2005

Sensorsimulator: simulation framework for sensor networks

Cariappa D. Mallanda
Louisiana State University and Agricultural and Mechanical College

SENSORSIMULATOR: SIMULATION FRAMEWORK FOR SENSOR NETWORKS

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in System Science

in

The Department of Computer Science

By

Cariappa D. Mallanda
B.E. Computer Science & Engineering, Kuvempu University, 2000
May, 2005

Acknowledgments

I would like to express my sincere gratitude to Dr. S. Sitharama Iyengar, my advisor, for his invaluable guidance and encouragement extended throughout the study. His tenacious supervision, helpful suggestion, patience and time deserve a special mention

I would like to express my appreciation to my committee members Dr. Rajgopal Kannan and Dr. Arjan Durresi for their support and suggestions.

I would also like to thank my colleagues Ankur Suri,Vatsalya Kunchakarra, Neelay Shah, Vamsi K Paruchuri, Ramaraju Kalidindi, Sasanka Madiraju for their suggestions during progress of this thesis.

I thank my parents Mr. M. P. Devaiah and Mrs. Rani Devaiah and my brother Mr. Belliappa M. D., for their continued support and encouragement in the course of doing my Masters degree

Last, but not the least, I would like to gratefully acknowledge Department of Computer Science, Louisiana State University for providing the resources and needs during the thesis.

Table of Contents

Acknowledgments ii

List of Figures iv

Abstract v

Chapter 1: Introduction 1

Chapter 2: Currently Available Simulators 5

Chapter 3: OMNeT++ Simulation Environment 8

Chapter 4: Overview of the Proposed Simulation Framework 11

Chapter 5: Design Approach 16
- 5.1 Target Node 16
- 5.2 Sensor Channel 16
- 5.3 Sensor Node 18
  - 5.3.1 Coordinator Module 18
  - 5.3.2 Hardware Modeling 19
  - 5.3.3 Software Model 20
- 5.4 Wireless Channel 21

Chapter 6: Demonstrative Use of the Simulation Framework 23
- 6.1 Geographic Routing (GEAR) 23
- 6.2 Directed Diffusion 24
- 6.3 Implementation Details 25
  - 6.3.1 Directed Diffusion with GEAR implementation 26
  - 6.3.2 MAC 802.11 27
- 6.4 Experimental Results 30
  - 6.4.1 Validating Directed Diffusion Implementation 30
  - 6.4.2 Directed Diffusion-GEAR with SimpleMAC 31
  - 6.4.3 Directed Diffusion-GEAR with IEEE 802.11 MAC 36

Chapter 7: Conclusion and Future Works 39

References 40

Vita 43

List of Figures

Figure 3.1: Simple and Compound modules in OMNeT++ 9

Figure 4.1: Sensor Node Representation in a Network 11

Figure: 4.2 Abstract view of a Sensor Network 14

Figure 5.1: Sequence of activities carried out by the Coordinator in the simulation 19

Figure 6.1 Implementation and flow diagram of Directed Diffusion-GEAR with MAC 802.11 Simulation 25

Figure 6.2 Structure of a Query Message 26

Figure 6.3 Parameters for 802-11 Simulations 30

Figure 6.4 Message delivery ratio SensorSimulator Vs NS-2 31

Figure 6.5 Execution time for 10 Queries for 150 simulation seconds 33

Figure 6.6 Execution time for 100 Queries for 150 simulation seconds 33

Figure 6.7 Memory Utilized to setup the network for 10 Queries 34

Figure 6.8 Memory Utilization during simulation for 10 Queries 35

Figure 6.9 Memory Utilized to setup network for 100 queries 35

Figure 6.10 Memory utilization comparison during simulation for 100 Queries 36

Figure 6.11 Directed Diffusion-GEAR-MAC802.11 execution Time for 10 queries simulated for 300 simulation seconds 37

Figure 6.12 Directed Diffusion-GEAR-MAC802.11 memory usage for 10 queries simulated for 300 simulation seconds 38

Abstract

Wireless sensor networks have the potential to become significant subsystems of engineering applications. Before relegating important and safety-critical tasks to such subsystems, it is necessary to understand the dynamic behavior of these subsystems in simulation environments. There is an urgent need to develop a simulation platform that is useful to explore both the networking issues and the distributed computing aspects of wireless sensor networks. Current approaches to simulating wireless sensor networks largely focus on the networking issues. These approaches use well-known network simulation tools that are often difficult to extend to explore distributed computing issues. Discrete-event simulation is a trusted platform for modeling and simulation of a variety of systems. SensorSimulator is a discreet event simulation framework for sensor networks built over OMNeT++. It is a customizable and an extensible framework for wireless sensor network simulation. This framework allows the user to debug and test software for distributed sensor networks independent of hardware constraints. The extensibility of SensorSimulator allows developers and researchers to investigate topological, phenomenological, networking, robustness and scaling issues, to explore arbitrary algorithms for distributed sensors, and to defeat those algorithms through simulated failure. The framework provides modules for various layers. Applications can be implemented by using these framework modules by sub classing the framework classes and customizing their behavior at various network layers.

We validate and demonstrate the usability of these capabilities through analyzing the simulation results of Directed Diffusion and GEAR. A comparison study of performance of SensorSimulator v/s NS2 for various network densities and traffic have

shown that SensorSimulator is able to achieve higher scalability and requires less time for execution.

Chapter 1: Introduction

Wireless Sensor Networks (WSN)[2] consists of numerous tiny sensors deployed at high density in regions requiring surveillance and monitoring. These sensors can be deployed at a cost much lower than the traditional wired sensor system. A typical sensor node consists of one or more sensing elements (motion, temperature, pressure, etc.), a battery, and low power radio trans-receiver, microprocessor and limited memory. An important aspect of such networks is that the nodes are unattended, have limited energy and the network topology is unknown. Many design challenges that arise in sensor networks are due to the limited resources they have and their deployment in hostile environments.

Sensor nodes are deployed in environments where it is impractical or infeasible for humans to interact or monitor them. Some applications require sensors to be small in size and have short transmission ranges to reduce the chances of detection. These size constraints cause further constraints on CPU speed, amount of memory, RF bandwidth and battery lifetime. Hence, efficient communication techniques are essential for increasing the lifetime and quality of data collection and decreasing the communication latency of such wireless devices.

Unlike the mobile ad hoc networks, sensor nodes are most likely to be stationary for the entire period of their lifetime. Even though the sensor nodes are fixed, the topology of the network can change. During periods of low activity, nodes may go to inactive sleep state, to conserve energy. When some nodes run out of battery power and die, new nodes may be added to the network. Although all nodes are initially equipped with equal energy, some nodes may experience higher activity as result of

region they are located in. Communication pattern is intermittent and sensor applications are data-centric in nature. An important property of sensor networks is the need of the sensors to reliably disseminate the data to the sink or the base station within a time interval that allows the user or controller application to respond to the information in a timely manner, as out of date information is of no use and may lead to disastrous results.

Another important attribute is the scalability to the change in network size, node density and topology. Sensor networks are very dense as compared to mobile ad hoc and wired networks. This arises from the fact that the sensing range is lesser than the communication range and hence more nodes are needed to achieve sufficient sensing coverage. Sensor nodes are required to be resistant to failures and attacks.

Typical Sensor Network applications [22] can be classified based on its functionality as follows.

- Continuous Sensing: This includes all applications in which sensors continuously sense their surroundings for certain parameters (temperature, pressure, etc) and transmit this data to the sink. The information sent to the sink may be aggregated. As the result of continuous sensing and communication, the node energies deplete at a rapid rate.
- Event Based Sensing: In this class of applications the node senses its environment for certain parameters. If the parameter value exceeds a certain threshold, an event is triggered and a report is sent to the base station. In this case the energy depletion occurs at a slower rate when compared to Continuous Sensing.

- Query Based Sensing: The base station requiring information from designated region of the network, forwards a query to that region. On receiving the request, the sensors in that region transmit the required parameters to the base station. The energy depletion rate of the sensors is least when compared to the other cases.

Practical Sensor Networks usually include one or more of the above functional capabilities. Despite the prolific conceptualization of sensor networks as being useful for large-scale military applications, the reality is that the best migration path for sensor networks research into non-academic applications is via integration with existing engineering applications infrastructure. For example, sensor networks have the potential to offer fresh solutions to fault diagnosis, health monitoring and innovative human-machine interaction paradigms. [1], [7], [17],[24].

The multitudes of design challenges imposed on Sensor Networks tend to be quite complex and usually defy the analytical methods that are quite effective for traditional networks. At current stage of technology very few Sensor Networks have come into existence. Although there are many unsolved research problems in this domain, actual deployment and study is infeasible. The only practical alternate to study Sensor Networks is through simulation, which can provide better insight to behavior and performance of various algorithms and protocols. The goal of SensorSimulator is to provide a framework to closely model and simulate various Sensor Networks scenarios.

The basic architecture of SensorSimulator is similar to that of SensorSim [8] developed by Park et. al. However SensorSim was built as an extension to NS-2 [4]. NS-2 was designed to simulate traditional wired networks. The design of NS-2 causes

unnecessary interdependency between modules. This dependency makes the addition of new protocols extremely difficult, mastered by only those who have intimate familiarity with the simulator. The SensorSim project was unfinished and is currently unavailable for public distribution and use. The difficulty to extend is not the main issue for simulators used for traditional networks. For sensor networks there exist no dominant protocols or algorithms, since sensor networks are usually application specific and it is highly unlikely that a single protocol or algorithm will be optimal under different situations.

The framework of SensorSimulator has been built on the OMNeT++[13] simulation environment. The next section provides a brief description of the available simulators. Section 3 gives an overview of the OMNeT++ simulation environment. Section 4 gives an overview of the proposed simulation framework for sensor networks. Section 5 gives detailed design of the SensorSimulator. Section 6 gives description about the demonstrative implementation of Directed Diffusion with GEAR along with the performance with MAC 802.11. A performance study is presented by comparing the results of simulation on the SensorSimulator framework against the simulation on NS2. Finally we conclude with a list of research avenues for future work.

Chapter 2: Currently Available Simulators

NS-2 is a well-established discrete event simulator[15] that provides extensive support for simulating TCP/IP, routing and multicast protocols over wired and wireless networks [6]. Radio propagation model based on two ray ground reflection approximation and a shared media model in the physical layer, an IEEE 802.11 MAC protocol in the link layer and an implementation of dynamic source routing for the network layer were developed in the Monarch project [14].

SensorSim builds on NS-2 and claims to include models for energy and the sensor channel [8]. At each node, energy consumers are said to operate in multiple modes and consume different amounts of energy in each mode. The sensor channel models the dynamic inter-action between the physical environment and the sensor nodes. This simulator is no longer being developed and a public release is not available.

OPNET Modeler is a commercial platform for simulating communication networks [16]. Conceptually, OPNET model comprises processes that are based on finite state machines and these processes communicate as specified in the top-level model. The wireless model uses a 13-stage pipeline to determine connectivity and propagation among nodes. Users can specify frequency, bandwidth, and power among other characteristics including antenna gain patterns and terrain models.

J-Sim [10] is another object-oriented, component-based, discrete event, network simulation framework that is written in Java. Modules can be added and deleted in a plug-and-play manner and J-Sim is useful both for network simulation and emulation by incorporating one or more real sensor devices. This framework provides support for

target, sensor and sink nodes, sensor channels and wireless communication channels, physical media such as seismic channels, power models and energy models.

GlomoSim is a collection of library modules, each of which simulated a specific wireless communication protocol in the protocol stack [23]. It is used to simulate Ad-hoc and Mobile wireless networks.

In a recent report [25] the following paragraph summarizes the need for a new simulator.

“NS-2, perhaps the most widely used network simulator, has been extended to include some basic facilities to simulate Sensor Networks. However, one of the problems of NS-2 is its object-oriented design that introduces much unnecessary interdependency between modules. Such interdependency sometimes makes the addition of new protocol models extremely difficult, only mastered by those who have intimate familiarity with the simulator. Being difficult to extend is not a major problem for simulators targeted at traditional networks, for there the set of popular protocols is relatively small. For example, Ethernet is widely used for wired LAN, IEEE 802.11 for wireless LAN, TCP for reliable transmission over unreliable media. For sensor networks, however, the situation is quite different. There are no such dominant protocols or algorithms and there will unlikely be any, because a sensor network is often tailored for a particular application with specific features, and it is unlikely that a single algorithm can always be the optimal one under various circumstances.

Many other publicly available network simulators, such as J-Sim, SSFNet[26], Glomosim and its descendant Qualnet[27], attempted to address problems that were left unsolved by NS-2. Among them, J-Sim developers realized the drawback of object-

oriented design and tried to attack this problem by building a component-oriented architecture. However, they chose Java as the simulation language, inevitably sacrificing the efficiency of the simulation. SSFNet and Glomosim designers were more concerned about parallel simulation, with the latter more focused on wireless networks. They are not superior to NS-2 in terms of design and extensibility."

Chapter 3: OMNeT++ Simulation Environment

OMNeT++ [13][20], Objective Modular Network Test-bed in C++ is a public-source, component-based, modular and open-architecture simulation environment with strong GUI support and an embeddable simulation kernel. Its primary application area is the simulation of communication networks, but because of its generic and flexible architecture, it has been successfully used in other areas.

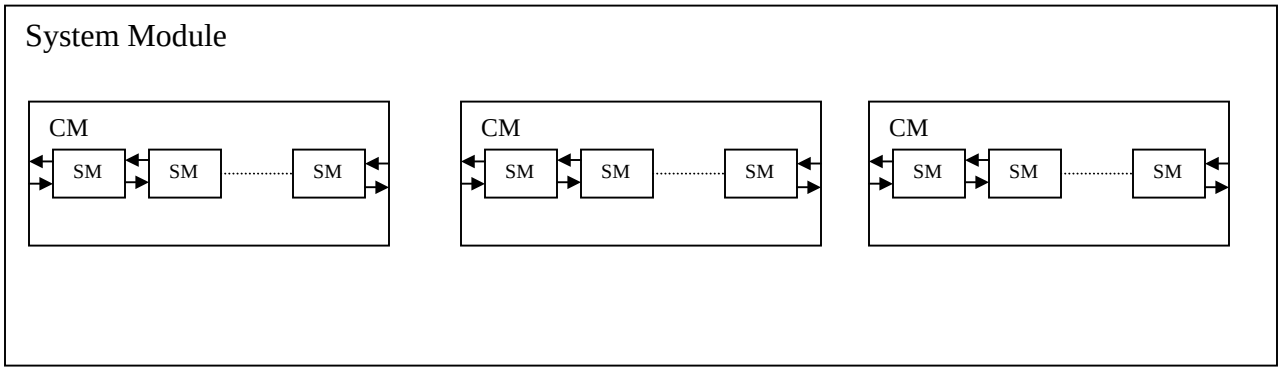
The OMNeT++ model consists of hierarchically nested modules. The top-level model is the system model, which encompasses the complete simulation model and is referred to as the “networks”. The system contains sub-modules which themselves may have sub-modules. Thus the modules can be described to any depth of nesting as a result able to describe complex system models as a combination of a number of simple modules. Modules that contain sub-modules are called compound models. Simple modules contain the algorithms in the modules and form the lowest level of module hierarchy. The user implements the simple modules in C++, using the OMNeT++ simulation class library. Modules communicate by message passing which may be a complex data structure.

Modules may send messages directly to their destination or through a series of gates and connections to other modules. The messages can represent frames or packets in a computer network simulation. The local simulation time advances when the module receives messages from other modules or from the same module as self-messages, which is the representation of timers in simulation world. These self-messages are used to schedule events to be executed by itself at a later time.

Each of the modules has input and output interfaces called Gates through which message passing between modules is achieved. Messages are sent out through the out-Gate and received through the in-Gate.

Connections are created between the sub-modules or between sub-module to compound module depending on the requirement of the system or the topology. The structure of a system maybe represented as shown in figure 3.1

As a hierarchal model is followed, the messages typically travel through a series of connections that start and end at simple modules.



CM – Compound Module

SM – Simple Module

⟶ Gate between sub modules and sub module and compound module

Figure 3.1: Simple and Compound modules in OMNeT++

The description of the topology, the structure and specification of the modules, the Gates and connections are specified through the Network Description Language (NED). NED files are not used directly: they are translated into C++ code by the NEDC compiler, then compiled by the C++ compiler and linked into the simulation executable.

The actual behavior of the modules is written in C++ code using the OMNeT++ simulation library and the description of the modules:- parameters, Gates , connections between different modules, is specified by the NED language. In this way, there is a separation of behavior and interface definition. This allows reusability of module interfaces defined by NED code. For the implementation of the simple modules OMNeT++ offers an API consisting of a simple module interface, a message interface and a rich simulation library providing support for essential functions, as a lot of routines for the simulation purposes as e.g. I/O-functions, statistics-classes for gathering the achieved results, etc. but also more general stuff like statistical distributions, random numbers generators and even container classes like queues, stacks, containers, etc.
The simulation tool allows the collection of the final results and also the statistics of the performance of the simulation transparently into scalar and vector files.

Simulations runs are easy to configure and run through initialization files, through which the various data values of the parameters in modules can be specified or changed and simulation re-run with requiring the re-compilation of the simulation setup.

In this way OMNeT++ represents a simulation engine, keeping track of the events generated and making sure that messages are delivered to the right modules at the right time, thus accomplishing the task for discrete event simulation.

Chapter 4: Overview of the Proposed Simulation Framework

The goal of the SensorSimulator is to provide a framework to closely model the sensor network scenario. The broad outline of any sensor network can be represented by this high-level representation as shown below in figure 1. The sensor model can be represented by the sensor node model and the Power model.

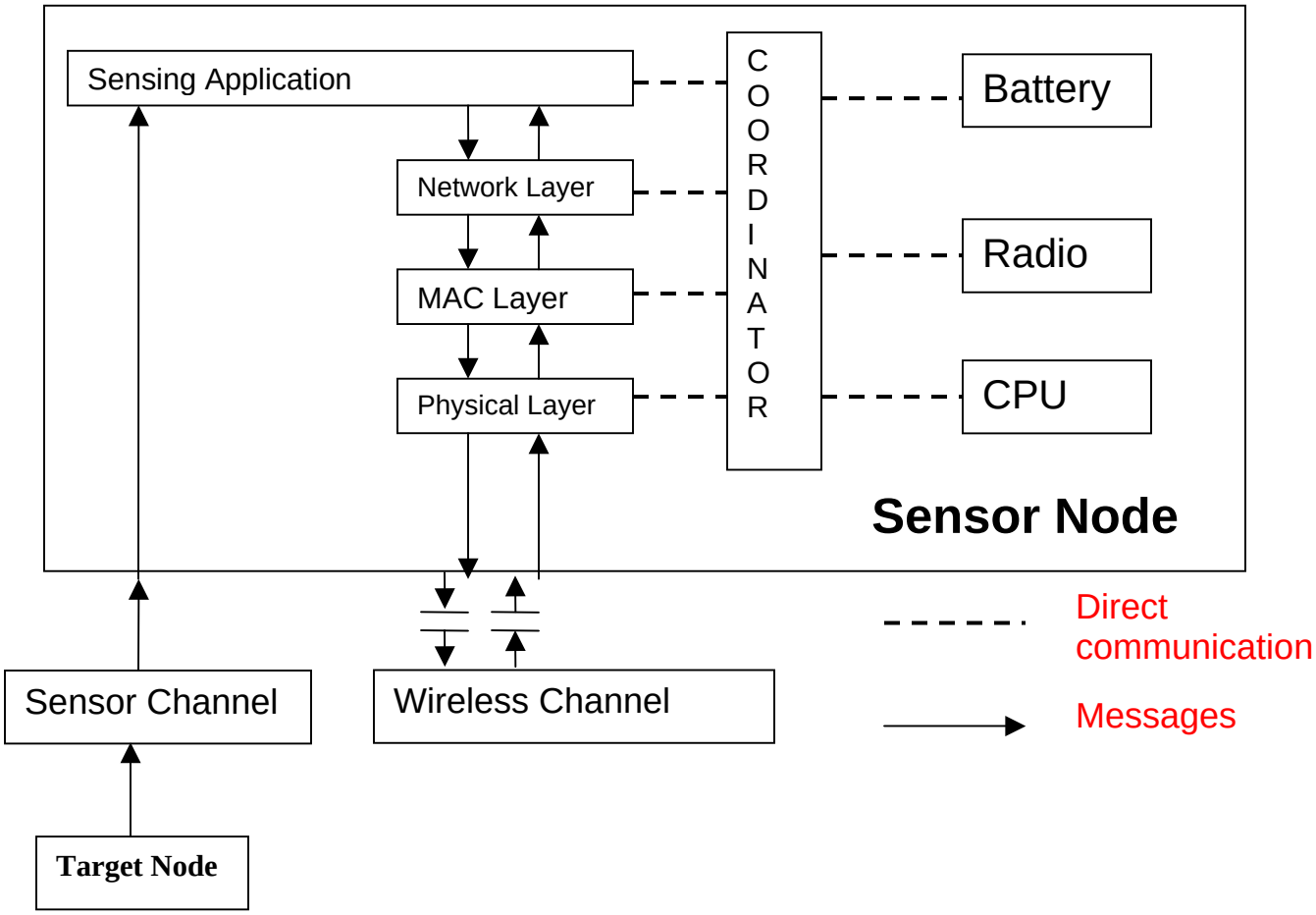


Figure 4.1: Sensor Node Representation in a Network

The sensor node model represents the network stack and the sensor applications. The power model represents the hardware of the sensor node: the CPU, sensor and RF transceiver. The two models act in parallel to represent the software and the hardware. The hardware model's state is changed based on the function carried out by the sensor node model. The power model is hardware abstraction of sensor node, which interacts with sensor node model to estimate the power usage. In the power model there is a single energy source and many consumers. The battery is the single provider with finite amount of energy. The consumers are radio, CPU and other sensing devices that maybe added to the device as illustrated in figure 1. Consumers report their power usage change to the energy source (battery) and the energy source updates the remaining energy.

SensorSimulator framework consists of the network of sensor nodes that can communicate by wireless means. The layers of network stack in the sensor model are configurable based on the protocol needed for the simulation. The users of this framework have to write the code for their own protocols and integrate it into the framework. The simulation and network parameters can be specified in the configuration file and then simulation can proceed. The parameters can be changed without any code change or need of recompilation.

The sensor simulation framework can be described in a target tracking application, as the basic functionality of sensors is to sense or detect events or conditions in the environment around. The sensing application maybe described as the sensors detect events that are generated by a target or object in the environment near the sensors. The sensors then need to report the event or data collected to the base

station or the sink using a multi-hop routing approach in an efficient manner. The sensor network can be represented by sensor nodes that sense and detect events, the Target Node that generates events and the Sink that consumes the data or the final destination of data delivery or the node that can query the network to obtain specific data. The sensors sense events through the sensor channel, which is the representation of the sensing propagation model. The detected events are propagated across the network through the wireless channel that has implementation of the different propagation models in wireless medium.

A sensor node has the network protocol stack that enables it to detect the events generated by the target node and also to send the messages to the other sensor nodes in the network depending on the different protocols implemented at each of the layers of the protocol stack. The functioning of the framework and abstract view of sensor network as shown in figure 4.2 is described below:

The target node moves across the network in a fixed path or in a random fashion at a configurable speed. The target node sends stimuli to the sensor channel. The sensor channel in turn will pass on the stimuli to only those sensor nodes in the vicinity of the target node. A sensor node is able to receive the stimuli only if the signal strength power of the received packet is above a certain threshold. The propagation model configured at the sensor channel determines the attenuation of the signal and the received signal strength power.

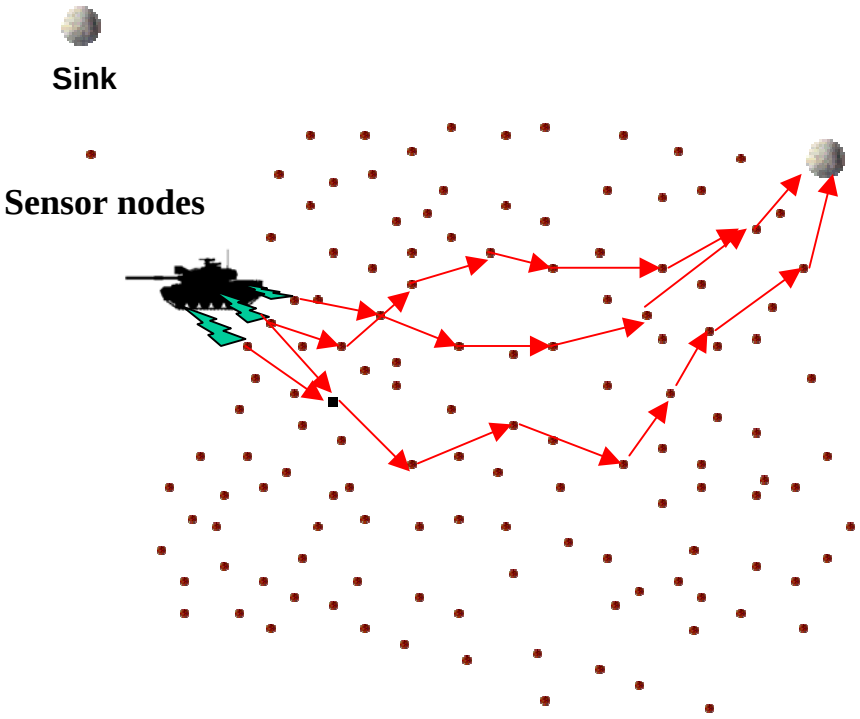


Figure: 4.2 Abstract view of a Sensor Network

In the sensor node a number of algorithm or protocols can be implemented depending on the application such as aggregation, cluster functionality, security implementations and other in-network processing implementations at the Sensing Application layer.

The data needs to reach the sink or the base-station through the wireless channel. A prototype of the sensor device can transmit data to a distance of around 30 feet and so it is possible that the sink is not in the vicinity of the sensor node and so a multi-hop route needs to be followed with other sensor nodes acting as routers to pass on the message to the sink. The sensor nodes are memory and energy constrained devices and so may fail or die due to power depletion or other environmental conditions and so the network topology will always change. The routing protocol should be able to handle the dynamics of the network and be able to transmit the data to the sink in a

reliable, timely and energy efficient or conserving manner. In the framework provided we have implemented directed diffusion and the Geographic Energy aware routing protocol (GEAR) to test the functioning of the simulation framework and to show the proof of concept of a sensor network protocol implementation. At the wireless channel other nodes depending on the propagation model will receive the data sent by a node. The freespace model and the two-ray propagation model have been implemented.

Chapter 5: Design Approach

In this section we describe the design approach taken by us to represent and define the SensorSimulator framework.

The simulator is designed in the form of a layered architecture and the communication between the different layers and modules are accomplished through message passing. The general architecture of a sensor network is as shown in figure 4.1. The description of the different modules of the framework and their interaction with the other modules is described in this section

The SensorSimulator simulation is defined in the SensorNetwork module. SensorNetwork module is the compound module that contains all the different modules like the TargetNode, sensor Nodes, Wireless channel and Sensor channel

5.1 Target Node

TargetBase class is the base class that represents the Target Node. The TargetBase has the base class functionalities that are essential for any TargetNode such as the position of the target node and the ID. TargetNodeSimple extends the TargetBase and has the functional implementation of the TargetNode. The TargetNode module maintains Gate connection with the sensor channel. The TargetNodeSimple class generates stimuli and passes the message to the sensor channel. The mobility model provides the functionality of the TargetNode movement thereby generating stimuli at various points in the network.

5.2 Sensor Channel

The SensorChannel Module and the SensorChannel Base class represent the Sensor channel. The SensorChannel module maintains Gate connections to

SensorNode Module as well as to the TargetNode. The SensorChannel Base is an abstract class for SensorChannel property classes

The location information of all the sensor nodes is maintained at the Network level, which is the parent module of the Sensor Channel. This kind of an abstraction has been designed, as the network module that encompasses the whole simulation model must have information of the topology of the network.

The SensorChannel class decides the nodes that should receive the stimuli depending on the propagation model and the channel properties. We have implemented the

- **Seismic Propagation**

The Seismic propagation model calculates the received signal power as a function of distance between sender and receiver and the attenuation factor. The received signal power Pr is calculated as

$$P_r = \frac{P_t}{\max(d, d_0)^{f_a}}$$

where

P_t : power with which signal transmitted

d : distance between sender and receiver

d_o , f_a : signal attenuation factor ,can be configured

- **Acoustic Propagation**

In Acoustic Propagation, the received signal power Pr is calculated according to the following equation

$$P_r = N(p \times \mu_g, \sigma_g^{\ 2})$$

where

$$p = \frac{P_t}{\max(d, d_o)^{f_a}}, \mu_g = U(\min_g, \max_g)$$

P_t: power with which signal was transmitted

d : is the distance between sender and receiver

$\min_g, \max_g$, μ_g , σ_g : min, max, mean and variance of microphone gain

d_o , f_a : signal attenuation factor, can be configured

5.3 Sensor Node

The SensorNode module is a compound module that has the different layers of the protocol stack as the sub-modules. The sensor Node module definition and the class represent all the components of the sensor node.

5.3.1 Coordinator Module

Coordinator class has the functionalities that coordinate the activities of the hardware and the software modules of the sensor node. The Coordinator need to be extended and functionality added for access to properties of new hardware or consumers added. The Coordinator class has the reference to all the layers in the sensor node and all the layers in the sensor node may access the Coordinator. Thus through the Coordinator any layer may access and update the properties of the other layer. For example the battery needs to be informed on transmission or receiving packets and the energy consumption updated at the node.

The Coordinator class is responsible for registering the sensor node to the sensor network. Registering of the sensor node is an indication that the sensor node is up and functioning. On complete energy depletion the node is unregistered from the sensor network.

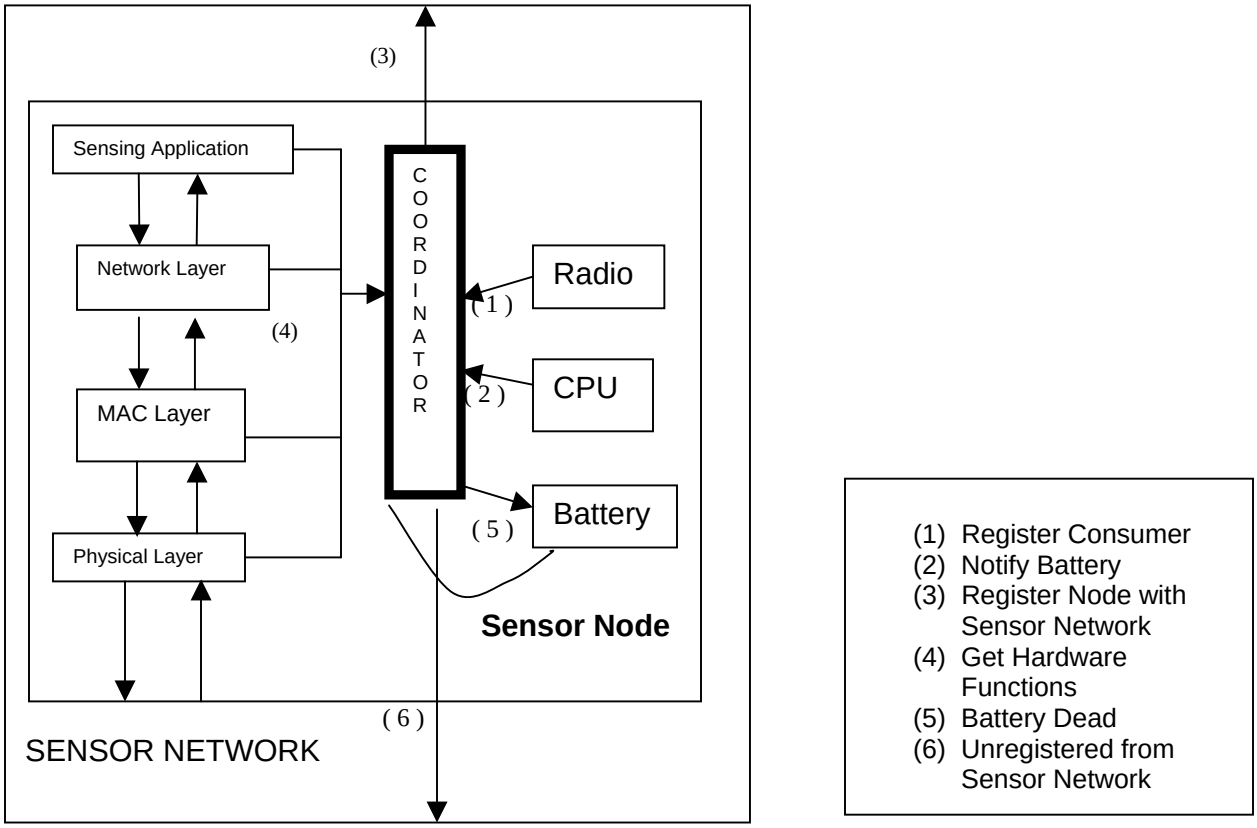


Figure 5.1: Sequence of activities carried out by the Coordinator in the simulation

5.3.2 Hardware Modeling

5.3.2.1 Battery Module

a) BatteryBase is the abstract class for the different battery models that can be added to the Battery Model.

b) BatterySimple is a subclass of BatteryBase and updates the energy depending on the number of consumers and the state of activity of the consumers.

c) The specifics of the energy consumption rate and the operation maybe extended to the battery model.

5.3.2.2 CPU Model

a) CPU Base is the abstract class for the different CPU models.

b) CPU Simple has implementation of the power consumption of the CPU in different states: idle, sleep and active.

5.3.2.3 Radio Model

a) RadioBase is an abstract class for the different Radio models.

b) Radio Simple a subclass of RadioBase updates the energy of the battery depending on the state of the Radio: idle, sleep, transmit, receive.

The values for the different properties of the hardware and consumers maybe provided through the configuration file.

5.3.3 Software Model

The software model represents the different layers of the wireless protocol stack:

5.3.3.1 Sensing Application Layer

Implements the application specific functions and other in-network processing depending on the application simulated such as aggregation and pass the result on to the network layer. The Sensing Application Layer receives the stimuli from the TargetNode through the sensor channel and takes appropriate action.

5.3.3.2 Network Layer

Implements the routing protocol for sensor networks. Directed Diffusion and Geographic aware routing protocol have been implemented in this layer. The network Layer receives the message from the application layer, and then transforms the message to a macPacket type message and sends it to the bottom layer to the MAC layer. The NetworkPacket maybe broadcast or unicast to specific node (sink node).

5.3.3.3 MAC Layer

The MAC_802_11 and Simple Mac implementation has been done at this layer. The MacType message received form the above layer is sent to Wireless Channel through the PhyLayer that in turn interacts with the radio model to transform the state of the radio before sending the message to the wirelesschannel. Energy is updated at regular intervals in the node as and when the different consumers change state.

5.4 Wireless Channel

The wireless channel represents the medium through which the sensor nodes communicate. Any message from a node to the wireless channel is sent to all the neighbors within its transmission region with a delay d where d is (Distance between the communicating Sensor Nodes) / Speed of Light.

Various Radio Propagation models are used to predict the received signal power of each packet. These models affect the communicating region between any 2 nodes and are derived by the Wireless Channel.

- **Free Space Propagation Model**

The free space propagation model assumes the ideal propagation condition that there is only one clear line-of-sight path between the transmitter and receiver. H. T. Friis presented the following equation to calculate the received signal power in free space at distance _ from the transmitter

$$P_r = (P_t * G_t * G_r * \lambda^2) / (4\pi)^2 * d^2 * L^2$$

P_t is the transmitted signal power

P_r is the received signal power

G_t, G_r are the antenna gains of the transmitter and the receiver respectively.

L is the system loss, and _ is the wavelength.

- **Two-ray ground reflection model**

A single line-of-sight path between two mobile nodes is seldom the only means of propagation. The two-ray ground reflection model considers both the direct path and a ground reflection path. This model gives more accurate prediction at a long distance than the free space model. The received power at distance ‘d’ is predicted by

$P_r = (P_t * G_t * G_r * h_t^2 * h_r^2) / (d^4 * L)$

h_t and h_r - heights of transmit and receive antennas respectively

The above equation shows a faster power loss than for Free Space Model as distance increases.

Chapter 6: Demonstrative Use of the Simulation Framework

In this section we demonstrate the implementation of the above framework to prove the concept of usage of the framework to implement a protocol for sensor networks. We simulated directed diffusion with Geographic energy aware routing (GEAR) on the framework in exactly the same setup as implemented in NS2 to compare the performance of the simulator with respect to NS2.

6.1 Geographic Routing (GEAR)

Geographical Energy Aware routing [28] uses a geographical and energy aware neighbor selection heuristic to route the packet towards the target region. The process of forwarding a packet towards the region involves

- choosing a neighbor that is closest to the destination among all the neighbors
- when all neighbors are away, chose a neighbor that minimizes the cost value to the neighbor which is computed as

$$c(N_i, R) = \alpha d(N_i, R) + (1 - \alpha) e(N_i)$$

where $d(N_i, R)$ is the distance from node N_i to the centroid D of the region R normalized by the largest distance among all the neighbors N_i and $e(N_i)$ is the consumed energy at node N_i normalized by the largest consumed energy among the neighbors of N.

On reaching the region of interest recursive forwarding technique is followed to flood the packet in the region to minimize the cost consumption.

6.2 Directed Diffusion

Directed Diffusion is a data-centric information dissemination paradigm for Wireless sensor networks [3][4][5]. The elements of directed diffusion are sending interests, setting up gradients, and reinforcing the paths. An interest message is a query that has the information about the data that is required from the sensor nodes. Data can be either collection of information or an event triggered by some physical phenomena. Gradients are directional state created in each node, set towards the neighbor from which interest is received. One or more of these paths are reinforced. Each task is named in an attribute list. The task description specifies an interest for data matching. Interest is a named task. Interest is sent into the network from a sink. Interest may also have information about duration of the task and the interval at which response is required. Initial interest messages are also called Exploratory, and it tries to form a connection with the nodes that have the required data. At each node a cache of distinct interests is maintained (this allows interest aggregation). They contain information about the previous hop. The interests propagate through the network. The nodes in the region or nodes that have data for a particular interest send data marked as exploratory through the gradient established. As a result exploratory data may follow multiple gradient paths to the query source node. Once the exploratory data is received, the query source node reinforces one of the paths based on the routing protocol being used. To reinforce the node sends a positive reinforcement message to the neighbor initiating the sending of data. The data-sending interval is less than the exploratory sending interval. The reinforced neighbor reinforces its neighbor in turn, and this is done

all the way till the data source. Data messages are marked as exploratory at a regular interval.

6.3 Implementation Details

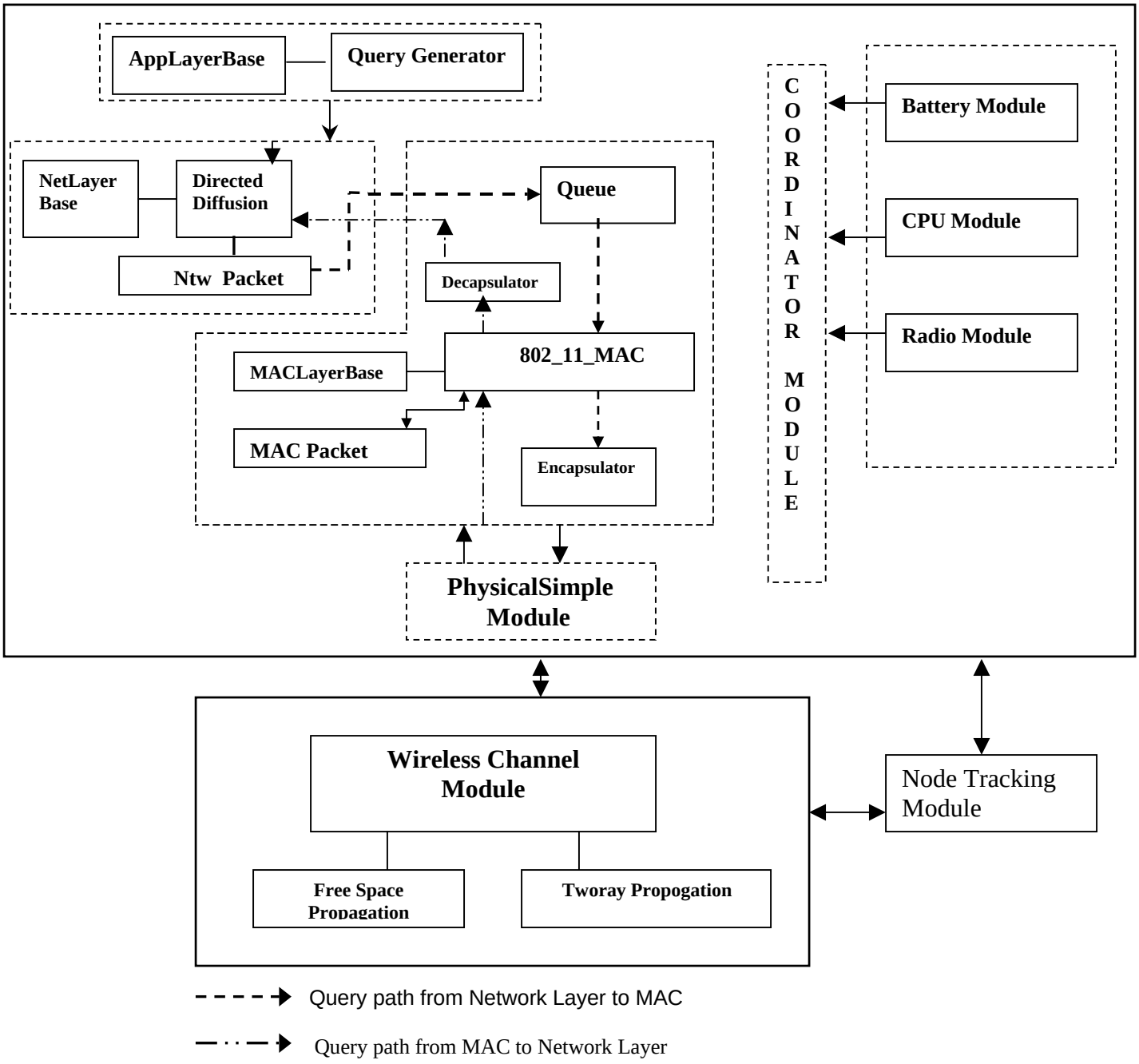


Figure 6.1 Implementation and flow diagram of Directed Diffusion-GEAR with MAC 802.11 Simulation

6.3.1 Directed Diffusion with GEAR implementation

We have implemented Directed Diffusion along with Geographic Routing. The Application Layer generates interests that specify the region, the kind of data required and rate of delivery of data. The structure of the query message is as shown in figure 6.2. The attribute structure has features to specify interest properties such as the region of interest and any user-defined query messages.

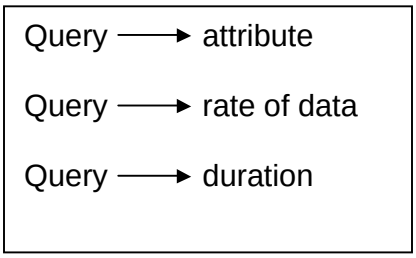


Figure 6.2 Structure of a Query Message

These nodes that initiate the interest are called Subscribers. On receiving the interest message, the network layer broadcasts the beacon messages in the network. The immediate neighbors of the node on receiving beacon messages reply back with beacon-reply type of message that contains their geographic location and the energy left in them. The node waits a period of time to receive the beacon-reply from its neighbors. The interest message is then forwarded to the node that has a higher estimated cost to the region as calculated by the GEAR protocol. The next node follows the same procedure and forwards the message towards the region by Geographic Routing. If a node in the path does not have any neighbors or all its neighbors are away from the region, then it sends a message to its parent node that it is a dead-end. The parent node on updating the cost of the unreachable node, forwards the query in an alternate route towards the region. In the specified region, the interest is recursively

flooded. The interest cache is maintained at each of the nodes in the path with its gradient of interest to each of the neighbors. The nodes in the region that have the specified properties of the interest send out data. These nodes are referred to as Publishers.

The data is marked as Exploratory to reinforce the path that was taken by the interest. On receiving the data marked as Exploratory by the subscriber, positive reinforcement message is sent out by the Subscriber node. Each node on path forwards this message thus reinforcing the path to the region. When the node reinforces a path, its cost to the region is known and this cost is sent back to its source node, which updates the cost information of that node to the particular region of interest. Thus the path with the highest cost is always maintained, reinforcing the route. The data from the region follow the path established by the reinforced messages. The nodes in the region send out data at the rate that is specified in the query. Data caching is implemented in intermediate nodes and so the data requested by different subscribers from the same region maybe satisfied by the common node in the path thus reducing the traffic and redundant messages. The data marked as exploratory are sent to identify better paths and reinforce at regular intervals. Also the neighbor- updating procedure phase is carried out, i.e. at regular intervals the beacon messages are broadcast and beacon-reply messages are sent by neighbors thus maintaining latest neighbor information.

6.3.2 MAC 802.11

The implementation detail of the MAC 802.11[9][19] protocol is described below. Any network layer packet received by the MAC-802-11 module is encapsulated into MAC frame with the MAC header added to it. The Network layer packets have the

information whether the packet needs to be broadcast or sent to a particular node. Broadcast packet is encapsulated into Broadcast MAC frame with appropriate MAC Header and is put in the Messages-queue of the MAC Layer. If the Network packet is for a particular destination, RTS frame is created and is inserted in the Messages-queue of MAC layer. If the Network packet length is more than the MAC frame, it is fragmented and the fragments for that Network Packet are created with MAC headers and are inserted into the Fragments Queue. The MAC layer then waits for the channel to be idle to send its frame from the Messages-queue. MAC layer has a NAV Timer, which specifies the busy/idle state of the medium. NAV Timer set for a node implies that the channel is busy. If the NAV Timer gets expired then the MAC layer waits for the channel to be free for DIFS time and if the channel is still idle after DIFS timer gets expired, it then goes into Exponential BackOff. It then waits for a random time set by the BackOff Timer. The node whose BackOff Timer expires earlier will get the chance to transmit its next frame. All the intermediate nodes receive this frame, set their NAVTimer to the value obtained from the Header field of the received frame. Then the BackOff Timer of the intermediate nodes is stopped from decrementing. Once the channel becomes idle (when the NAVTimer expires) all the nodes start decrementing their BackOff Timer. The node whose Back Off Timer expired earlier and got the channel will send the first message from the Messages Queue. If it is a broadcast message, then all the nodes in its region receive it and the MAC layer of those nodes decapsulate the Network packet and send it to the Network Layer. If it is a RTS frame, the Destination node checks whether its NAV timer is set or not (its transmission region is busy or not) and then responds to it by sending CTS. All the other intermediate nodes receiving this RTS

update their NAV Timer to the CTS+DATA+ACK duration, which implies that the channel is busy for that duration, and hence refrain from transmitting during this interval. If the Destination node receives more than two RTS requests within a time interval then collision occurs and the Destination node does not respond (send CTS) to any of these RTS requests. The Source node that is sending RTS have an RTSExpired Timer set for RTS frames, when they are sent to the Destination node. This timer is scheduled to expire after RTS+CTS duration. If the Source node does not receive CTS within this duration, RTSExpired Timer gets expired and retry counter of that RTS frame is incremented. If the retry counter is less than ShortRetyLimit (as per the specification), then the Contention Window is doubled and the random time set by the BackOff Timer is chosen between one and the Contention Window size. Retry counter on reaching the ShortRetryLimit, the message (RTS and corresponding Fragment) is dropped by the MAC.

If the Destination node responds to RTS by sending back the CTS, the intermediate nodes for CTS will update their NAVTimer obtained from the Header field of CTS frame (Data + Ack duration) and hence refrain from transmitting during this interval. Once the Source node gets the CTS, it will send the corresponding fragment of the Network Packet to the Destination and waits for an Acknowledgement. The Destination node upon receiving the Data frame

Property	Values
SIFS	**10 μsec**
DIFS	**28 μsec**
Slot Time	**20 μsec**
Data Rate	**1 Mbps**
RTS Length	**44 bytes**
CTS length	**38 bytes**
ACK Length	**38 bytes**
DATA Length	**Variable**

Figure 6.3 Parameters for 802-11 Simulations

decapsulates the Network packet, sends it to the Network layer and sends back the Acknowledgement to the Source node. Once the Source node gets the Acknowledgement it checks and sends if there are any other fragments to be sent to this node without any additional RTS frames. The parameter values for the MAC 802.11 simulation are shown in figure 6.3

6.4 Experimental Results

In this section, we presents results from three different studies. First, we establish that the Directed Diffusion simulation in our work is consistent with the Diffusion implementation in NS-2. Next we compare the performance, with respect to execution time and memory used, between our simulation and that of NS-2.

6.4.1 Validating Directed Diffusion Implementation

In this experiment, we considered Sensor Networks with different number of nodes between 5 and 200. For each Sensor Network, we identified the maximum size of the sensor field (with respect to grid coordinates). Then, we identified a fixed number of query generating nodes and distributed these nodes randomly in the sensor field. Next, we determined a target region and specified the boundary of the region in terms of the

grid coordinate and the number of sensor nodes in the region. We executed the simulation for a specified duration and observed the ratio of the number of packets generated in the region and the number of packets received by the query generating nodes. 802-11 MAC is being considered at the MAC layer with a simple pass through Physical layer for these simulations. The results for 5-200 nodes are shown in Figure 6.4 with individual query packet delivery. The results show that for a similar topology and simulation environment, the delivery ratio is comparable with ns2

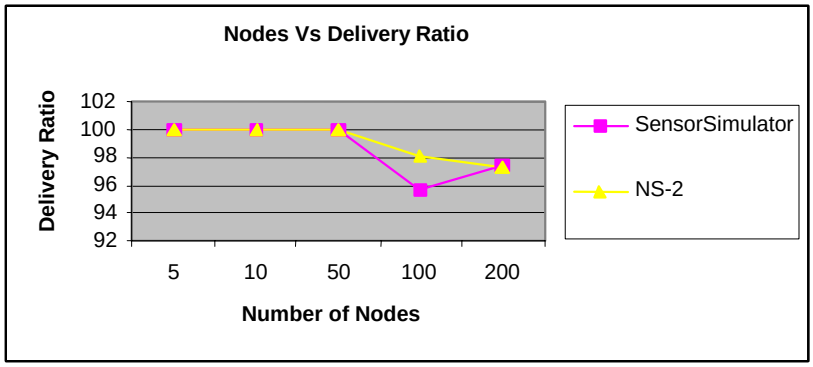


Figure 6.4 Message delivery ratio SensorSimulator Vs NS-2

The experimental setup of directed diffusion with GEAR has been implemented as a module in the SensorSimulator with the following functionality. It was simulated similar to the simulation on NS2

6.4.2 Directed Diffusion-GEAR with SimpleMAC

The simulation implementation of Directed Diffusion with GEAR is as described in section 5.3.1. SimpleMAC is used at the data-link layer. SimpleMAC has the simple implementation and do not have the details mechanism for collision avoidance. Whenever a packet is received from the above layer, it is forwarded to the physical layer to the Wireless channel to the other nodes. This kind of implementation has been done

to check the performance of the simulator on executing Directed Diffusion with GEAR and there are no extra control packets from the other layers. In order to test the performance of the simulation we ran the setup with queries generated by 10 nodes at random locations in the network. A similar test was performed with 100 nodes generating queries. The queries follow a multi-hop route to the region following the procedure mentioned above. Once the query reaches the region the data is sent back once every 5 seconds for the complete simulation time by all the nodes in the region. The objective of this kind of setup is to check whether the simulation framework is able to handle the traffic generated and run to completion as well as to check the amount of time required to run the simulation.

Figures 6.5 and 6.6 shows the performance of the two simulators (NS-2 Vs. our Simulation) for the setup with 10 nodes and 100 nodes generating queries. It is can be observed that the performance of both the simulators NS2 and SensorSimulator showed similar results at less number of nodes in the network. As the number of nodes in the network increases, SensorSimulator is able to handle the traffic and the events generated in a better fashion so as to complete the simulation in a reasonable time faster than NS2. It has been observed that NS2 ran out of memory for network above 2000 nodes.

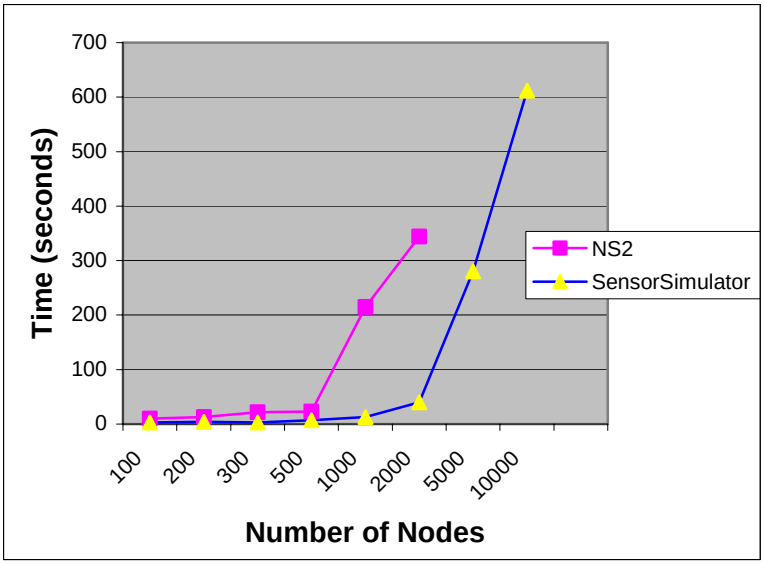


Figure 6.5 Execution time for 10 Queries for 150 simulation seconds

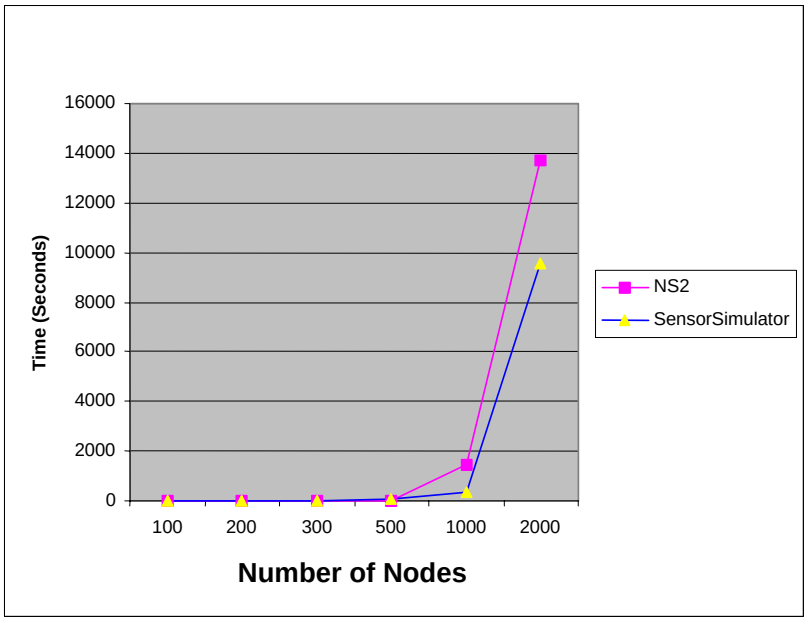


Figure 6.6 Execution time for 100 Queries for 150 simulation seconds

During the simulation runs, we measured the memory allocated before the start of the simulation, i.e. it gives the memory usage for the initialization and the setup of the

objects of the simulation. The memory usage during the simulation was also measured. The results for the memory usage are as shown in figure 6.7 and figure 6.8 for the 10 nodes sending queries for the initial setup of the whole network and then during the simulation. Figure 6.9 and figure 6.10 shows the performance of the simulators for 100 nodes sending queries to the region. This shows that the data structures used for the simulation are used in a scalable manner to represent the different classes and the interaction with the framework. It can also be observed that the rate of memory usage increases at a faster rate for NS2 than for SensorSimulator thus allowing for large simulation setup and more scalability in SensorSimulator than NS2.

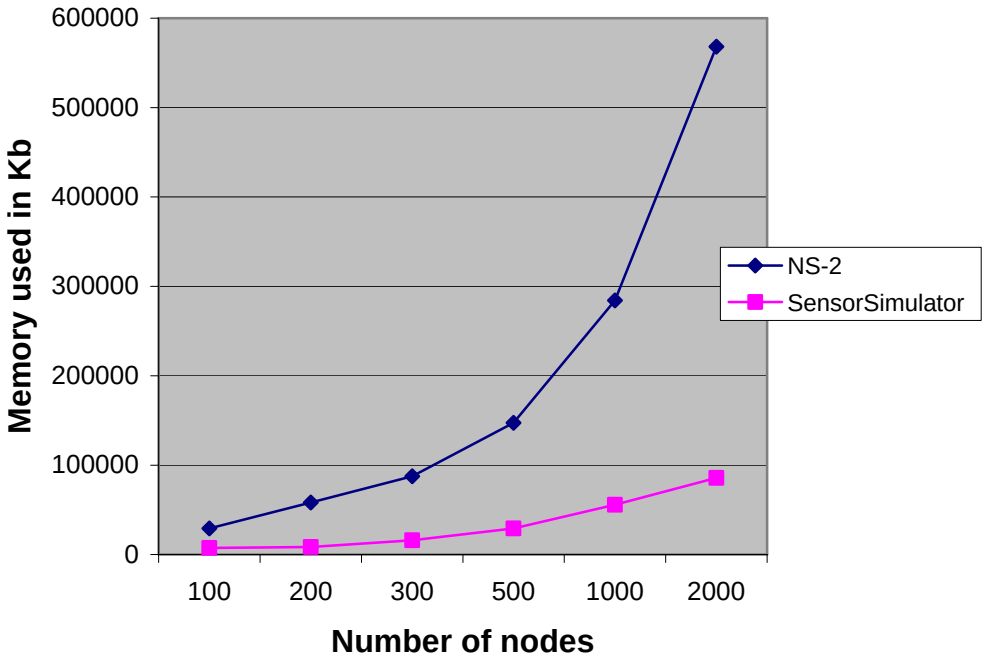


Figure 6.7 Memory Utilized to setup the network for 10 Queries

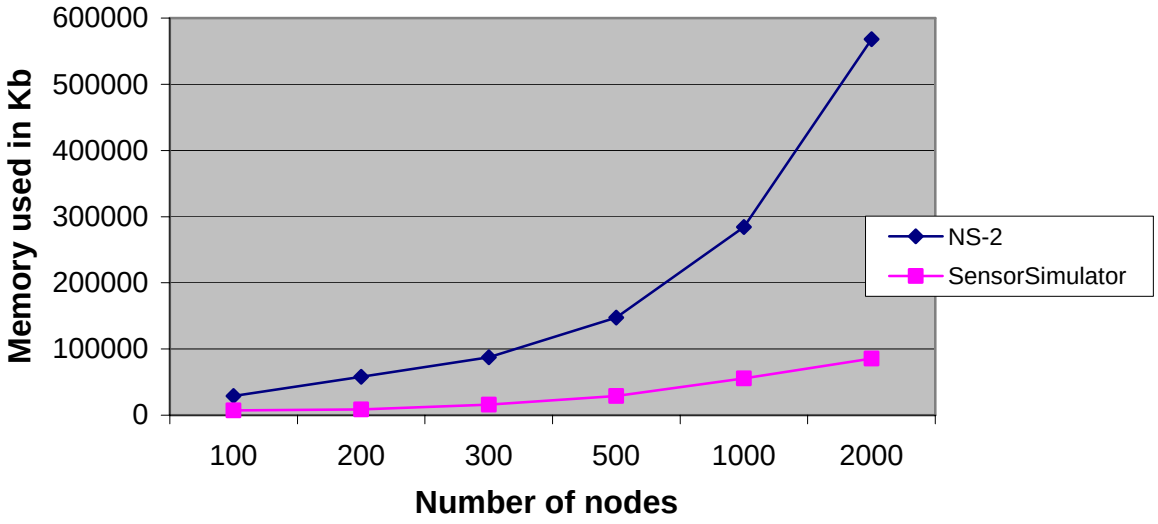


Figure 6.8 Memory Utilization during simulation for 10 Queries

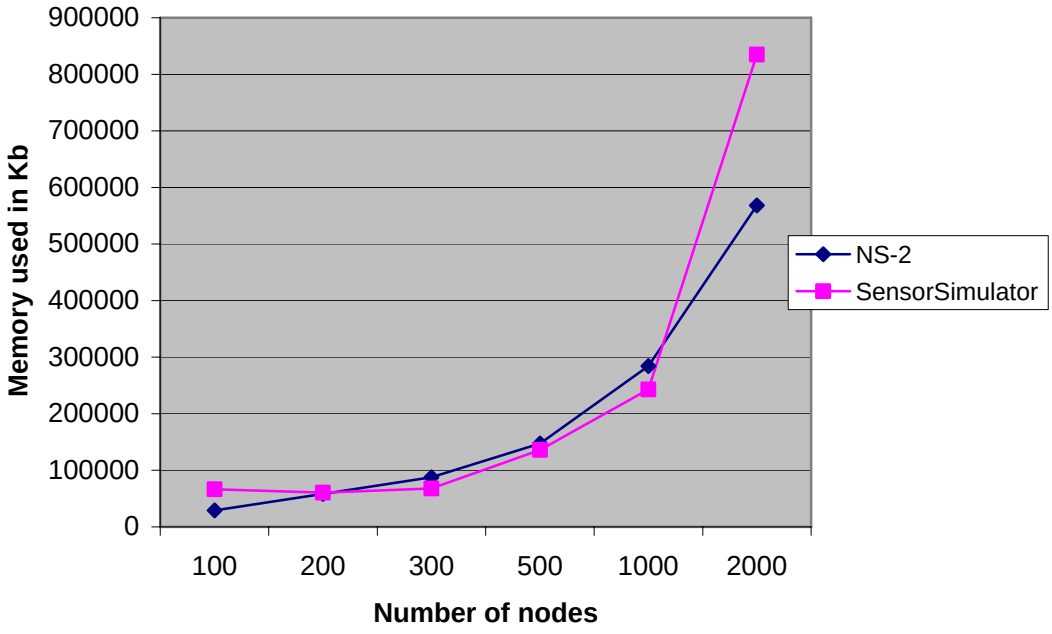


Figure 6.9 Memory Utilized to setup network for 100 queries

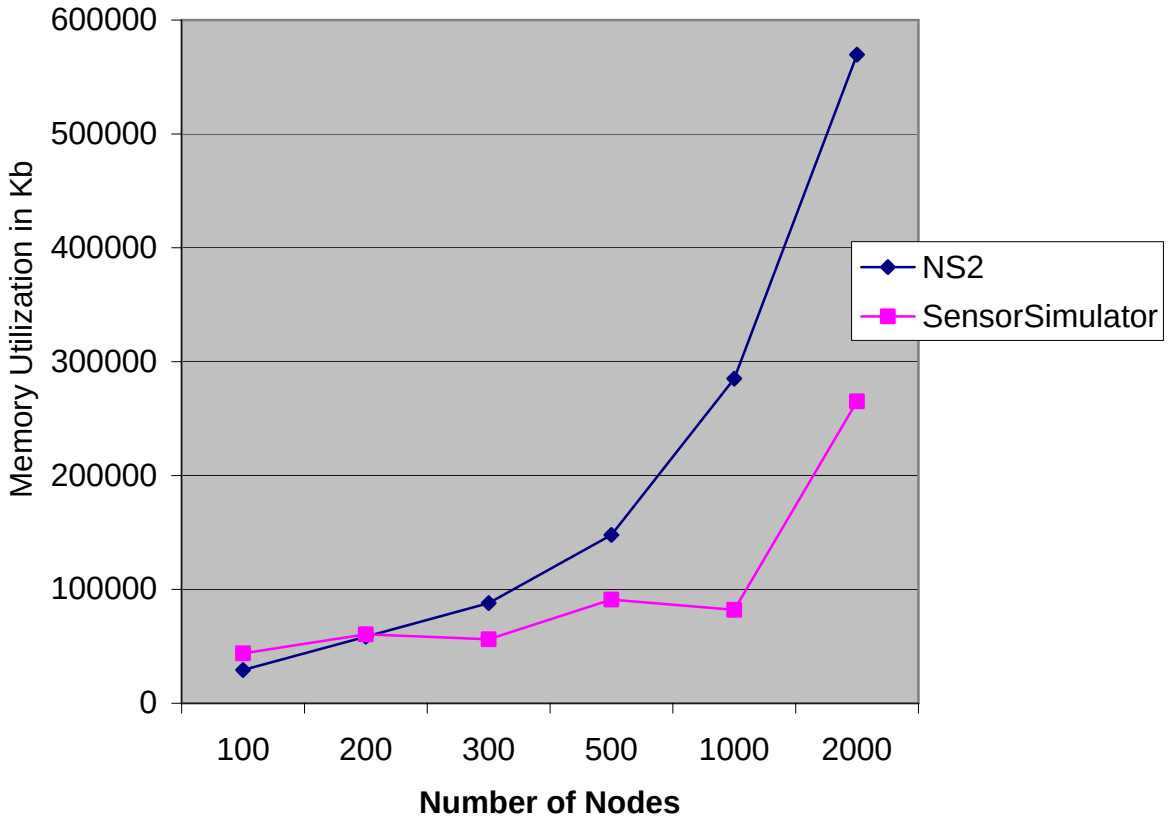


Figure 6.10 Memory utilization comparison during simulation for 100 Queries

6.4.3 Directed Diffusion-GEAR with IEEE 802.11 MAC

This series of experiments use MAC 802.11b at the MAC Layer and test the performance of SensorSimulator Vs NS-2. A simple pass through Physical Layer is considered. The simulation is run for 100, 500, 1000 and 2000 nodes. The nodes in the Sensor Network are deployed randomly in various locations with the network size configured such that the node density is constant. The simulation is setup for 10 nodes generating queries and 10 nodes in the region. The simulation is run for a period of 300 simulation seconds. The simulation is setup in the two simulators such that the nodes

have the same coordinates, deployed in the same network size and the publishers and subscribers are identified during the simulation setup. This is to make sure that the simulation setup is the same in both the simulators and then the performance is tested. The performance of the simulation is as shown in figure 6.11.

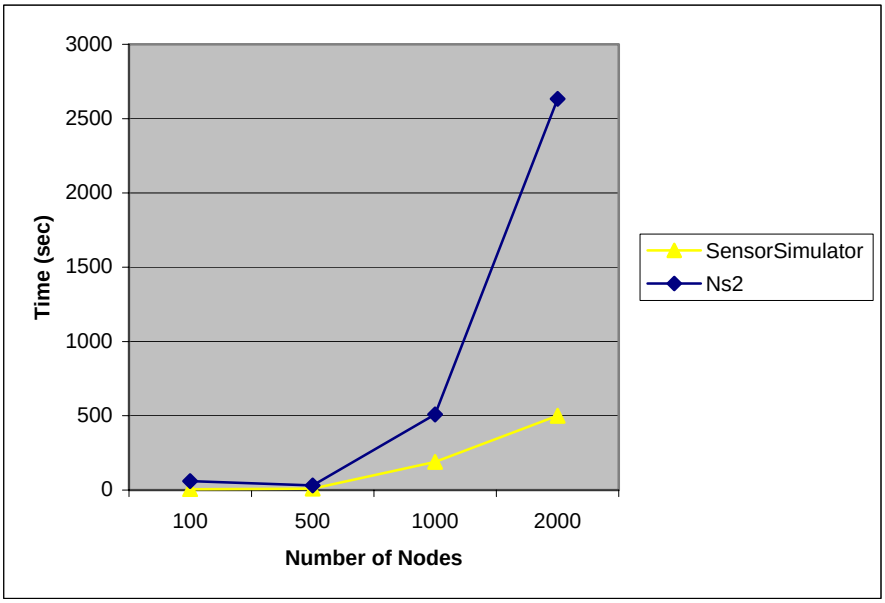


Figure 6.11 Directed Diffusion-GEAR-MAC802.11 execution Time for 10 queries simulated for 300 simulation seconds.

The results show that SensorSimulator takes less time than ns2 even when the numbers of nodes are increased to 2000. Confirming that the query nodes are getting back the appropriate data from the region and the delivery ratio is the similar validates the results.

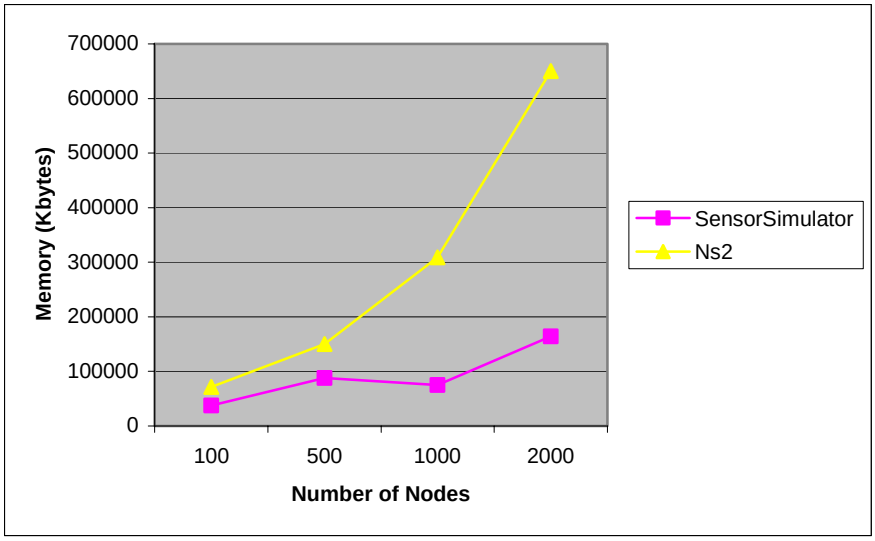


Figure 6.12 Directed Diffusion-GEAR-MAC802.11 memory usage for 10 queries simulated for 300 simulation seconds.

It is also observed that the memory consumption for 10 queries in SensorSimulator is very less compared to NS-2. The memory consumption figure 6.12 shows that the data structures used for the simulation are used in a scalable manner to represent the different classes and the interaction with the framework. It can also be observed that the rate of memory usage increases at a faster rate for NS-2 than for SensorSimulator thus allowing for large simulation setup and ability to simulate larger, scalable networks.

Chapter 7: Conclusion and Future Works

This thesis provides a simulation framework for wireless sensor networks. The framework has definitions for target node; sink nodes, sensor and wireless channel, and power model. Applications can subclass these framework classes and implement their functionality at each of the layers.

Directed Diffusion, GEAR were implemented on this framework to demonstrate the use of the framework. The performance of the simulator was studied with simulations of Directed Diffusion with MAC 802.11.The simulation results have been compared with the results in NS-2 for the same protocols under similar conditions. It was observed that with the SensorSimulator a greater scalability could be achieved. Also SensorSimulator could handle large networks, whereas NS-2 had memory management problems with large networks. Thus SensorSimulator is a good candidate for simulating wireless sensor network protocols. The simulator design and the support provided make it very easy to develop and test protocols very fast and obtain results for large simulations at a reasonable amount of time.

In future we would like to integrate the simulator to sensor devices, to obtain readings and data from the devices and use them to test the behavior of protocols to varying sensor node activities.

References

[1] J. Agre, L. Clare, and S. Sastry. A Taxonomy for Distributed Real-time Control Systems. *Advances in Computers, Ed. M. Zelkowitz*, 49:303–352, 1999.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless Sensor Networks: A Survey. Computer Networks, 38:393–422, 2002.

[3] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. *In Proc. of the ACM International Conference on Mobile Computing and Networking (ACM MobiCom'00), 2000.*

[4] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 1(1): 2–16, February 2003.

[5] J. Heidemann, F. Silva, and D. Estrin. Matching data dissemination algorithms to application requirements. In *Proc. of the ACM Conference on Embedded Networked Sensor Systems (ACM SenSys'03)*, 2003.

[6].NS -2. http://www.isi.edu/nsnam/ns

[7] S. Sastry and S.S. Iyengar. Real-time Sensor-Actuator Networks. *International Journal of Distributed Sensor Networks,* 2004

[8] S. Park, A. Savvides, and M. Srivastava. SensorSim: A simulation framework for Sensor networks. In *Proc. of the ACM international Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2000.

[9] "Information technology - telecommunication and information exchange between systems - local and metropolitan area networks – specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Standard, Tech. Rep.*, 1999

[10] A. Sobeih and J. C. Hou. A simulation framework for sensor networks in J-Sim. Technical Report UIUCDCS-R-2003-2386, Department of Computer Science, University of Illinois at Urbana-Champaign, November 2003.

[11] H.-Y. Tyan and J. C. Hou. *JavaSim*: A component-based compositional network simulation environment. In *Proc. of Western* 23 *Simulation Multiconference, CNDS'01*, January 2001.

[12] H.-Y.Tyan. Design, Realization and Evaluation of a Component-based Compositional Software Architecture for Network Simulation. PhD thesis,

Department of Electrical Engineering, The Ohio State University, 2002.

[13] *OMNeT++, http://www.omnetpp.org/*

[14] D.B. Johnson. The Rice University Monarch Project. Technical Report, Rice University, 2004.

[15] J. Misra. Distributed discrete-event simulation. ACM Computing Surveys, Vol. 18, No. 1, pp. 39-65, March 1986.

[16] OPNET Technolgies, Inc. OPNET Modeler.

[17] S. Sastry and S.S. Iyengar. Sensor Technologies for Future Automation Systems. Sensor Letters, 2(1): 9–17, 2004.

[18] A. Sobieh and J.C. Hou. A simulation framework for sensor networks in J-sim. Technical Report UIUCDCS-R-2003-2386, University of Illinois at Urbana-Champaign, Department of Computer Science, November 2003.

[19] A. Tannenbaum. Computer Networks. Prentice Hall, 2002.

[20] A. Vargas. OMNET++ Discrete Event Simulation System, version 2.3 edition, 2003.

[21] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks, August 2001.

[22] Cariappa Mallanda, Shivakumar Basavaraju, Archit Kulshrestha, Rajgopal Kannan, Arjan Durresi and S. S. Iyengar "Secure Cluster Based Energy Aware Routing for Wireless Sensor Networks," in *Proceedings of International Conference on Wireless Networks ICWN 04* , pp. 461-466.

[23] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: A library for parallel simulation of large-scale wireless networks. In Workshop on Parallel and Distributed Simulation, pages 154–161, 1998.

[24] S. S. Iyengar and R. R. Brooks, “Distributed Sensor Networks” CRC press ‘Inc. Dec 28, 2004 pp 1188

[25] http://www.cs.rpi.edu/~cheng3/sense/

[26] “The SSFNET Project," http://www.ssfnet.org

[27] “QualNet” http://www.scalable-networks.com/”

[28] “Geographical and Energy Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks” Yan Yu, Ramesh Govindan and Deborah Estrin UCLA Computer Science Department *Technical Report UCLA/CSD-TR-01-0023,* May 2001.

Vita

Cariappa Mallanda was born in Bangalore, India , on April 21,1978. He received his bachelor's degree in Computer Science from Kuvempu University, India, in 2000. He is currently a master's student in Computer Science Department at Louisiana State University. As a member of sensor network lab, he has conducted a two-day workshop on sensor networks at Raytheon Company, Dallas. He has worked on embedded system devices such as the Mica2 motes and the TINYOS. His research areas include networks, sensor networks, cryptography and simulations.