MINT 709, Capstone Project Report

A Comparative Study of Time Synchronization Protocols for Sensor Networks

Prepared by: Amardeep Singh Baweja

Prepared for: Dr. M.H. MacGregor

April 2011

Acknowledgements

This project report is part of the requirements for the degree of Master of Science in Internetworking at University of Alberta.

I would like to thank my project supervisor Dr. M.H. Macgregor, for his guidance, supervision, inspiration and support throughout the duration of the project, starting with the approval of my project definition document to the completion of project.

Over the last two years, while enrolled in the M.Sc. in Internetworking Program, I had a chance to meet and learn from faculty Professors, take interesting courses and refine my interpersonal and leadership skills. I would like to extend my thanks to all classmates and faculty members of the department for their kind cooperation during my M.Sc. studies.

I also acknowledge the help rendered by my beloved parents, without their constant support and inspiration I would not have reached at this stage.

Abstract

Wireless sensors network is a type of the Ad-hoc network. It is comprised of many sensors, interlinked with each other for performing the same function collectively such as monitoring the weather conditions, temperature, different kind of vibrations, sound etc. Any distributed system requires time synchronization. In particular, time synchronization is extremely important for wireless sensor network applications (e.g. for **Data Fusion**, **TDMA Schedules**, **Synchronizes Sleep Periods**, etc.).

In this project work, we study different time synchronization protocols available for sensor networks, like Reference Broadcast Synchronization **(RBS)**, Flooding Time Synchronization Protocol **(FTSP)** and Time Synchronization Protocol for Sensor Networks **(TPSN)**. Network Time Protocol **(NTP)**, which is very famous in the computer network**,** is also studied. The simulation of these protocols performed on the sensor network with the help of a simulator. The effects of these protocols on different parameters are studied and results obtained are compared in this project work.

Table of Contents

CHAPTER 4 **NETWORK DESIGN AND IMPLEMENTATION**

CHAPTER 5 **SIMULATION RESULTS**

CHAPTER 6 **CONCLUSION AND FUTURE WORK**

Chapter 1

Introduction

New advancement towards the minimization, reducing the cost and power requirements have motivated the researcher to have research in wireless sensor network. The wireless sensor network is a type of distributed network. The aim of many researchers is to create an environment that is rich of sensors. The deployment of sensors can be helpful in detecting the local condition of the environment like sound, temperature and movement of objects. There are a wide range of applications envisioned for such sensor networks, including microclimate studies, groundwater contaminant monitoring, precision agriculture, condition-based maintenance of machinery in complex environments, urban disaster prevention and response, and military interests. Traditional and existing architecture are not serving these applications.

1.1 Wireless sensor network

High-Powered, long-range sensors are well suited for unobstructed environments for example sky observations by weather radar but these sensors are not effective in complex cluttered environments where line-of-sight paths are generally very short. Thus using more short-range sensors closer to their targets can reduce the effect of clutter. In many scenarios, deploying wired sensors in large areas is impractical due to requirement of co-deployed infrastructure. Putting manual observation in fields such as environmental monitoring is not only time consuming but also it requires a lot of labor to cover a large area**.** Moreover**,** the events in such type of environment are low enough that sometime even a few occurs in a day. Deploying a large number of sensors in these areas makes sure that the area is covered. Generally**,** the sensors are thrown instead of manually deploy to cover the area.

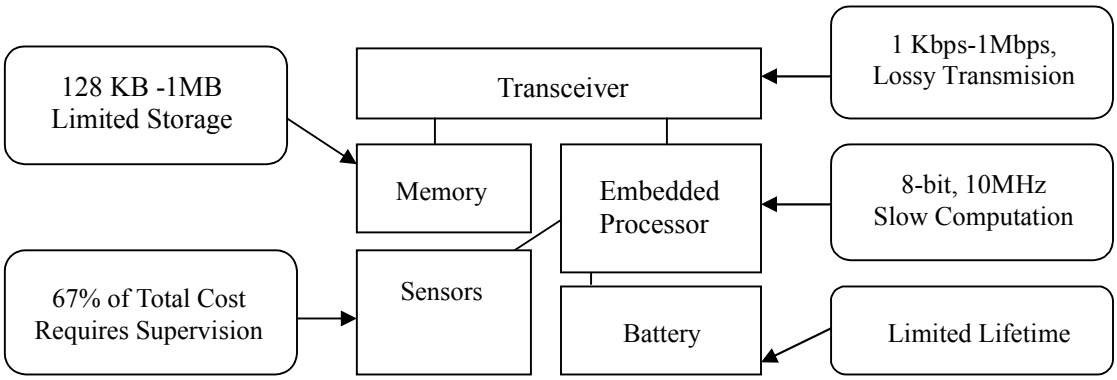

Figure 1.1 Hardware for Mica Mote.

Ad-hoc deployment of sensor network can observe the environment in a different way than other networks do. They may succeed in applications where traditional solutions have failed. This vision has captured the interest and imagination of many scientists. Sensor networks allow us to reveal the new phenomena that are unobservable until now in the environment. Such network will help in improving the environment in addition to observing it. The design of such systems poses serious challenges, and therefore active and broad research is going on for such system.

To facilitate easy deployment without an infrastructure, sensors are generally air thrown with many nodes, having only finite energy reserves from a battery. Unlike laptops or other handheld devices that enjoy constant attention and maintenance by humans, the scale of a sensor network's deployment will make replenishment of these reserves impossible. Different requirements of sensor network make them different from the traditional network**.** A common research aim is to reduce the overhead of collaboration in sensor network and increase the local processing in the sensors. The communication overhead is the main energy-consuming factor in the sensor network. The depletion of energy is mainly due to the communication that occurs in the sensor network. The communication is within the neighbors only.

The novel designs that are emerging allow users to put a task to the network with a high-level query such as “When the temperature goes beyond 20 degree put the sensor off for 5 minutes” or “report the time when there is a car passing from some particular area". If a node can correlate an incoming audio stream to the desired pattern locally and report only the time and location of a match, the system will be many orders of magnitude more efficient than one that transmits the complete time-series of sampled audio. The key to the energy efficiency is the use of local processing, hierarchical collaboration, and domain knowledge to convert data into increasingly distilled and high-level representations. A perfect system rather than incur the energy expense of transmitting raw sensor values further along the path to the user, the system will reduce the data as much as possible.

Dynamic nature of sensor network is another fundamental property of sensor networks. As the time passes, the nodes fail due to energy. Other type of failure can be due to overheat in the sun, can drain out with wind, or can crush by a wild animal. The range of nodes changes due to various environmental factors also. These changes are difficult to predict in advance. In case of sensor network, there may be a single human responsible for thousands of nodes in a dense sensor network. Any design in which each device requires individual attention is infeasible. Other requirement of self-configuring and adaptive to change in their environment is also arises due to the factors in sensor network.

The unique design requirements in sensor networks affect virtually every aspect of a system's design routing and addressing mechanisms, naming and binding services, application architectures, security mechanisms, and so forth. Development process in sensor network is also an iterative process that requires changes with according to the new need in sensor network.

1.2 Applications

WSN applications can be classified into two categories: Monitoring and Tracking (Fig. 1.2). These application are mentioned in a survey article.[6] Monitoring applications include indoor/outdoor environmental monitoring, health and wellness monitoring, power monitoring, inventory location monitoring, factory and process automation, and seismic and structural monitoring. Tracking applications include tracking objects, animals, humans, and vehicles. While there are many different applications, below we describe a few example of applications that have been deployed and tested in the real environment.

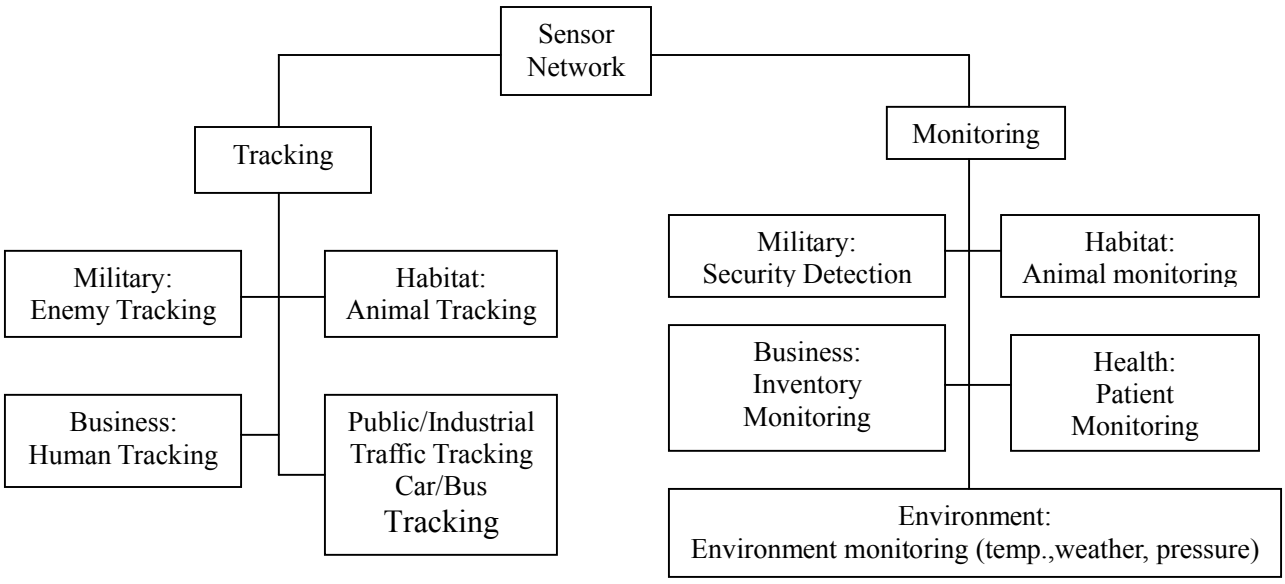

Figure 1.2 Overview of sensor applications

Health monitoring applications using WSN can improve the existing health care and patient monitoring. Five prototype designs have been developed for applications such as infant monitoring, alerting the deaf, blood pressure monitoring and tracking, and fire-fighter vital sign monitoring. These prototypes are discussed in an article [6]. The prototypes used two types of motes: T-mote sky devices and SHIMMER (Intel Digital Health Group's Sensing Health with Intelligence, Modularity, Mobility, and Experimental Re-usability). Because many infant die from sudden infant death syndrome (SIDS) each year, Sleep Safe is designed for monitoring an infant while they sleep. This

system detects the sleeping position of an infant and alerts the parent when the infant is lying on its stomach. Sleep Safe consists of two sensor motes. One SHIMMER mote is attached to an infant's clothing while a T-mote is connected to base station computer. The SHIMMER node has a three-axis accelerometer for sensing the infant's position relative to gravity. The SHIMMER node periodically sends packets to the base station for processing. Based on the size of the sensing window and the threshold set by the user, the data is processed to determine if the infant is on their back.

Macroscopic of redwood is a case study of a WSN that monitors and records the redwood trees in Sonoma, California [7]. Each sensor node measures air temperature, relative humidity, and photo-synthetically-active solar radiation. Sensor nodes are placed at different heights of the tree. Plant biologists track changes of spatial gradients in the microclimate around a redwood tree and validate their biological theories.

The use of sensor network in underwater monitoring, specifically for the purpose of coral reefs and fisheries are studied [8]. The sensor network consists of static and mobile underwater sensor nodes. The nodes communicate via point-to-point links using high-speed optical communications. Nodes broadcast using an acoustic protocol integrated in the Tiny OS Protocol Stack. They have a variety of sensing devices, including temperature and pressure sensing devices and cameras. Mobile nodes can locate and move above the static nodes to collect data and perform network maintenance functions for deployment, re-location, and recovery.

WSN equipments are smaller, lighter, and consume less power. The challenges of a WSN application for volcanic data collection include reliable event detection, efficient data collection, high data rates, and sparse deployment of nodes. Volcanic monitoring with WSN can help accelerate the deployment, installation, and maintenance process [9]. When an interesting event occurs, the node will route a message to the base station. If multiple nodes report the same event, then data is collected from the nodes in a round-robin fashion. When data collection is completed, the nodes return to sampling and storing sensor data locally.

1.3 Objective

In sensor networks, a confluence of factors makes flexible and robust time synchronization particularly important, while simultaneously making it more difficult to achieve than in traditional networks. Some nodes have large batteries and run all the time; others are so constrained that they only wake up occasionally, take a single sensor reading and transmit it before immediately returning to sleep. In this project work, we compare the existing time synchronization protocols and show the results based on different parameters selected. We try to show that the particular protocol is better in a particular situation and need of the application.

A common view of physical time is a basic requirement for nodes to reason about events that occur in the physical world. For example, precise time is needed to measure the time-of-flight of sound; distribute an acoustic beam-forming array; form a low-power TDMA radio schedule; integrate a time series of proximity detections into a velocity estimate; or suppress redundant messages by recognizing duplicate detections of the same event by different sensors.

Even in a traditional distributed system, creation of a synchronization scheme that satisfies requirements is challenging. The task becomes particularly difficult in sensor networks, in light of their additional domain requirements including energy efficiency, scalability through localized interactions, and automatic adaptation to dynamics. For example, the energy constraints violate a number of assumptions routinely made by classical synchronization algorithms: that using the CPU in moderation is free, listening to the network is free, and occasional transmissions have a negligible impact. Network and node dynamics require continuous, automatic configuration; this precludes a priori selection of a particular node as the master clock. A paradox of sensor networks, then, is that they make stronger demands on a time synchronization system than traditional distributed systems, while simultaneously limiting the resources available to achieve it.

1.4 Contributions

Computer clock synchronization is a well-studied problem, with a rich history stretching back to the advent of the first computer networks. NTP is a well-known protocol widely used over the internet for time synchronization. However, the unique system architecture seen in sensor networks brings a new dimension to many old problems. Our contributions fall into several categories:

1.4.1 We consider the various uses of time synchronization in detail, and describe the axes along which these applications can be characterized. A detailed understanding of the requirements is important in this realm where efficiency is so critical.

1.4.2 Based on our experience exploring this problem space, we propose several general guidelines for the use of time synchronization protocols in sensor networks. No single synchronization scheme can be optimal on all axes (e.g., precision, lifetime, scope, energy, availability), but most applications do not require peak performance on all axes.

- An ideal synchronization system will minimize its energy use by providing service that is exactly necessary and sufficient for the needs of the application. Tunable parameters can allow synchronization nodes to be matched more closely to the requirements of the application.
- Most existing time synchronization schemes make a common assumption: that their goal is to keep the clock synchronized all the time. Applications assume that they can query the clock at any time, and it will be synchronized. Another approach is to let clocks at sensors to run at their natural rate and when any event of interest occur the node time stamp the event with the clock of the cluster head. This has many advantages; for example, it enables post-facto synchronization, peer-to-peer synchronization, and participation in multiple timescales.

1.5 Chapter Outline

In the remainder of this project, we will consider time synchronization in sensor networks in more detail: its motivations, the inherent limitations, and our comparison results.

Chapter 2 describes how synchronized time is a critical service in sensor networks. It is a basic requirement for virtually all algorithms that reason about time-varying observations made by distributed sensors. This chapter reviews literature survey in clock synchronization. It also describes various challenges that should be considered during the design of time synchronization protocol for sensor networks.

Chapter 3 reviews related work in clock synchronization. This chapter discusses the results of related research**,** which compare time synchronization protocols. We explore a number of metrics that have found relevant for evaluating time synchronization in the sensor network domain in this related research.

Chapter 4 describes the OMNET++ simulator, used in this project for the purpose of simulation. The model adopt for simulation is discussed in detail.

In Chapter 5, we show the comparison results of our simulation of the time synchronization protocols.

Finally, in Chapter 6, we present our conclusions and describe directions for future research in this area.

Chapter 2

Literature Review

Now we summarize various existing synchronization protocols. We also discuss the relative advantages and disadvantages of these protocols. Given that, sensor networks are generally closely tied to the real-world environment that they monitor; different networks will have different characteristics affecting their synchronization requirements. For this reason, some of the protocols that we discuss below will be more suitable than others in some cases and vice versa. We will specifically consider the following protocols.

1. **Network Time Protocol (NTP)** [2] Network Time Protocol is used in computer network for time synchronization of the computers. The computer repeatedly sends request for time to the server and server responds accordingly. The computer then set the time according to the different time stamps received by the server.

2. **Reference Broadcast Synchronization (RBS)** [4], seeks to reduce non-deterministic latency using receiver-to-receiver synchronization and can also conserve energy via post-facto synchronization.

3. **Flooding Time Synchronization Protocol (FTSP)** [3] 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. MAC layer time stamping can eliminate many of the errors.

4. **Timing-Sync Protocol for Sensor Network (TPSN)** [5] aims at providing network-wide time synchronization in a sensor network. The algorithm works in two steps. In the first step, a hierarchical structure is established in the network and then a pair-wise synchronization is performed along the edges of this

structure to establish a global timescale throughout the network. Eventually bb nodes in the network synchronize their clocks to a reference node.

2.1. Network Time Protocol

In traditional computers, NTP is used for time synchronization. The network time protocol (NTP) [2] is the protocol that is used in traditional computer network to keep the clock synchronized. It provides coordinated universal time (UTC). NTP uses level of clock servers for the purpose of synchronization. Each level called stratum is assigned a different level starting with 0. The next level server has another level as 1 and the hierarchy is maintained for the levels. Normally stratum 0 is GPS clock or atomic clocks, where as stratum 1 are those servers that get time from stratum 0 level. Servers at stratum 1 act as time synchronization source for stratum 2 servers.

In NTP client sends multiple requests to the server and stores the pair of offset and delay for later calculations. The main disadvantage of NTP in the sensor network is that it needs to send multiple messages to the server for synchronization. More the messages are send more the energy of a sensor node is consumed. As sensors are limited in their energy the NTP consumes much energy, therefore a less energy consuming protocol for time synchronization is required for the sensor network.

2.2. Reference Broadcast Synchronization

The Reference Broadcast Synchronization (RBS) [4] protocol utilizes the concept of broadcast nature of wireless communication. According to this property, two receivers located within listening distance of the same sender will receive the same message at approximately the same time. It utilizes the concept that when we send the message using the physical layer than it will arrive at different receivers at the same time. The propagation delay in sensor network is minimal and the range of sensor network is very low. Therefore, the message send from physical layer will arrive at same time to different nodes. When the nodes receive the message they will record the time of arrival of that message and compare their clock with each other. This process will allow them to synchronize at high degree of precision. This protocol uses a sequence of synchronization

messages from a given sender in order to estimate both offset and skew of the local clocks relative to each other. The protocol exploits the concept of time-critical path, that is, the path of a message that contributes to non-deterministic errors in a protocol. Fig. 2.1 and Fig. 2.2 compare the time-critical path of traditional protocols, based on sender-to-receiver synchronization, with receiver-to-receiver synchronization in RBS.

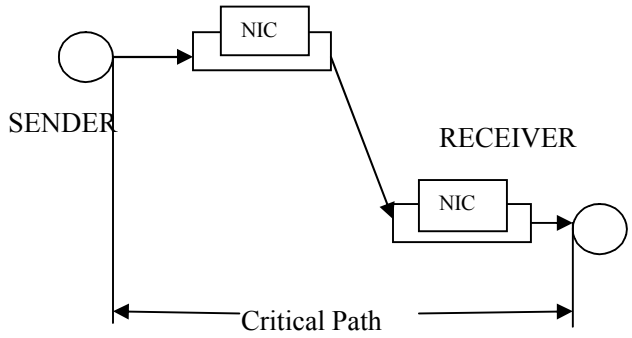

Figure 2.1: Critical path analysis for a protocol

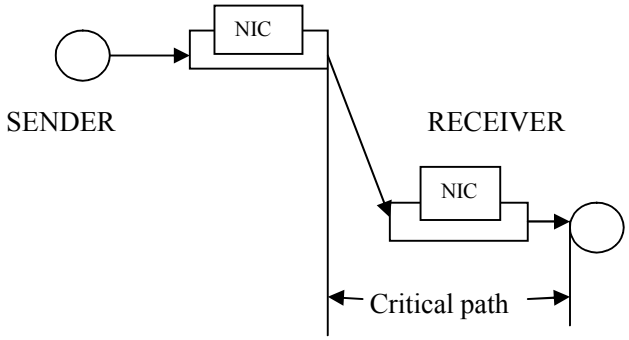

Figure 2.2: Critical path analysis for RBS protocol

Due to non-deterministic transmission delays it become difficult to synchronize the clock at very high accuracy. The delays that occur at the sender side eliminate by using the physical layer broadcast concept of sensor networks. In general, the time involved in sending a message from a sender to a receiver is the result of the following four factors, all of which can vary non-deterministically.

1. ***Send Time*:** The time spent by the sender for message construction and the time spent to transmit the message from the sender's host to the network interface.

2. ***Access Time*:** The time spent waiting to access the transmit channel.

3. ***Propagation Time*:** The time spent by the message to reach the receiver, once it has left the sender.

4. ***Receive Time*:** The time spent by the receiver to process the message.

By considering only the times at which a message reaches different receivers, the RBS Protocol directly removes two of the largest sources of non-determinism involved in message transmission, namely the send time and the access time. Thus, this protocol can provide a high degree of synchronization accuracy in sensor networks. Algorithms used in RBS to estimate the phase offset between the clocks of two receivers. RBS protocol can produce highly accurate results if each receiver can record its local clock reading as soon as the message receives. This is often the case for single-hop communication in a wireless network. In real scenario, it is possible that messages sent over a wireless sensor network can be corrupted. Also sometime a receiving node may not be able to record the time of message arrival promptly due to various reasons. One reason may be that node was busy with other computations when the message arrived. To solve this problem the protocol uses a sequence of reference messages from the same sender, rather than a single message. Any Receiver say x will compute its offset relative to any other Receiver y as the average of clock differences for each packet received by nodes x and y.

Post-Facto Synchronization: As the term post-facto means, this type of synchronization occur only when the requirement of synchronization occur in the network. The RBS can also run in this mode to save energy. In this mode, the clock runs independently and whenever the event of interest occurs only then the clocks needs to be synchronized. This technique is similar to reactive routing which work different to proactive routing. In reactive routing route is traced only when we need to send message. By synchronizing the nodes only when necessary, energy is conserved because the nodes can be switched to power-saving mode at all other times.

***Multi-hop Communication*:** Multi-hop Synchronization is required in sensor networks as they are large in numbers. The possibility of multiple hops in sensor network would introduce a large variety in message transmission time between multiple receivers of the same message. If this case is there then the RBS Protocol cannot maintain accuracy. To maintain the accuracy, another node that is lying in the intersection area of two neighbors is used**.** The support for multi-hop communication is a necessity in sensor network.

Advantages

1. The largest sources of error (send time and access time) are removed from the critical path by using physical broadcast medium of the sender.

2. Clock offset and skew are estimated independently of each other. Clock correction does not interfere with either estimation because local clocks are never modified according to RBS protocol.

3. Post-facto synchronization prevents energy from being wasted on expensive clock updates, as it works only when there is a necessity.

4. Multi-hop support is provided in RBS by using nodes that belongs to multiple neighborhoods (i.e. broadcasting domains) as interfaces.

5. This protocol is applicable to both wired and wireless networks.

6. Both absolute and relative timescales can be maintained in this protocol.

Disadvantages

1. The protocol is not applicable to point-to-point networks a broadcasting medium is required.

2. As the network size grows, the requirement of message exchange also grows. For a single-hop network of n nodes, this protocol requires O (n2) message exchanges.

3. Time required to synchronize the network, can be high due to the large number of message exchanges.

4. The reference sender, which is used to synchronize other nodes, is left unsynchronized in this method. If the reference sender needs to be synchronized, it will lead to a significant waste of energy in this protocol.

2.3 Flooding Time Synchronization Protocol

FTSP Protocol achieves time synchronization with very low error rate (in the microsecond range) and due to flooding property**;** it is scalable up to hundreds of nodes. This protocol is robust to network links and nodes failure due to any reasons. The FTSP maintain accuracy with MAC layer time-stamping algorithm and skew compensation with linear regression method [3]. The concept use in FTSP is already applied but the combination used in FTSP proved effective in wireless sensor network and hence provide good accuracy and less communication. It assume that each node has a local clock that are prone to timing errors due to nature of crystal clock and can communicate over an unreliable wireless link to its neighbors.

In FTSP**,** the synchronization between a sender and multiple receivers is obtained using a single message that is broadcasted by the sender to multiple receivers. The message is time stamped by both the sender and receivers at their MAC layers. The use of this type of time stamping i.e. at MAC layer will eliminate many errors. However, accurate time-synchronization at repeated different time is not a permanent solution. If some method can modify or change the clock drift of the nodes then this method will provide a permanent or a longer type of solution to the problem. Linear regression is used in FTSP to compensate for clock drift. Typical WSN operate in areas larger than the broadcast range of a single nod. The support of multi hop network is kept in mind during the designing phases of FTSP. The root of the network is a single node that is dynamically elected. This root node maintains the global time and synchronizes other nodes to the clock of root node. The FTSP use the ad-hoc type of structure as compared to the spanning tree type approach used in TPSN. The approach of FTSP saves the initial phase of maintaining the tree and this approach is more robust to different failures reasons in the network.

2.3.1 Time-Stamping

A Radio broadcasting concept is used in the FTSP to synchronize the receivers to the time provided by the root node. The broadcast message contain the time that is stamped by the root node. The use of Mac layer for time stamping the reception time of message reduced the errors that occur in node. The broadcast message provides a synchronization point to each of the receivers. Clock offset of receiver is estimated by finding the difference between global and local time of the receiver. In RBS Protocol, the time stamp is not incorporated in the message that is broadcast where as in FTSP the time stamp of the sender is embedded in the currently transmitted message. Therefore, the time stamping on the sender side must be performed before the message is transmitted to the receiver.

In FTSP, time stamping at Mac layer reduces different errors by time stamping the message on both the sender and the receiver side. The format of the message is made in such way that the time stamps are made at each byte boundary after the Sync byte. The byte transmission time is firstly subtracted by the time available in the stamp at the receiver side. The only time stamp obtained after deducting various errors is put into the message before transmission.

2.3.2 Clock drift management

In the real scenario if we have the clocks that works on same frequency always, setting the clock offset once will be sufficient. However**,** this is not the case and we need to send synchronization message repeatedly. However, the frequency differences of the crystals used in Mica2 motes introduce drifts up to 40μs per second**;** it is confirmed in the research article by the author who implements their protocol on Mica2 motes. The author also claims that it is mandatory to resynchronize the network with a period that is less than one second. If we do the resynchronization, less than a second than only we can achieve the accuracy within microseconds. Otherwise**,** the clock looses their accuracy due to skew in them. Sending resynchronization message very shortly is a big overhead in terms of energy as well as it also utilizes a lot of bandwidth in the network.

In FTSP**,** we estimate the clock drift of the receiver clock with the sender clock. The offset between the two clocks that we change with time stamp message broadcast will provide short-term stability of the clocks.

2.3.3 Multi-hop time synchronization

In practical WSN applications, it is not possible to have all the sensors with in direct reach of each other. Sensor is small device with limited broadcast range. The message to the other nodes in the network is sent using the concept of multi hop architecture. In multicast environment FTSP**,** assume that every node has different ID. The reference points are made in the network. This reference point act somewhere like server and provide the synchronization messages to the nodes that are not in direct range of root node. The root is a special node, which is dynamically reelected by the sensor network. The root node is the one to which the whole network is synchronized. A node that is in the broadcast radius of the root can collect reference points directly from it. Nodes outside the range of root can get reference points from the nodes that are located nearby to the root node. A node estimates the offset, skews of its own local clock, and synchronizes itself by adjusting its clock. This synchronized node now broadcast the synchronization message to other nodes in the network and process keep on going until the end nodes arrive.

2.4 Timing-Sync Protocol for Sensor Network

Timing-sync Protocol for Sensor Networks (TPSN) [5] is a sender-receiver approach. RBS is the protocol that works on receiver-receiver approach therefore the time stamp is not required in the network. TPSN require time stamping within the broadcast message to make it work. For sensor networks, the handshaking type approach is much effective as compare to receiver-receiver synchronization. This result came when we time stamp the packet at Mac layer. Time stamping at Mac layer is feasible in sensor networks. To prove this claim that in sensor network sender-receiver approach is better than receiver-receiver approach, author compares the performance of TPSN with Reference Broadcast Synchronization (RBS). The results obtained by the author by

implementing protocols on motes shows that TPSN gives 2x better performance than RBS.

In sensor networks, the requirement of clock synchronization may not arise all the time. Post facto method can also be used with TPSN for more energy conservation in the network. The protocol like TDMA requires the same global time throughout the network to work. For such scenario**,** a self-configuring system can be created and is suitable for sensor networks. In such algorithm, servers are maintained like NTP that provide the time synchronization message to other nodes. An effective and simple solution of time synchronization is provided by the TPSN for sensor network. Another property of TPSN is that it is flexible enough that one can tune it to meet the desired level of accuracy in the network.

TPSN has two different algorithms to provide synchronization. First algorithm is call “level discovery phase” in which levels are assigned to the node. The root node is assigned a level 0 and other nodes are assigned level according to their distance in hops from the root node. The node at level x must communicate only to node at level x-1. Even the author claim that this “level discovery phase” is not only required for time synchronization only. Much other application like localization, target tracking and aggregation used this phase and therefore, this phase should not be considered as an overhead to TPSN. TPSN is designed to improve the accuracy of the applications discussed.

After the implementation of first algorithm, the second algorithm starts its work. This algorithm is known as “synchronization phase” which actually synchronize the node at level x with node at x-1. This synchronization process repeated and finally the whole network synchronized. Each node in the network is synchronized to a single node called root node and in this way network wide time synchronization is achieved. Root node is one that acts as interface between the external world and the sensor network. The root node can also be equipped with GPS receiver to synchronize the sensor network to the real world.

In a scenario when it is not possible to have an external link and root node is not specific, in such case sensors node can periodically change their role and act as root node. Leader election is used to choose the root node in such situation. The node with maximum energy available is chosen as root node in most of the scenarios.

TPSN assume that there is bidirectional links in the network to do the synchronization. Having bidirectional links also help in creating the spanning tree, which is required to be created in level discovery phase. The maintaining of this hierarchical structure, which is formed in level discovery phase, is the responsibility of TPSN protocol itself. As this structure is used and required for many of the applications of sensor network, therefore, author claims that it should not be considered as an overhead to the protocol.

2.4.1 Level Discovery Phase

Whenever the network in deployed this phase of algorithm occurs first. The level 0 is assigned to root node and as the phase initiate it assign different levels to different nodes. The level discovery packet is used for the purpose. This packet contains the identity and the level of the sender. The neighbor nodes of the sender sensor receive this packet and assign themselves one greater level than that is in the packet. Once the level of a node is sent, it put its level in the packet and sends it to its neighbors. A sensor node check the level in the packet received. It neglects the packet if it already has level equal to that level. This process is repeated with every node and finally the hierarchy structure is created. In the paper Author, also propose that the other flooding type of technique can be used to create the hierarchy structure instead of using spanning tree. However, that approach will not provide as accurate structure as the spanning tree provides.

2.4.2 Synchronization Phase

Pair wise synchronization is performed in this phase. The synchronization is performed along the edges of the hierarchical structure that is maintained by the level discovery phase. The figure below analyzes the message exchange between two nodes.

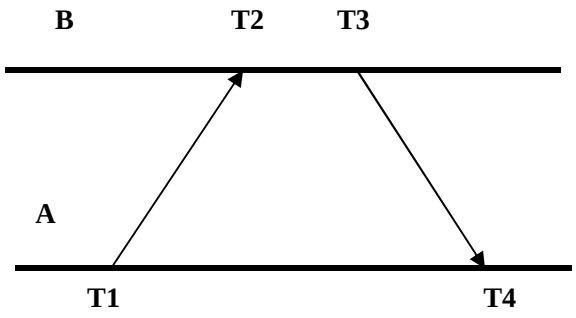

Figure.2.3 Two-way message exchange between nodes

Figure shows this message-exchange between nodes '*A*' and '*B*'. Here, *T1, T4* represent the time measured by local clock of '*A*'. Similarly**,** *T2, T3* represent the time measured by local clock of '*B*'. At time *T1*, '*A*' sends synchronization packet to '*B*'. The synchronization packet contains the level number of '*A*' and the value of *T1*. Node *B* receives this packet at *T2,* where *T2* is equal to *T1* + D + *d*. Here D and *d* represents the clock drift between the two nodes and propagation delay respectively. At time *T3*, '*B*' sends back an *acknowledgement* packet to '*A*'. The *acknowledgement* packet contains the level number of '*B*' and the values of *T1, T2* and *T3*. Node *A* receives the packet at *T4.* Assuming that the clock drift and the propagation delay do not change in this small span of time, '*A*' can calculate the clock drift and propagation delay as:

$$\boldsymbol{\Delta = \frac{(T2 - T1) - (T4 - T3)}{2}}$$

……. (1)

$$d = \frac{(T2 - T1) + (T4 - T3)}{2}$$

When the drift is calculated, Node *A* can correct its clock; so that it synchronizes to Node *B*. TPSN is a sender-initiated approach, where the sender synchronizes its clock to that of the receiver. The root node first sends the packet to all. This packet is called time synchronization packet that intimates that root is ready to distribute message. When a node receives the packet it waits for some time and then request the time from the root node. The concept of random wait is incorporated so that more than one node on receiving the time synchronization message should request the time at a same moment. Root node replies the request for time stamp with a message that contain time stamp. Sensor node on receiving the message synchronizes itself to the root node. The process is repeated for each level and network wide synchronization is achieved.

Chapter 3

Related work

3.1 FTSP Related Work

In FTSP, every node must broadcast its time stamp message upon receiving beacons to the other nodes in a hop area so that they can estimate the relative clock offsets among each other. The performance of FTSP is also very good as compared to other protocols, but experimental results shown in paper [11] show that as more the number of hops in a multi hop network the accuracy of FTSP decreases. The efficiency of FTSP is checked with the help of simulation and it provides the decrease in accuracy where the number of hops is more [12]. Therefore, FTSP can be used in the networks where the number of hops is few. In our simulation scenario, the decrease in accuracy is not much visible due to limited number of hops. Our model assumes only 2-hop network and hence cannot show the decrease in accuracy of FTSP. Implementation of FTSP on mica and mica2 for experimental results reveals that FTSP removes various jitters as compared to RBS with the help of time stamping at the MAC layer [3]. Our simulation model also proves the same concept, when we show that the accuracy of FTSP is much better than that of RBS protocol.

3.2 NTP Related Work

NTP is a famous synchronization method for setting the clocks of computer. In our project work**,** we simulate the NTP in senor networks. NTP is developed for computer network, assumes that the sending and receiving the messages in the network is free. In paper**,** named post-facto synchronization [16], the accuracy of NTP is found to be very accurate at the level of 1 microsecond. Our simulation results also prove that the accuracy provide by NTP is greater than all the other protocols. As sensor network are very energy constraints, therefore, NTP is not fit for sensor network despite the fact that it provide high accuracy. As discussed in article [13], wireless sensor nodes have onboard

memory, for example Crossbow's Mica2 and Mica2Dot sensors. As large the sensor network is, large is the requirement of memory for storing files and messages. If these files can be programmed to store into each node, it would leave very little memory to hold the data monitored by the sensor, limiting NTP use for WSN. This paper discusses the issue of storage in NTP over sensor network.

3.3 TPSN Related Work

TPSN proposed by S. Ganeriwal uses two-way communications that increase the message load on the network. It is simulated in two phases in our model. The first phase is the root discovery phase and other is the synchronization phase. The output result shows that the accuracy of TPSN is more than RBS, which is also concluded in other paper [14]. The accuracy of TPSN is lower than FTSP in our results, which is due to the fact that TPSN does not lose the accuracy in multi hop network, where as FTSP loses the accuracy with increase in hops in the network. Root discovery phase in TPSN generates extra messages. The flooding used in FTSP eliminates the need of root discovery phase. Due to this root discovery phase the TPSN is in need of more messages as compared to FTSP and hence more energy consumption results. TPSN is a time synchronization method that provides a balance between the energy consumption and accuracy. It consumes more energy as compared to FTSP but it provides accuracy much better than FTSP [5] in multi hop networks. The performance analysis of RBS and TPSN is done in paper [5]. The author claim that the accuracy of RBS is not achieved with one synchronization message. Our simulation results show the same output that the RBS accuracy is increases with increase in resynchronization time.

3.4 RBS Related Work

Since RBS does not perform any time synchronization based on UTC or other external source it eliminate the error occurrence from the unexpected sending and accessing time. In the case of multi-hop network, it again is not as suitable as TPSN is because the common node that acts as interface between the two hops if dies, need to be chosen again. The TPSN due to its level discovery phase overcome these issues. The

chances to die of common node are more as it is more involved in time synchronization. More a node is involved more is the chance to exhaust its energy and sooner it dies. Moreover, the working of RBS in multi hop network is not suitable for the network as discussed in paper [14]. The lifetime of sensor is shorten with multiple messages are being exchanged between the nodes. Similar kind of results is available with our simulation. Another factor that affects the credibility of RBS is that it requires more resynchronization to achieve its best accuracy. Our simulation result prove that the accuracy of RBS keep on increasing with resynchronization time. It is not possible to achieve its accuracy with the single synchronization message and its accuracy keep on increasing with the increase of resynchronization messages. In research article [15], the work is based on the assumption that by reducing transmission rate the energy used can be improved and hence increase the network lifetime. The goal was not to make high accurate time synchronization but to make it energy efficient. The comparative energy efficiency analysis is performed over Micas motes. Motes use Tiny OS operating system. For comparison, purpose protocols are set with each parent node with 3 or less children. They conclude that Sender -receiver synchronization is good for large networks and receiver-receiver synchronization work better in smaller network. In our simulation criteria, a small network is modeled and RBS, which is based on receiver- receiver synchronization, perform well in our model.

3.5 Comparison of TPSN and RBS

In multi hop Ad-hoc networks, a depleted sensor drop information that came from other sensors node through it. This factor affects the area monitored by the network. Considering the various situations the performance of TPSN and RBS is compared on Mat lab [5] and a new hybrid type of protocol is proposed. The author compares their results on different parameters and according to their result; a new hybrid type of algorithm is devised. We refer to the paper for the purpose of comparison results that they found between TPSN and RBS. TPSN and RBS both achieve accurate clock synchronization but they both worked for small network. Although these algorithms will work for large network but as the nodes start losing power they become inefficient. TPSN

does not perform well in sparse network where as RBS deteriorates in denser network. Our simulation results show that the TPSN accuracy is more than that of RBS network in initial phases. Our model is not a dense network and it comes under the category of sparse network where RBS outperform TPSN in later resynchronization phases.

Chapter 4

Network Design and Implementation

Simulation is a relatively fast means to obtain an estimate of network performance and tuning. The most widely used simulators for WSNs are OMNET++, CASTALIA, NETWORK SIMULATOR 2 (ns-2), and JAVA SIMULATOR (J-Sim). OMNET++ version 4.0 is used in our project work for simulating different algorithm and comparison with each other. OMNET++ is an object-oriented modular discrete event simulator. The name itself stands for Objective Modular Network Testbed in C++.

4.1 Introduction to OMNET++ Simulator

The OMNET++ is an Integrated Development Environment based on Eclipse platform, used to perform batch execution and analyzing simulation results. OMNET++ include an NED editor, which can edit NED files both graphically and in text mode. The user can choose to use any mode any time using tabs at the bottom of the window. The compound modules, channels and other types of components can be created easily with graphic mode. The editor offers many features that are just drag, drop, and reduce the effort of coding the things. Graphical features are also incorporated that can be used to change the background image, icon image**,** coloring etc**.** Text mode allow user to work with NED source directly. The simulation for the completion of keywords parameter and sub modules name provides the Ctrl + Space key combination. The NED source is continually parsed and validated as the user is typing and errors are displayed in the real time on the left margin.

Another type of feature that provided is INI file editor that lets the user to configure simulation model for execution. The structure of INI file is also visualized and editable via drag and drop and dialogs in OMNET++ simulator. Each running process sends its output to a separate console buffer within the Console View, so the user can review the output after a simulation has finished. One can switch between console buffers using the console view menu or toolbar.

4.2 Network Simulation Model

The network consists of a root node and many sensor nodes. The nodes that are in reach of sensor node are called common nodes, as they are the nodes that act as interface between the end sensors and the root node. The time obtained from the root node is sent to the rest of the network with the help of these common nodes. Delays are calculated according to the protocols. We also assume that the messages generated for the synchronization are of same length. The common nodes are in the range of root node and the end nodes are in the range of common nodes. Root cannot send a message directly to the end nodes, as they are not in the reach of the root node. We named root node as sink node according to the name proposed in the simulating protocol. Each protocol needs to send the synchronization message repeatedly. The value of time at which we need to send message again is user dependent. Before starting the simulation**,** we ask the user to enter the resynchronization time for the protocol. After the specified time the network again forces the root node to send the synchronization method.

The simulation model chosen for the simulation of different methods are given as:

- The nodes were randomly deployed over simulation area.
- A sink node was also deployed randomly in the network.
- The area in which nodes were deployed was 500 X 800-unit area.
- The resynchronization time message was initially 30s.
- The range for delays was chosen with distribution over the range of 10 to 600.
- The number of nodes chosen was 21 for simulation purpose.
- The transmission range of the sensor nodes chosen was 100 units.

In the Figure 4.1, which is actually the snapshot of the network we model on our system for the purpose of simulation**,** show one root node and different node scattered along side of the root node. The sensor is randomly deployed in the area and the model will not be same as it is there in the real scenario. We also deployed the sensor in the random fashion and choose the model that best suits our requirement for the purpose of simulation of various time synchronization protocols. The wireless sensor network is one in which the sensors are not connected to each other but they send the message to each other sensors which are in the range of a given sensor. In our model, we use round circles to show the area of a specific sensor. The particular sensor can only send the message to those sensors, which come under the area of this circle. The lines connected to sensors are only

to show that they can communicate, as in real scenario they are wireless and no such line connecting sensor exists.

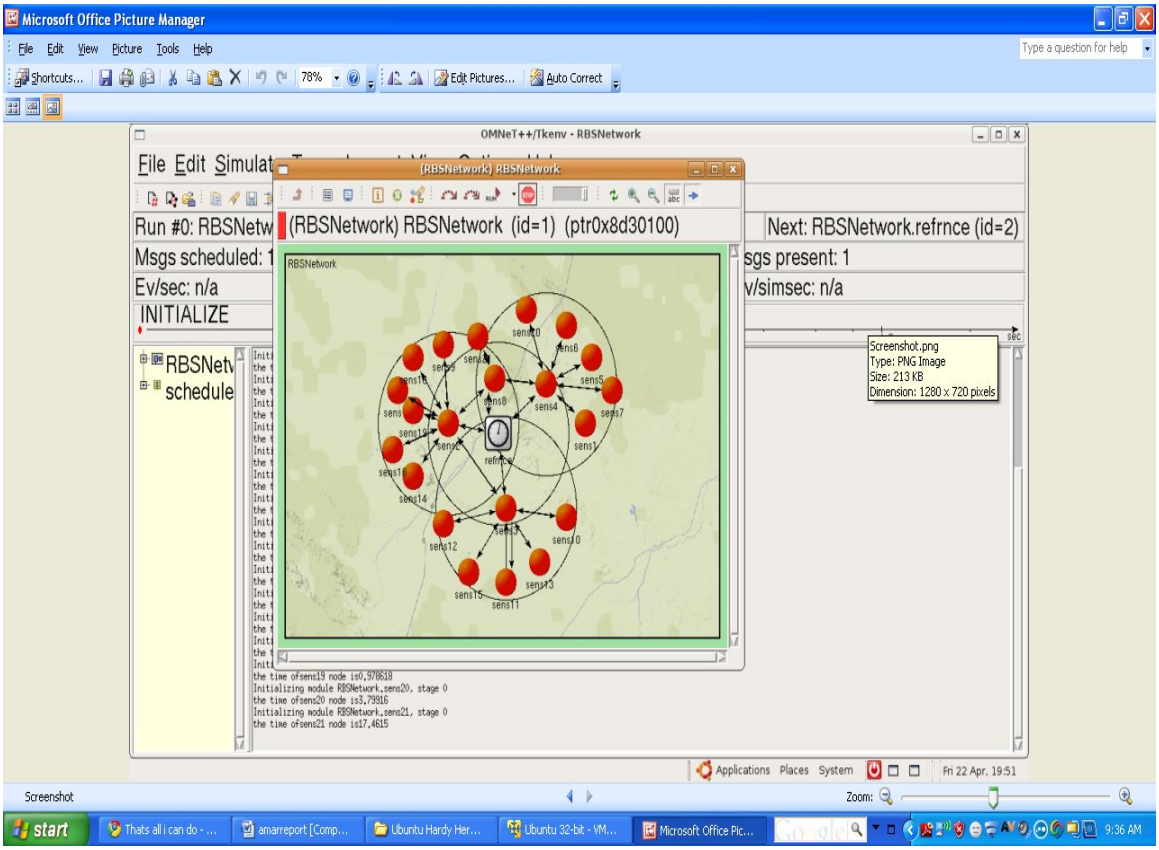

Figure 4.1: The Network model

4.3 Sensor node model

The sensor node model is common for all the sensors in the network. The energy of each sensor is set to 1000 and the message received is set to 0. The OMNET++ simulator allows writing the working of these modules in C++. There are two main functioned maintained by the simulator, we can add other functions according to our need also. First function is initialize, which is used to initialize the variable before just the simulation starts; the other function is handle message. The handle message function is the important one, which will work whenever the module got the message. In our scenario whenever the sensor node gets a message the handle message function will take

the charge. The handle message function can delete the message, create new message and do anything that the C++ language allowed to do. In our project whenever the message is received, we decrease the energy and increase the message received counter by one. The contents of the message and the kind of the message are checked, accordingly new message is created and send to the other nodes in the range.

To implement the concept of Mac layer time stamping in sensor node for FTSP we add few more features to our sensor node. The concept of layer is introduced in the coding. The Mac layer receives the message and sends it to the layer above it. The layer above Mac layer is specifically designed to add delay to the message before handling it to application layer. Application layer is responsible for reading and creating the messages. Application layer again send the message to the lower layer, which further add delay and send it to Mac layer. The Mac layer time stamp the message and broadcast the message. Other features of this sensor node are as common to the other sensor nodes, used to simulate other protocols.

4.4 Root node model

In wireless sensor network**,** the nodes are of same type and one of the nodes will act as the root node. Root node is one that provide interface to the outer world and all other nodes communicate their finding to the root node. For the purpose of simplicity, we made a different simple module called root module, which has some additional characteristics to the other sensor nodes. The working of the root node is different from the simple sensor node. The initialize method and handle message are the two functions that are in root node. We use the initialize message to send the first message in the whole network. This initialize will send a message to itself which is further modified and send to the common sensor nodes and finally to the other sensors as per requirement of the protocols. The main functionality of the root node is to put the time in the message and forward it to the other nodes. The time is initializing randomly on all the nodes in the network. In TPSN**,** our root node is responsible for initiating the level discovery phase as well as the time synchronization messages. The time of the nodes increased as the simulation time goes on. We keep on adding simulation time to the clocks to keep them ticking and working. The root node sends the time in the message with the help of a build in function called Set Timestamp. The root node also gets the messages when any common node broadcast it. The decision of ignoring the message is taken as per specification of the protocol.

4.5 Functions used

Various functions and data structures used for the simulation of NTP, RBS, TPSN and FTSP are discussed as follows:-

Functions used

a. ***Initialization ():***

It initializes the simple modules of the setup and called very first time when all the setup was initialized like constructor in C++. This is one of the in build function of C++ and used to initialize the values and parameters in the network. We use this function for setting initial time on the clocks of all sensors. The initialize function will run for all the sensor nodes and set the time for the node. The time is obtained with the help of random functions**,** which generate random value of time for each node. Due to random function, each node has different time and therefore, required to be synchronized.

b. ***Handle Message (CMessage *Msg)***

This function is called when any node gets a message from other node and to schedule a message to itself. The message can be either self-message or other module message. For the first time when we need to start the simulation**,** a self-message is sent. This self-message is caught by this handle Message function. We check the type of message if it is a self-message than we do some start work needed for the protocol. Otherwise, some protocol specific task is performed**.** The handle Message function takes an instance of message as a parameter.

c. ***Int rand ()***

This function is used to obtain a random value that is used for the time setting of the clock. The function returns an integer value.

d. ***Double rand ()***

Double rand function is similar to int rand function except it returns a double random value instead of an integer value. This value is added to the integer value to obtain a different time of sensor clock.

e. ***Sim Time ()***

This function is an important and most commonly used in our project work. The sim time represents simulation time. The timing of clocks **is** incremented according to the simulation time. Simulation time of OMNET++ starts with 0 and keeps on increasing. We add the simulation time to the clock of our sensor node to keep their clocks moving. The availability of these types of functions in OMNET++ allows us to use these functions rather than developing our own.

f. ***Set Timestamp ()***

This function is used to set the time stamp at that particular time to the message. Sensor networks are distributed in nature and demands the communication only through the exchange of messages. To put the time stamp in message this function is useful.

g. ***Set kind ()***

The set kind function is used to set the kind of message. Different types of messages are sent in different type of protocols. As in TPSN, there are two types of messages that are used, one for lever discovery and other for synchronization. The set kind method is used to differentiate these different types of messages.

h. ***Dup ()***

The dup function represents duplicate. When we need to send a same message to different nodes in the sensors area**,** we use this method. This method allows us to send the same message to different nodes.

i. ***GetArrival gate ( )***

GetArrival gate function is used to get the port number of the sensor at which the message arrives. Sometimes we need to reply to the same port from where the message arrives. We use this method for the purpose.

j. ***Poisson ( )***

OMNET++ allows us to use any of the distribution available for simulation. A**s** in build function is available that produce the same result as we got when we apply Poisson distribution in mathematics.

k. ***Send Delayed ( )***

This is also the mostly used function to send a message adding some delays. Delays are those that occur on the sensor side and are deterministic. We add the value of those delays and send the message by delaying that value.

l. ***Get kind ()***

Get kind is another function that is used in contrast to set kind. It returns the kind of the message. Whatever the kind of message we set, we can get the type using get kind function.

m. ***Get name ()***

This function is called when we need to find the name of the node. The code that the sensor node has is common to all sensors. At a particular moment**,** if we need to start work from a particular node, than we can call it by name. Get name helps us to do so.

n. ***Get Parent ()***

It returns the parent of the node.

o. ***Delay ()***

This function is used to put the delays in the network. In this method, we gave some value in integer and that value is used to put that delay in the working of

a sensor. It is mainly used in the program where one layer is sending message to other layer adding some delay.

4.6 Parameters

The following parameters are assumed for simulating various protocols. The parameters remain same for all the protocols. The simulation is tested for the resynchronization time of 30s, but it is a user specific parameter can be changed to other value also.

Parameter	**Value**
Maximum simulation time	1000
Non-deterministic delay range (micro sec.)	10-600
Transmission range (units)	150
Initial energy of the node (joule)	1000
Transmission Energy consumed (milli joules)	10
Receiving Energy consumed (milli joules)	5
Resynchronization time	30
Initial message at each node	0

Table 1 Parameters for simulation

Chapter 5

Results

In this chapter**,** we describe the results that we got after simulating the protocols on simulator. The results are collected and shown using line chart for clarity. The line chart we show contain four different lines of different colors. Each line represents the specific protocol and its performance on the parameter used for drawing that graph. The different results are shown as explained below.

5.1 Result 1: Number of Messages vs. Resynchronization Time

Figure 5.1 shows the comparison of FTSP, NTP, TPSN and RBS Protocols. The graph is drawn between the resynchronization times vs. the number of messages. The vertical axis shows the number of messages that keep on increasing with resynchronization time. Horizontal axis represents the resynchronization time. We choose the time interval of 30 seconds for simulation and we observe the data up to 150 seconds for each protocol. The total of the messages that are created for a particular protocol at a particular moment when the synchronization ends is collected and shown in the graph. The simulation is run for different value of resynchronization time of nodes. The NTP line is far above than all other nodes, this is due to the reason that NTP generate more messages as compared to other protocols. The FTSP generate the lowest number of messages according to our results. The TPSN provide more messages in our simulation results, because TPSN provide level discovery phase and that phase also need to send messages, which are further added to the messages that are sent for the purpose of time synchronization. RBS produce more messages as compared to FTSP but lower than TPSN, as RBS need to exchange messages between two neighbors for comparing their clock timing for the purpose of time synchronization.

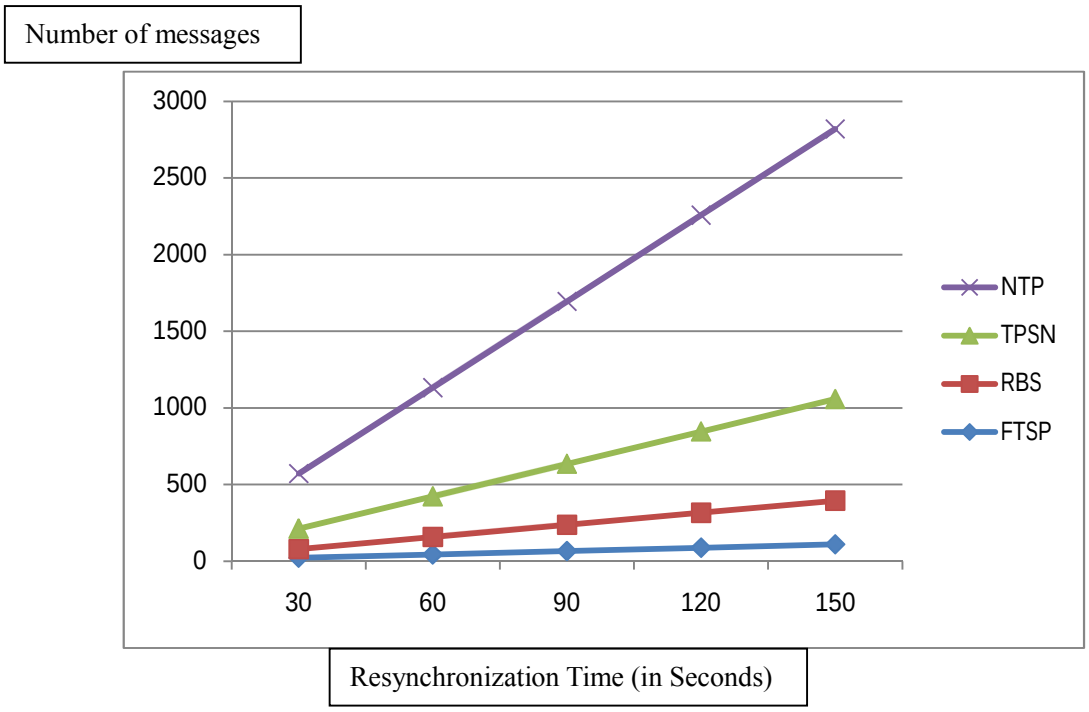

Figure 5.1 Number of messages vs. Resynchronization time

5.2 Result 2: Synchronization error vs. Resynchronization time

This line graph is used to show the performance of the time synchronization protocols for the parameter of accuracy. We draw the graph in which we show that that error decrease. As the error decreases between the two clocks, more accurate they become. The graph shows that the error in seconds for different protocols. The line of all the protocols go in a same direction, but the RBS line is coming downwards. This represents that the accuracy of the RBS increases with the increase of resynchronization period. The FTSP here maintain their clock at lowest error therefore**,** the FTSP shows better results as compare to other protocols. The network design in our project is just 2 hop network. Due to small number of hops in our simulation**,** the accuracy of FTSP is not affected much. The accuracy of FTSP decreases with the increase in hops in the multi hop sensor network [15]. FTSP provides a good accuracy in our model. The accuracy of NTP is better than both TPSN and RBS, but as the resynchronization time increase**,** the RBS

outperforms NTP. This is because more the resynchronization time occurs more is the time exchanges happen between two nodes. More the time exchanges are there more accurate the clock of sensors will be there.

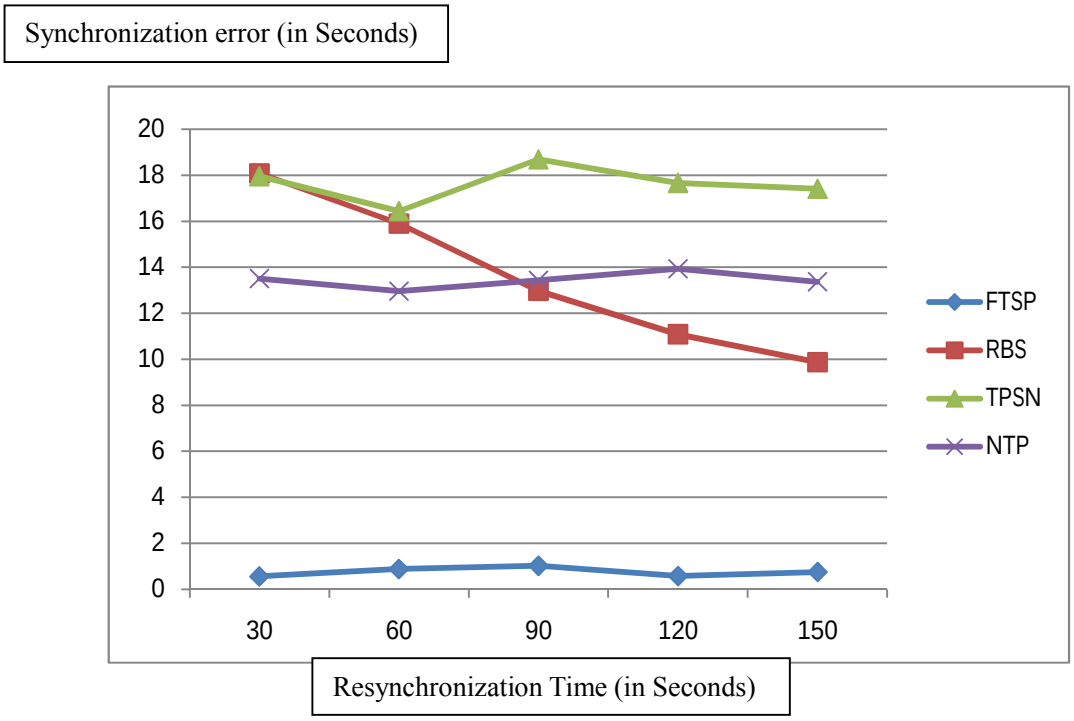

Figure 5.2 Synchronization Error in time vs. Resynchronization time

5.3 Result 3: Average energy remaining vs. Resynchronization time

This line graph is used to show the energy consumption of different protocols. We assume an energy decrement of 10 millijoules for a message sending and 5 millijoules for message receiving. As a message receives on a node**,** we decrease the energy again as soon as it sends the message we again decrease the energy of a node. The graph plotted here shows the average energy consumption of the network. This means that we are considering about the total energy of the network rather than for a single node. As the number of messages increases in the network with the increase in resynchronization time, similarly the consumption of energy also increases in the network. The average energy consumed of node at with different numbers is shown in figure 5.3.

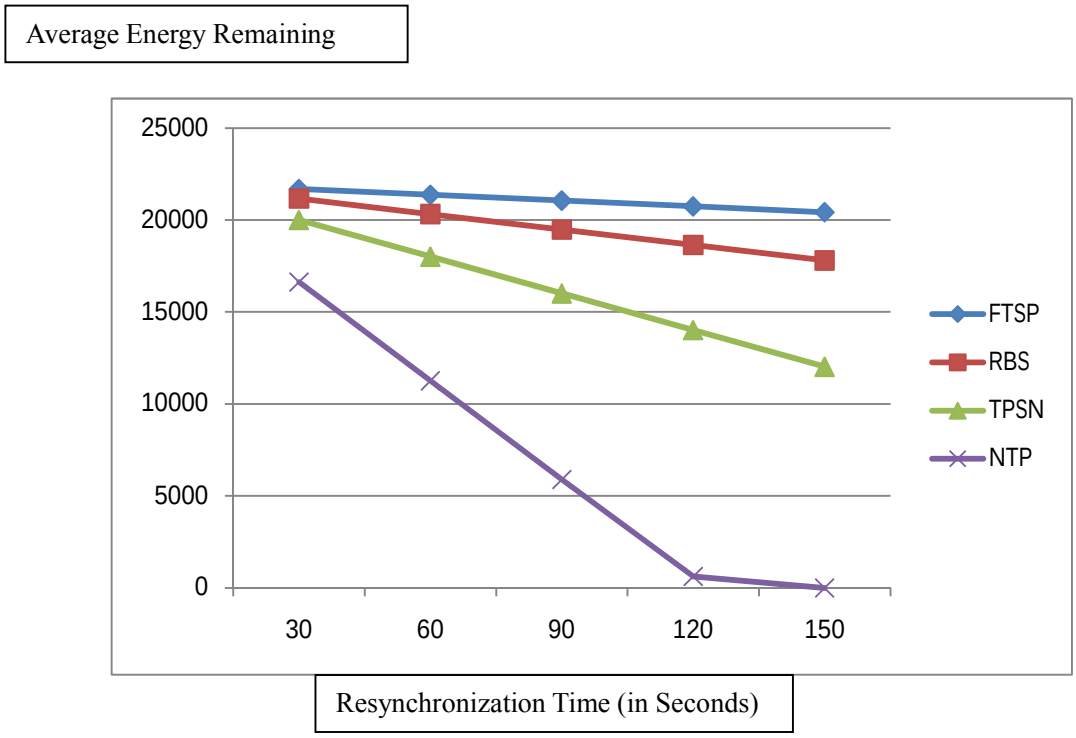

Figure 5.3 Average energy Remaining vs. Resynchronization time

The Figure shows that NTP make the system exhaust very fast as compare to other protocols. FTSP is the least energy consuming protocol according to our simulation results. As only one message is flooded to each node and no other step is followed to make it more effective, which reduce the number of messages and hence decrease the energy consumptions of the protocol. RBS due to its time exchange consume more energy as compared to FTSP and TPSN due to its synchronization level needs to send more messages and therefore, consumes more energy as compared to FTSP and RBS.

Chapter 6

Conclusion and Future Scope

6.1 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 application of distributed system. The comparison of various protocols and their results bring out the facts, which can be used before deploying any network. NTP, which is famous protocol for network, proves its performance in sensor network in term of accuracy but it, lack in term of energy consumption**.** It consumes 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. Therefore, the initial time accuracy is not provided by this protocol and hence avoided in circumstances here initial accuracy is desired. FTSP perform very well in terms of energy and accuracy, but as discussed in previous chapter, the accuracy decreases with increase in multiple hops. Therefore**,** it is not fit for multi hop networks but provide a good accuracy with limited hop of network.

6.2 Future work

Based on the result we can device a new Hybrid type of method that can take the features of these protocols. The ideas presented here could be fully or partially applied to improve the performance of existing protocols or for designing new protocols. Experimental performance evaluation and comparisons with other existing protocols represent an open research problem for future network**.**

References

[1] F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. “Wireless Sensor Networks: A Survey." Computer Networks,384:393, March.

[2] D.L. Mills, “Internet time synchronization: The Network Time Protocol” In Z. Yang and T.A. Marsland, editors, Global States and Time in distributed Systems. IEEE Computer Society Press, 1994.

[3] Miklos Maroti , Branislav Kusy , Gyula Simon , Akos Ledeczi, “The flooding time synchronization protocol”, Proceedings of the 2nd international conference on Embedded networked sensor systems, November 03-05, 2004, Baltimore, MD, USA

[4] Elson, J. E., Girod, L., and Estrin, D. “Fine-Grained Network Time Synchronization using Reference Broadcasts”. The Fifth Symposium on Operating Systems Design and Implementation (OSDI), pp. 147–163, December 2002.

[5] S. Ganeriwal, R. Kumar, and M. B. Srivastava, “Timing-sync protocol for sensor networks,” in Proceedings of the 1st Int’l Conf. on Embedded Networked Sensor Systems. ACM Press, 2003, pp. 138-149.

[6] Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, MichaelHamilton, and Jerry Zhao. “ Habitat Monitoring: Application Driver for Wireless Communications Technology." In Proceedings of the2001 ACM SIGCOMM Workshop on Data Communications in LatinAmerica and the Caribbean, April 2001.

[7] A.A. Berlin, J.G. Chase, M.H. Yim, B.J. Maclean, M. Oliver, and S.C. Jacobsen. “MEMS-Based Control of Structural Dynamic Instability." Journal of Intelligent Material Systems and Structures, 9(7):574-586, 1998.

[8] DARPA Advanced Technology Office (ATO). “Self-Healing Minefield." http://www.darpa.mil/ato/programs/SHM/.

[9] C.R. Baker, K. Armijo, S. Belka, M. Benhabib, V. Bhargava, N. Burkhart, A.D. Minassians, G. Dervisoglu, L. Gutnik, M.B. Haick, C. Ho, M. Koplow, J. Mangold, S. Robinson, M. Rosa, M. Schwartz, C. Sims, H. Stoffregen, A. Waterbury, E.S. Leland, T. Pering, P.K. Wright, Wireless sensor networks for home health care, in: AINAW, Ontario, Canada, 2007.

[10] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, J. Lees, Deploying a wireless sensor network on an active volcano, IEEE Internet Computing 10 (2006) 18–25.

[11] Tanmay Ratnaparkhe, Sagar Natekar, Sridhar Chandan , Vaishali P.Sadaphal. “Selection of Time Synchronizing Nodes in Wireless Sensor Network”

[12] Khurram Shahzad, Arshad Ali, N.D Gohar. “ETSP: An Energy-efficient Time Synchronization Protocol for Wireless Sensor Networks”, International Conference on Advanced Information Networking and Applicatoins.

[13] Xiaoya Hu, Bingwen Wang. “A Novel Energy-Balanced Time Synchronization protocol in Wireless Sensor Networks for Bridge Structure Health Monitoring

[14] Youngtae jo, Chongmyung park, Joahyoung Lee, Inbum Jung. “Energy Effective Time Synchronization inWireless Sensor Network”. Internationa conference on Computational Science and Applications.

[15] Hyojung Lee, Wonpil Yu and Youngmi Kwon. “Efficient RBS in sensor network”. Third international conference on Information Technology: New Generations(

ITNG'06).

[16] J. Elson, D. Estrin, "Time Synchronization for Wireless Sensor Networks," Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Parallel and Distributed Computing Issues in Wireless and Mobile Computing, San Francisco, California, USA, April 2001.

APPENDIX

FTSP. MAC .CC

```
#include "Mac.h"
Define_Module(Mac);
void Mac::initialize()
{
energy=1000;
messRec=0;
}
void Mac::handleMessage(cMessage *msg)
{
energy=energy-5;
messRec++;
ev<<"Energy is"<<energy<<"message received"<<messRec;
if(msg->getArrivalGate()==gate("muppPort$i"))
{
double getTime=getParentModule()->par("Times").doubleValue();
double tempdelay=getParentModule()->par("delay").doubleValue();
double temp=getTime+tempdelay;
msg->setTimestamp(temp);
ev<<" MAC Layer will Send the message with time stamp"<<temp;
if(strcmp("sens4", getParentModule()->getName()) == 0)
{
for(int i=1;i<6; i++)
send(msg->dup(), "mlowPort$o",i);
}
if(strcmp(getParentModule()->getName(),"sens8")==0)
{
send(msg,"mlowPort$o",1);
}
if(strcmp("sens2", getParentModule()->getName()) == 0)
{
for(int i=1;i<7; i++)
send(msg->dup(), "mlowPort$o",i);
}
if(strcmp("sens3", getParentModule()->getName()) == 0)
{
for(int i=1;i<6; i++)
send(msg->dup(), "mlowPort$o",i);
}
}
else
{
double temptime=getParentModule()->par("Times").doubleValue();
```

```
temptime=temptime+simTime().dbl();
getParentModule()->par("Arr_mesg").setDoubleValue(temptime);
ev<<"mac layer note the message with time"<<temptime;
send(msg,"muppPort$o");
}
}
```

FTSP.Delay.cc

```
#include "Delay.h"
Define_Module(Delay);
void Delay::initialize()
{
}
void Delay::handleMessage(cMessage *msg)
{
if(msg->getArrivalGate()==gate("duppPort$i"))
{
double send=poisson(250);
double delay1=send/1000;
getParentModule()->par("delay").setDoubleValue(delay1);
ev<<"The delay module added the delay"<<delay1<<endl;
sendDelayed(msg,delay1,"dlowPort$o");
}
else
{
double receive=poisson(250);
double delay1=receive/1000;
getParentModule()->par("delay").setDoubleValue(delay1);
ev<<"The delay module added the delay"<<delay1<<endl;
sendDelayed(msg,delay1,"duppPort$o");
}
}
```

FTSP.application.cc

```
#include "Application.h"
Define_Module(Application);
void Application::initialize()
{
double temp=intrand(24)+dblrand();
getParentModule() ->par("randm").setDoubleValue(temp);
getParentModule() ->par("Times").setDoubleValue(temp+simTime().dbl()) ;
ev<<"the time of"<<getParentModule()->getName()<<" node is"<<getParentModule()-
>par("Times").doubleValue()<<endl;
}
void Application::handleMessage(cMessage *msg)
{
```

```
if(msg->getKind()==11)
{
double rectime=msg->getTimestamp().dbl();
ev<<"i got time stamp"<<rectime<<endl;
rectime=rectime+getParentModule()->par("delay").doubleValue();
ev<<" i will set my time i.e time of"<<getParentModule()->getName()<<" as"<<rectime;
getParentModule()->par("Times").setDoubleValue(rectime);
msg->setKind(10);
sendDelayed(msg,1,"alowPort$o");
}
else
{
double rectime=msg->getTimestamp().dbl();
ev<<"i got time stamp"<<rectime<<endl;
rectime=rectime+getParentModule()->par("delay").doubleValue();
ev<<" i will set my time as"<<rectime;
getParentModule()->par("Times").setDoubleValue(rectime);
delete msg;
}
}
```

FTSP.Sink.cc

```
#include "Sink.h"
Define_Module(Sink);
void Sink::initialize()
{
energy =1000;
messRec=0;
randm=intrand(23) +  dblrand();
Times =randm+simTime().dbl();
ev<<"the time of"<<getName()<<" node is"<<Times<<endl;
scheduleAt(simTime(), new cMessage("Initialize"));
}
void Sink::handleMessage(cMessage *msg)
{
if(energy>0)
{
Times=Times+simTime().dbl();
int FTSPTime=getParentModule()->par("Resyn_time");
cMessage *FTSP = new cMessage("Flooding Time");
FTSP->setKind(11);
FTSP->setTimestamp(Times+1);
ev<<"The reference node time stamp the message and send it with"<<(Times+1);;
for(int i=0;i<gateSize("siport");i++)
sendDelayed(FTSP->dup(),1,"siport$o", i);
energy = energy -10;
delete msg;
```

```
scheduleAt(simTime()+ FTSPTime, new cMessage());
}
}
```

NTP.Server.cc

```
#include "Server.h"
Define_Module(Server);
void Server::initialize()
{
ev<<"Set the level of the Server as Zero"<<endl;
level=0;
double randm=intrand(23) +  dblrand();
Timee =randm;
ev<<"the time of server is"<<Timee<<endl;
scheduleAt(simTime(),new cMessage("Ready"));
energy=1000;
messRec=0;
}
void Server::handleMessage(cMessage *msg)
{
if(energy>=0)
{
if(msg->isSelfMessage())
{
msg->setKind(level);
double delay=dblrand();
for(int i=0; i<gateSize("seport"); i++)
sendDelayed(msg->dup(),delay,"seport$o",i);
energy=energy-10;
delete msg;
int NTPTime=getParentModule()->par("Resyn_time");
scheduleAt(simTime()+ NTPTime, new cMessage("Ready"));
}
else
{
energy=energy-5;
ev<<"energy remains"<<energy;
messRec++;
ev<<"Total message received"<<messRec<<endl;
ev<<"Receive request for time stamp message"<<endl;
double Arr=msg->getArrivalTime().dbl();
Arr=Timee+Arr;
ev<<"Server receive request for time stamp at "<<Arr<<endl;
cMessage *FinalTime =new cMessage("Time with stamps");
int Timest=Arr*1000;
FinalTime->setSrcProcId(Timest);
```

```
double delay=dblrand();
ev<<"Server make the another time stamp as "<<Timee+simTime().dbl()+delay<<endl;
FinalTime->setTimestamp(Timee+simTime().dbl()+delay);
FinalTime->setKind(Finalreply);
int portnumber=msg->getArrivalGate()->getIndex();
sendDelayed(FinalTime,delay,"seport$o",portnumber);
energy=energy-10;
ev<<"energy remains"<<energy<<endl;
}
}
}
```

NTP. Sensors.cc

```
#include "Sensors.h"
Define_Module(Sensors);
void Sensors::initialize()
{
energy=1000;
messRec=0;
level=100;
randm=intrand(23) +  dblrand();
Time =randm;
ev<<"the time of"<<getName()<<" node is"<<Time<<endl;
count=0;
}
void Sensors::handleMessage(cMessage *msg)
{
if(energy<=0)
{
cDisplayString dstr = getDisplayString();
dstr.setTagArg("i", 0, "old/ball");
setDisplayString(dstr);
}
if(energy>0)
{
int templevel=msg->getKind();
energy=energy-5;
messRec++;
ev<<"Energy remains "<<energy<<" and message received = "<<messRec<<endl;
if(templevel == 0 || templevel==1)
{
level=templevel+1;
ev<<"i got Ready message"<<endl;
int portnumb = msg->getArrivalGate()->getIndex();
double delay=dblrand();
```

```
cMessage * TimeReq = new cMessage("Time Request");
TimeReq->setKind(TReq);
Timestamp1=Time+simTime().dbl();
ev<<"Timestamp 1 is"<<Timestamp1<<endl;
sendDelayed(TimeReq->dup(),delay,"sport$o",portnumb);
energy=energy-10;
ev<<"Send Time request to Energy remains"<<energy;
delete msg;
delete TimeReq;
}
else if(msg->getKind()==Finalreply )
{
double a,b;
double offset, RTdelay;
if(count<4)
{
int tt=msg->getSrcProcId();
Timestamp2=tt;
Timestamp2=Timestamp2/1000;
Timestamp3=msg->getTimestamp().dbl();
Timestamp4=Time+simTime().dbl();
ev<<"Time stamp 1 "<<Timestamp1<<endl;
ev<<"Time stamp 2 "<<Timestamp2<<endl;
ev<<"Time stamp 3 "<<Timestamp3<<endl;
ev<<"Time stamp 4 "<<Timestamp4<<endl;
a=(Timestamp2-Timestamp1);
ev<<"The calculated value of a is "<<a<<endl;
b=(Timestamp3-Timestamp4);
ev<<"The calculated value of b is "<<b<<endl;
RTdelay=a-b;
offset = (a+b)/2;
Drift[count]=offset;
Delay[count]=RTdelay;
cMessage * TimeReq = new cMessage("Time Request");
TimeReq->setKind(TReq);
Timestamp1=Time+simTime().dbl();
int portnumb = msg->getArrivalGate()->getIndex();
double delay=dblrand();
sendDelayed(TimeReq->dup(),delay,"sport$o",portnumb);
energy=energy-10;
ev<<"Send Time request to Energy remains"<<energy;
count++;
delete TimeReq;
}
else
{
```

```
count=0;
double minDelay, minoffset, mintemp=100.0;
for(int z=0;z<4;z++)
{
if(mintemp>Delay[z])
{
mintemp=Delay[z];
}
}
ev<<"Finally the value of minimum delay  is"<<mintemp;
minDelay=mintemp;
int index;
for(int v=0;v<4;v++)
{
if(minDelay==Delay[v])
index=v;
}
minoffset=Drift[index];
Time = Time + simTime().dbl() + minDelay + minoffset;
ev<<"Final time setting is"<<Time<<"...."<<minDelay<<"..."<<minoffset;
cMessage *ready =new cMessage("Ready");
int portnumb = msg->getArrivalGate()->getIndex();
ready->setKind(level);
double delay=dblrand();
for(int i=0; i<gateSize("sport"); i++)
{
if(i!=portnumb)
sendDelayed(ready->dup(),delay,"sport$o",i);
}
energy=energy-10;
delete msg;
}
}
else if(msg->getKind()==TReq)
{
ev<<"Receive request for time stamp message"<<endl;
double Arr=msg->getArrivalTime().dbl();
Arr=Time+Arr;
ev<<"Sensor receive request for time stamp at "<<Arr<<endl;
cMessage *FinalTime =new cMessage("Time with stamps");
int Timest=Arr*1000;
FinalTime->setSrcProcId(Timest);
double delay=dblrand();
ev<<"Sensor make the another time stamp as <<Time+simTime().dbl()+delay<<endl;
FinalTime->setTimestamp(Time+simTime().dbl()+delay);
FinalTime->setKind(Finalreply);
```

```
int portnumber=msg->getArrivalGate()->getIndex();
sendDelayed(FinalTime,delay,"sport$o",portnumber);
energy=energy-10;
ev<<"energy remains"<<energy<<endl;
}
}
}
```

RBS.Sensors.cc

```
#include "Sensors.h"
Define_Module(Sensors);
void Sensors::initialize()
{
energy =1000;
messRec=0;
randm=intrand(23) +  dblrand();
Times =randm+simTime().dbl();
ev<<"the time of"<<getName()<<" node is"<<Times<<endl;
}
void Sensors::handleMessage(cMessage *msg)
{
if(energy<=0)
{
cDisplayString dstr = getDisplayString();
dstr.setTagArg("i", 0, "old/ball");
setDisplayString(dstr);
}
if(energy>0)
{
ev<<"Intial energy"<<energy<<endl;
energy = energy-5;
ev<<"Energy of"<<getName()<<"remains"<<energy<<endl;
if (msg->getKind() == REF)
{
int portnumb=msg->getArrivalGate()->getIndex();
messRec++;
ev<<"total message received by"<<getName()<<"are "<<messRec;
cMessage *Texch = new cMessage("TIME");
Texch->setKind(11);
Texch->setTimestamp(Times);
for(int i=0;i<gateSize("sport");i++
{
if(i!=portnumb)
sendDelayed(Texch->dup(),1,"sport$o", i);
}
```

```
energy = energy-10;
ev<<endl<<getName()<<"sends a message, Energy now is"<<energy;
delete msg;
delete Texch;
}
else if(msg->getKind() == 11)
{
simtime_t Rtime=msg->getTimestamp();
ev<<"i got time"<<Rtime;
Times= Times+simTime().dbl();
ev<<"MY TIME IS"<<Times;
Times=(Times+Rtime.dbl())/2;
ev<<"MY TIME IS"<<Times;
int portnumb = msg->getArrivalGate()->getIndex();
cMessage *ack = new cMessage("ACK");
sendDelayed(ack,1,"sport$o",portnumb);
delete msg;
energy = energy-10;
ev<<endl<<getName()<<"sends a message, Energy now is"<<energy<<endl;
messRec++;
ev<<"total message received by"<<getName()<<"are "<<messRec;
}
else
delete msg;
}
}
```

RBS. ReferenceNode.cc

```
#include "RefNode.h"
Define_Module(RefNode);
void RefNode::initialize()
{
ev<<"This is initialization of Reference Node"<<endl;
scheduleAt(simTime(),new cMessage("INITIALIZE"));
energy = 1000;
}
void RefNode::handleMessage(cMessage *msg)
{
if(energy>=0)
{
if(msg->isSelfMessage())
{
int RBSTime=getParentModule()->par("Resyn_time");
cMessage *RBS = new cMessage("Reference Message");
RBS->setKind(REF);
for(int i=0;i<gateSize("rport");i++)
```

```
sendDelayed(RBS->dup(),1,"rport$o", i);
energy = energy -10;
delete msg;
scheduleAt(simTime()+ RBSTime, new cMessage());
}
}
}
```

TPSN. Root.cc

```
#include "Root.h"
Define_Module(Root);
void Root::initialize(int stage)
{
if (stage ==0)
{
ev<<"Set the level of the root as Zero"<<endl;
level=0;
scheduleAt(simTime(), new cMessage("Level Discovery"));
}
else if (stage==1)
{
ev<<"Initializing time in this stage"<<endl;
energy =1000;
messRec=0;
randm=intrand(23) +  dblrand();
Timee=randm;
ev<<"the time of"<<getName()<<" node is"<<Timee<<endl;
}
}
void Root::handleMessage(cMessage *msg)
{
if(msg->isSelfMessage())
{
msg->setKind(level);
double delay=dblrand();
for(int i=0; i<gateSize("rport"); i++)
sendDelayed(msg->dup(),delay,"rport$o",i);
energy=energy-10;
delete msg;
int TPSNTime=getParentModule()->par("Resyn_time");
scheduleAt(simTime()+ TPSNTime, new cMessage("LEvel Discovery"));
}
else if(msg->getKind()==1)
{
energy=energy-5;
ev<<"energy remains"<<energy<<endl;
```

```
messRec++;
ev<<"Total message received"<<messRec<<endl;
int temp=msg->getKind();
ev<<"Root got message with level ...."<<temp<<" ignoring the message"<<endl;
double mes=messRec%4;
if(mes==0)
{
ev<<"Ready to send READY PACKET"<<endl;
cMessage *Ready =new cMessage("Ready Packet");
Ready->setKind(RP);
for(int i=0;i<gateSize("rport"); i++)
sendDelayed(Ready->dup(),1,"rport$o",i);
energy=energy-10;
ev<<"energy remains"<<energy<<endl;
}
}
else if(msg->getKind()==TReq)
{
energy=energy-5;
ev<<"energy remains"<<energy<<endl;
messRec++;
ev<<"Total message received"<<messRec<<endl;
ev<<"Receive request for time stamp message"<<endl;
double Arr=msg->getArrivalTime().dbl();
Arr=Timee+Arr;
ev<<"Root receive request for time stamp at "<<Arr<<endl;
cMessage *FinalTime =new cMessage("Time with stamps");
int Timest=Arr*1000;
FinalTime->setSrcProcId(Timest);
double delay=dblrand();
ev<<"Root make the another time stamp as "<<Timee+simTime().dbl()+delay<<endl;
FinalTime->setTimestamp(Timee+simTime().dbl()+delay);
FinalTime->setKind(Finalreply);
int portnumber=msg->getArrivalGate()->getIndex();
sendDelayed(FinalTime,delay,"rport$o",portnumber);
energy=energy-10;
ev<<"energy remains"<<energy<<endl;
}
}
```

TPSN.Sensors.cc

```
#include "Sensor.h"
Define_Module(Sensor);
void Sensor::initialize(int stage)
{
if(stage==0)
{
```

```
ev<<"Initializing level in this stage"<<endl;
level=100;
}
if(stage==1)
{
ev<<"Initializing time in this stage"<<endl;energy =1000;
messRec=0;
randm=intrand(23) +  dblrand();
Time =randm;
ev<<"the time of"<<getName()<<" node is"<<Time<<endl;
}
}
void Sensor::handleMessage(cMessage *msg)
{
int templevel=msg->getKind();
energy=energy-5;
messRec++;
ev<<"Energy remains "<<energy<<" and message received = "<<messRec<<endl;
if(templevel == 0 )
{
ev<<"i got message from level  "<<templevel<<endl;
level=templevel+1;
ev<<"i set my level as  "<<level<<endl;
msg->setKind(level);
double delay=dblrand();
for(int i=0; i<gateSize("sport"); i++)
sendDelayed(msg->dup(),delay,"sport$o",i);
energy=energy-10;
ev<<"send message energy now is"<<energy;
delete msg;
}
else if(templevel == 1 )
{
ev<<"i got message from level  "<<templevel<<endl;
if(level>1)
{
level=templevel+1;
ev<<"i set my level as  "<<level<<endl;
}
else
{
ev<<"my level is "<<level<< "..... ignoring the message";
}
delete msg;
}
else if(msg->getKind()==RP || msg->getKind()==TSS)
```

```
{
ev<<"i got Time Ready message";
int portnumb = msg->getArrivalGate()->getIndex();
ev<<"Waiting for random time to request for time stamp "<<endl;
double delay=dblrand();
cMessage * TimeReq = new cMessage("Time Request");
TimeReq->setKind(TReq);
Timestamp1=Time+simTime().dbl();
ev<<"Timestamp 1 is"<<Timestamp1;
sendDelayed(TimeReq->dup(),delay,"sport$o",portnumb);
energy=energy-10;
ev<<"Send Time request to Energy remains"<<energy;
delete msg;
delete TimeReq;
}
else if(msg->getKind()==Finalreply)
{
int tt=msg->getSrcProcId();
Timestamp2=tt;
Timestamp2=Timestamp2/1000;
Timestamp3=msg->getTimestamp().dbl();
Timestamp4=Time+simTime().dbl();
ev<<"Time stamp 1 "<<Timestamp1<<endl;
ev<<"Time stamp 2 "<<Timestamp2<<endl;
ev<<"Time stamp 3 "<<Timestamp3<<endl;
ev<<"Time stamp 4 "<<Timestamp4<<endl;
double drift=((Timestamp2-Timestamp1)-(Timestamp4-Timestamp3))/2;
ev<<"The calculated drift is "<<drift<<endl;
double temptime=Time+simTime().dbl()+drift;
double getdelay=((Timestamp2-Timestamp1)+(Timestamp4-Timestamp3))/2;
ev<<"The calculated delay is "<<getdelay<<endl;
Time=Time + drift+ getdelay;
ev<<"Therefore Final time set is "<<Time+simTime().dbl()<<endl;
ev<<"Forwading Time Ready message"<<endl;
energy=energy-10;
ev<<"send message energy now is"<<energy<<endl;
int portnumb = msg->getArrivalGate()->getIndex();
cMessage *Ready =new cMessage("Ready Packet");
Ready->setKind(TSS);
for(int i=0;i<gateSize("sport"); i++)
{
if(i!=portnumb)
send(Ready->dup(),"sport$o",i);
}
}
else if(msg->getKind()==TReq)
```

```
{
ev<<"Receive request for time stamp message"<<endl;
double Arr=msg->getArrivalTime().dbl();
Arr=Time+Arr;
ev<<"Sensor receive request for time stamp at "<<Arr<<endl;
cMessage *FinalTime =new cMessage("Time with stamps");
int Timest=Arr*10000;
FinalTime->setSrcProcId(Timest);
double delay=dblrand();
ev<<"Sensor make the another time stamp as "<<Time+simTime().dbl()+delay<<endl;
FinalTime->setTimestamp(Time+simTime().dbl()+delay);
FinalTime->setKind(Finalreply);
int portnumber=msg->getArrivalGate()->getIndex();
sendDelayed(FinalTime,delay,"sport$o",portnumber);
energy=energy-10;
ev<<"energy remains"<<energy<<endl;
}
}
```