multicast mode send periodic NTP broadcasts. The workstation peers operating in client mode then determine the time based on an assumed delay in the order of a few milliseconds. By operating in multicast mode, the server announces its willingness to provide synchronization to many other peers, but to accept NTP messages from none of them



The procedure-call class is intended for operation with file servers and workstations requiring the highest accuracies or where multicast mode is unavailable or inappropriate. In the typical scenario a time server operating in client mode sends an NTP message to a peer operating in server mode, which then interchanges the addresses, inserts the requested time-stamps, recalculates the checksum and optional authenticator and returns the message immediately. By operating in client mode, a server announces its willingness to be synchronized by, but not provide synchronization to a peer. By operating in server mode, a server announces its willingness to provide synchronization to, but not be synchronized by a peer.

While the multicast and procedure-call classes may suffice on LAN's involving public time servers and perhaps many private workstation clients, the full generality of NTP requires distributed participation of several time servers arranged in a dynamically reconfigurable, hierarchically distributed con- figuration. This is the motivation for the symmetric modes (active and passive). By operating in these modes, a server announces its willingness to synchronize to or be synchronized by a peer, depending on the peer-selection algorithm. Symmetric active ode is designed for use by servers operating near the leaves (high stratum levels) of the synchronization subnet and with preconfigured peer addresses. Symmetric passive mode is designed for use by servers operating near the root (low stratum levels) and with a relatively large number of peers on a possibly intermittent basis.

When a pair of servers operating in symmetric modes first exchange messages, a loosely coupled connection or association is created. Each server creates an instantiation of the NTP protocol machine with persistent state variables; however, the main purpose of the protocol machine is not to assure delivery but to preserve timestamps and related information. In symmetric modes the servers refresh reachability status as each message is received and dissolve the association and recover state storage if this status has not been refreshed for a considerable time

D. Data Formats

All mathematical operations assumed in the protocol are two's-complement arithmetic with integer or fixed-point operands. Since NTP timestamps are cherished data and, in fact, represent the main product of the protocol, a special format has been established. An NTP timestamp is a 64 b unsigned fixed-point number, with the integer part in the first 32 b and the fraction part in the last 32 b and interpreted in standard seconds relative to UTC. When UTC began at Oh on January 1, 1972 the NTP clock was set to 2 272 060 800.0, representing the number of standard seconds since this time at Oh on January 1, 1900 (assuming no prior leap seconds). This format allows convenient multiple-precision arithmetic and conversion to other formats used by various protocols of the Internet suite. The precision of this representation is about 232 ms, which should be adequate for even the most exotic requirements. Note that since some time in 1968 the most significant bit of the 64 b field has been set and that the field will overflow some time in 2036. Should NTP be in use in 2036, some external means will be necessary to qualify time relative to 1900 and subsequent 136-year cycles. Historic timestamped data of such precision and requiring such qualification will be so precious that appropriate means should be readily conceived.

Timestamps are determined by copying the current value of the local clock to a timestamp variable when some significant event occurs, such as the arrival of a message. In some cases, a particular variable may not be available, such as when the server is rebooted, or the protocol is restarted. In these cases, the 64 b field is set to zero, indicating an invalid or undefined value. There exists a 232 ms interval, henceforth ignored, every 136 years when the 64 b field will naturally become zero and thus be considered invalid.

E. Clock Strata

NTP uses a hierarchical, semi-layered system of time sources. Each level of this hierarchy is termed a *stratum* and is assigned a number starting with zero for the reference clock at the top. A server synchronized to a stratum n server runs at stratum $n + 1$. The number represents the distance from the reference clock and is used to prevent cyclical dependencies in the hierarchy. Stratum is not always an indication of quality or reliability; it is common to find stratum 3-time sources that are higher quality than other stratum 2-time sources. A brief description of strata 0, 1, 2 and 3 is provided below:

Stratum 0

These are high-precision timekeeping devices such as atomic clocks, GPS or other radio clocks. They generate a very accurate pulse per second signal that triggers an interrupt and timestamp on a connected computer. Stratum 0 devices are also known as reference clocks.

Stratum 1

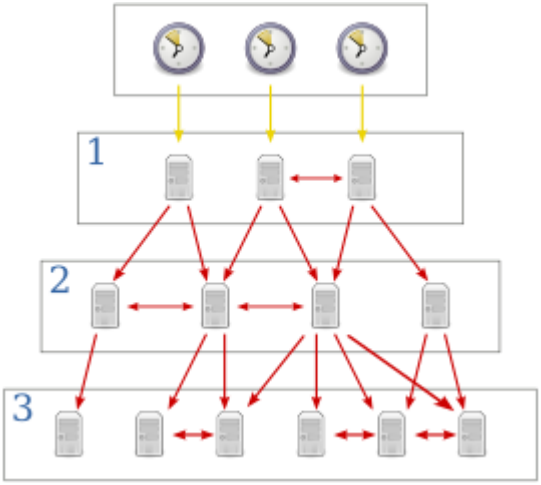
These are computers whose system time is synchronized to within a few microseconds of their attached stratum 0 devices. Stratum 1 servers may peer with other stratum 1 servers for sanity check and backup. They are also referred to as primary time servers.

Stratum 2

These are computers that are synchronized over a network to stratum 1 servers. Often a stratum 2 computer will query several stratum 1 servers. Stratum 2 computers may also peer with other stratum 2 computers to provide more stable and robust time for all devices in the peer group.

Stratum 3

These are computers that are synchronized to stratum 2 servers. They employ the same algorithms for peering and data sampling as stratum 2, and can themselves act as servers for stratum 4 computers, and so on.



REF:https://upload.wikimedia.org/wikipedia/commons/thumb/c/c9/Network_Time_Protocol_servers_and_clients.svg/269px-Network_Time_Protocol_servers_and_clients.svg.png

The upper limit for stratum is 15; stratum 16 is used to indicate that a device is unsynchronized. The NTP algorithms on each computer interact to construct a Bellman-Ford shortest-path spanning tree, to minimize the accumulated round-trip delay to the stratum 1 servers for all the clients.

F. ALGORITHM

A typical NTP client will regularly poll one or more NTP servers. To synchronize its clock, the client must compute its time offset and round-trip delay. Time offset θ is defined by

$$\theta = \frac{(t_1 - t_0) + (t_2 - t_3)}{2}$$

and the round-trip delay δ by

$$\delta = (t_3 - t_0) - (t_2 - t_1)$$

where

t0 is the client's timestamp of the request packet transmission,

t1 is the server's timestamp of the request packet reception,

t2 is the server's timestamp of the response packet transmission and

t3 is the client's timestamp of the response packet reception.

The values for θ and δ are passed through filters and subjected to statistical analysis. Outliers are discarded and an estimate of time offset is derived from the best three remaining candidates. The clock frequency is then adjusted to reduce the offset gradually, creating a feedback loop.

```
Checking current status of NTP service with ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
-ntp.nmi.nl      .PPS.            1 u   47   64  377   12.065    0.206   0.165
+ntp0.nl.uu.net  .PPS.            1 u   19   64  377   10.083    0.105   0.462
-ntp1.nl.uu.net  .PPS.            1 u    3   64  377   10.114    0.141   8.548
-chime2.surfnet. .GPS.            1 u   54   64  377   11.905   -0.293   0.319
+chime5.surfnet. .PPS.            1 u   40   64  377   12.508   -0.115   0.298
*metronom.dmz.c .PPS.            1 u   54   64  377   11.281   -0.037   0.174
+ntp4.bit.nl     .PPS.            1 u   48   64  377    9.915   -0.020   0.139
-ntp1.oma.be     .MRS.            1 u   34   64  377   13.661    0.220   0.201
+ntp2.oma.be     .PPS.            1 u   31   64  377   14.064   -0.006   0.226
#ntp-p1.obspm.fr .TS-4.           1 u   19   64  377   21.457   -1.277   0.434
#metasntp11.adni .PPS.            1 u    6   64  377   27.033    0.959   0.649
-ptbtime1.ptb.de .PTB.            1 u   45   64  377   28.312   -0.250   0.225
#139.143.5.30    139.143.45.11    2 u   59   64  377   17.301    0.339   1.936
#91.148.192.49 ( 193.67.79.202    2 u   40   64  377    8.725    0.154   0.374
#antongleuf.giga 193.67.79.202    2 u   58   64  377    9.019    0.302   0.220
#intimideer.1afa 5.200.6.34       3 u   28   64  377   10.155   -0.490   0.415
(Auto-Refresh every 10s  --- CTRL+C to Cancel)
```

The NTP management protocol utility ntpq being used to query the state of a stratum 2 server.

Accurate synchronization is achieved when both the incoming and outgoing routes between the client and the server have symmetrical nominal delay. If the routes do not have a common nominal delay, there will be a systematic bias of half the difference between the forward and backward travel times

VIII. HYBRID SYNCHRONIZATION ALGORITHM

A. Introduction

Sensor networks have recently emerged as a platform for several important surveillance and control applications. Sensor nodes are typically less mobile, more limited in capabilities, and more densely deployed than mobile ad hoc networks (MANETs). This necessitates devising novel energy-efficient solutions to some of the conventional wireless networking problems, such as medium access control, routing, self-organization, bandwidth allocation, and security. Exploiting the tradeoffs among energy, accuracy, and latency, and using hierarchical (tiered) architectures are important techniques for prolonging the network lifetime.

Network lifetime can be defined as the time elapsed until the first node (or the last node) in the network depletes its energy (dies). For example, in a military field where sensors are monitoring chemical activity, the lifetime of a sensor is critical for maximum field coverage. Energy consumption in a sensor node can be attributed to either “useful” or “wasteful” sources. Useful energy consumption can be due to

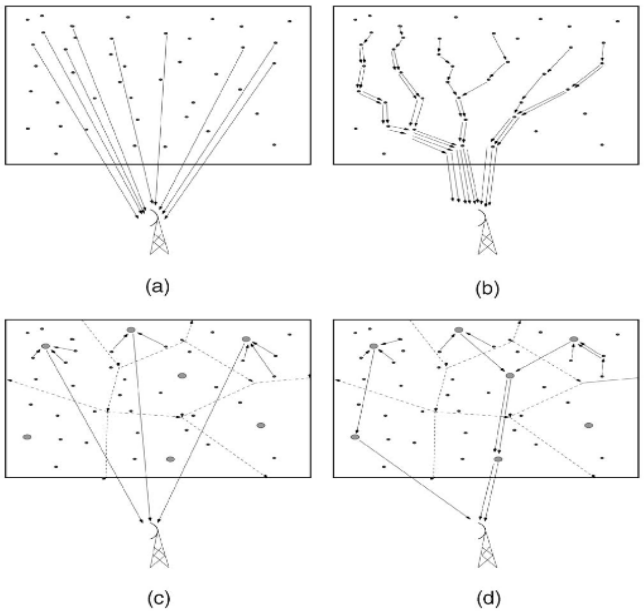
1. transmitting/receiving data,
2. processing query requests, and
3. forwarding queries/data to neighboring nodes.

Wasteful energy consumption can be due to

1. idle listening to the media,
2. retransmitting due to packet collisions,
3. overhearing, and
4. generating/handling control packets.

A number of protocols have been proposed to reduce useful energy consumption. These protocols can be classified into three classes. Protocols in the first class control the transmission power level at each node by increasing network capacity while keeping the network connected. Protocols in the second class make routing decisions based on power optimization goals. Protocols in the third class control the network topology by determining which nodes should participate in the network operation (be awake) and which should not (remain asleep). Nodes in this case, however, require knowledge of their locations via GPS-capable antennae or message exchange.

Hierarchical (clustering) techniques can aid in reducing useful energy consumption [8]. Clustering is particularly useful for applications that require scalability to hundreds or thousands of nodes. Scalability in this context implies the need for load balancing, efficient resource utilization, and data aggregation. Routing protocols can also employ clustering. Clustering can be extremely effective in one-to-many, many-to-one, one-to-any, or one-to-all (broadcast) communication.



constructed. Of course, node popularity due to interest in the data it provides can only be reduced by deploying several redundant nodes and rotating among them.

The essential operation in sensor node clustering is to select a set of cluster heads from the set of nodes in the network, and then cluster the remaining nodes with these heads. Cluster heads are responsible for coordination among the nodes within their clusters and aggregation of their data (intracluster coordination), and communication with each other and/or with external observers on behalf of their clusters (intercluster communication). Figure depicts an application where sensors periodically transmit information to a remote observer (e.g., a base station). The figure illustrates that clustering can reduce the communication overhead for both single-hop and multihop networks. Periodic reclustering can select nodes with higher residual energy to act as cluster heads. Network lifetime is prolonged through

1.reducing the number of nodes contending for channel access.
2. summarizing information and updates at the cluster heads, and
3. routing through an overlay among cluster heads, which has a relatively small network diameter.

B. Assumption

All nodes are assumed to have similar capabilities and equal significance and each sensor node has a unique identifier. Each node has a local clock, which gives the node the only notion of time. There exists a single static sink in the network and the network is assumed to have burst of activities. All sensor nodes are deployed densely, and the network is divided into clusters; every cluster has a CH and cluster members, which are assumed static. The clusters are multi-hop clusters to achieve better energy efficiency and scalability. There is a mix of links: both unidirectional and bidirectional links.

C. System Model

The network is represented by a graph G (V, E). V is the set of nodes and is divided into different sub graphs or cliques, GCP= {G1, G2, ..., Gn}. Each clique is considered a cluster and it is formed using a multi-hop clustering algorithm. The CH in each clique gathers information from all the cluster members. Two cliques are connected through links to gateway nodes (nodes common to two or more clusters) and the CH of each cluster. The system model considers that the nodes inside the cluster synchronize with the timing of the CHs and that all CHs synchronize with the timing of the sink. The application considered in developing the system model considers a certain tolerance allowed between cluster members and CHs throughout the life of the network.

D. Problem statement and methodology

The synchronization problem considered here synchronizes all nodes in the network for inter- and intra-cluster communication. The mechanism proposed to solve this

synchronization problem consists of a hybrid approach as a combination of the following two approaches, •Sender-receiver synchronization [1, 6] between sink and CHs providing higher accuracy and better scalability. •Diffusion-based synchronization [1, 7] between CHs and cluster members reducing energy consumption and improving the message tolerance.

E. Hybrid simulation mechanisam

The inter-cluster synchronization phase takes the graph GCH = (VCH, ECH) as input where VCH is the set of CHs and the sink in the network, and ECH is the set of edges connecting the CHs and the sink. The next step synchronizes each CH with the sink and collects the correct timing information from the sink for the sink to synchronize the CH's clock accordingly.

The sender-receiver synchronization used in inter-cluster has two phases: 1) level discovery and 2) synchronization. During the level discovery phase, the root node, sink, is assigned a level 0 and broadcasts the level_discovery packet. It contains the identity and level of the sender. The neighboring CHs receive this packet and assign themselves a level incremented with one compared to the received level. Once they have their level assigned, the CHs broadcast a new level_discovery packet containing their own level. The process continues until all CHs in the network have their level assigned. Any received level higher than the already assigned level is ignored.

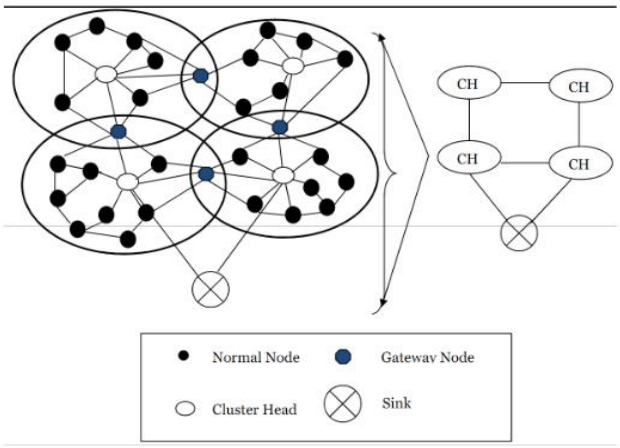
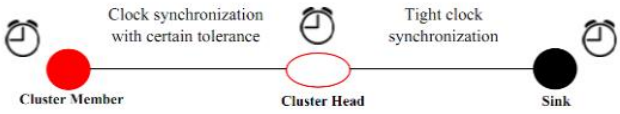


Figure 1. System model



In the synchronization phase, the CHs are synchronized with the sink using a two-way message exchange performed along each edge of the hierarchical structure established in the level discovery phase. CHs neighboring the sink send a synchronization_pulse packet to the sink. The packet contains the level of the CH and a value of time, T1. The sink receives this packet at time, T2, where T2 = T1 + Δ + d, Δ and d represent the clock drift between the two nodes and the propagation delay respectively. The clock drift considered here does not considers queuing delay. At time T3, the sink sends an acknowledgement packet to the CH along with the values of T1, T2, and T3. The CH receives the packet at T4and, by knowing the clock drift; the CH corrects its clock accordingly so that it is synchronized with the sink. CHs that are not neighboring the sink synchronize with their neighboring CHs, which are already synchronized the intra-cluster synchronization phase assumes that the network G (V, E) is divided into a number of clusters, GCP. In addition, it assumes that the CH of each cluster is already synchronized through the inter-cluster synchronization phase. The individual cluster members in each cluster are then synchronized using the diffusion-based synchronization algorithm by allowing a certain tolerance ranging randomly from 10s to 60s from the ideal time. Here, CHCP is considered the leader of the cluster, GCP, and the cluster members synchronize with it. Therefore, no specific election/re-election procedure for choosing a leader is required.

F. Conclusion

A wireless sensor network is a multi-hop ad hoc network of hundreds or thousands of sensor devices. The sensor nodes collect useful information such as sound, temperature, and light. Moreover, they play a role as the router by communicating through wireless channels under battery constraints. Time synchronization is required in many applications of distributed system. The comparison of various protocols and their results bring out some facts that can be used before deploying any network. NTP which is famous protocol for network also proves its performance in sensor network in term of accuracy, but it lacks in term of energy consumption. It consumes the more energy as compared to other protocols and therefore is not suitable for sensor networks. RBS is another protocol that lacks in accuracy first, but its accuracy increases as the resynchronization increases. Thus, initial accuracy is not provided by this protocol and hence avoided in circumstances where initial accuracy is desired. FTSP perform very well in terms of energy and accuracy, but as discussed in previous section the accuracy decreases with increase in multiple hops. So, it is not fit for multi hop networks but provide a good accuracy with limited hop of network. Based on the result a new Hybrid type of method that can take the features of these protocols can be designed. The ideas presented here could also be fully or partially applied to improve the performance of existing protocols. Experimental performance evaluation and comparisons with other existing protocols represent an open research problem.

References

[1] http://www3.cs.stonybrook.edu/~jgao/CSE590-spring11/91-ntp.pdf. (references)

[2] Saurabh Ganeriwal, Ram Kumar, Mani B. Srivastava, “Timing-Sync Protocol for Sensor Networks”, ACM SenSys '03, November 5-7, Los Angeles, CA, USA, 2003, 138-14

[3] Bharath Sundararaman, Ugo Buy, Ajay D. Kshemkalyani, “Clock Synchronization for Wireless Sensor Networks: A Survey”, Elsevier Ad Hoc Networks, Vol. 3, 2005, 281-323.

[4] Kopetz, H., and Schwabl, W.Global time in distributed real-time systems. Technical Report 15/89, Technische Universitat Wien, 1989.

[5] Yang, H., and Sikdar, B.A Protocol for Tracking Mobile Targets using Sensor Networks. IEEE Workshop on Sensor Network Protocols and Applications,May 2003.

[6] Elson, J., Estrin, D. (2002). "Fine-Grained Network Time Synchronization using Reference Broadcast.", The Fifth Symposium on Operating Systems Design and Implementation (OSDI), p. 147-163

[7] Tian, Z., Luo, X., Giannakis, G.B., "Cross-layer sensor network synchronization.", *Signals, Systems and Computers. Conference Record of the Thirty-Eighth Asilomar Conference*, Volume 1, 7-10, p. 1276 - 1280.

[8] Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., and Anderson, J.Wireless Sensor Networks for Habitat Monitoring. ACM International Workshop on Wireless Sensor Networks and Applications, p. 88–97, September2002.

[9] Schwiebert, L., Gupta, S., and Weinmann, J.Research Challenges in Wireless Networks of Biomedical Sensors.SIGMOBILE 2001, p. 151–165, July 2001.

[10] Mills, D. L. Internet Time Synchronization: The Network Time Protocol.IEEE Transactions on Communications COM 39 no. 10, p. 1482–1493, October 1991

[11] Kahn, J. M., Katz, R. H., and Pister, K. S. J. Mobile Networking for Smart Dust.In Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, p. 271–278, August 1999.