ADHOC ROUTING BASED DATA COLLECTION APPLICATION IN WIRELESS SENSOR NETWORKS

by

MALLIKARJUNA RAO PINJALA

B.E, OSMANIA UNIVERSITY, INDIA, 2004

A REPORT

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF COMPUTING AND INFORMATION SCIENCES
COLLEGE OF ENGINEERING

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2010

Approved by:

Major Professor
Dr. Gurdip Singh

Abstract

Ad hoc based routing protocol is a reactive protocol to route messages between mobile nodes. It allows nodes to pass messages through their neighbors to nodes which they cannot directly communicate. It uses Route Request (RREQ) and Route Reply (RREP) messages for communication. Wireless sensor networks consist of tiny sensor motes with capabilities of sensing, computation and wireless communication. This project aims to implement data collector application to collect the temperature data from the set of wireless sensor devices located within a building, which will help in gathering the information by finding the route with minimum number of hops to reach destination and generates low message traffic by not encouraging the duplicate message within the network. Using this application, wireless devices can communicate effectively to provide the network information to the user. This system consists of a mobile wireless sensor device called base station which is connected to a PC to communicate and is the root of the network. It also consists of set of client sensor devices which are present in different parts of the building. This project has been evaluated by determining how well the ad hoc protocol performs by measuring the number of messages and time consumed in learning about the complete topology. This application will eventually find the path with minimum number of hops. Simple Network Management Protocol (SNMP) is also used to monitor the sensor nodes remotely.

This project was developed using nesC and C programming languages with TinyOS and UNIX based operating systems. It has been tested with a sufficient number of motes and evaluated based on the number of messages generated and number of hops travelled for each route request.

Table of Contents

List of Figures

Acknowledgements

First, I would like to thank my Major Professor Dr. Gurdip Singh for his constant help, encouragement and guidance throughout the project.

I would also like to thank Dr. Torben Amtoft and Dr. Mitchell Nielsen for serving in my committee and for their valuable cooperation during the project.

Finally, I wish to thank my parents and my sister and brother-in-law for all their support and encouragement.

Chapter 1 - Introduction

This chapter provides an overview of the report, an introductive description of the concepts, the motivation and objective of the project.

1.1 Wireless Sensor Network

Wireless Sensor Network (WSN) consists of a set of distributed sensor devices which communicate with each other using radio communication. This kind of network is mainly used to monitor physical or environmental attributes such as temperature, pressure, humidity etc [1]. Each device in a sensor network is equipped with a radio transceiver, microcontroller and an energy source, usually a battery. A wireless sensor network consists of set of wireless sensor devices which send and receive packets and also read sensing information from the sensors present on the devices. These networks can also be used to develop a multi hop algorithm to communicate with the nodes which are away by more than one hop.

1.2 Applications

Wireless sensor networks can be used in various applications. The sensors for temperature, light, accelerometer can be used to detect fire, to monitor environmental changes and detect motion or speed of a vehicle respectively. There are many issues that need to be considered when developing an application. These issues can be related to loss of messages, changes in the topology, mobility of the devices, transmission etc. To collect the complete information of the network, some mechanism should be defined to collect data from devices which are more than one hop away as the transmission capabilities of devices in sensor network are limited.

1.3 Motivation

To collect data from the network, an application must build logical topology information on the network. The transmission power capability of the devices is limited, so the application should be capable of collecting the information of multiple hops. That is if the information is collected in such a way that it will have the information which is more than one hop away, then the root of the network or the base station can communicate with all the nodes in the network.

An energy intensive approach should be implemented to reduce the transmission power, latency, network traffic and gather the sensor devices information.

1.4 Problem Statement

This project aims to implement a data collector application to collect temperature data from a set of wireless sensor devices located within a building. This is done by collecting the complete network information using ad hoc routing mechanism on a regular basis. The application will also help in gathering the information using minimum number of hops from the destination to the root of the network, called base station and also it will reduce the number of messages generated in the network by not forwarding the duplicate messages within the network.

The system consists of a mobile wireless sensor device called base station which is connected to a PC. There will be set of sensor devices called clients that are arranged in the network to retrieve the temperature readings. The base station is used to communicate with all these client nodes within a network.

From the user perspective, the user will request for the temperature data from a client node, by entering the client node id on the PC. Then the application running on the PC will take the input and forward the request to the base station. The base station will request for the data from the client nodes within the network. This will be done by forwarding the request messages by the intermediate client nodes and finally reaching the destination. The destination will return the temperature reading present in that location. This temperature reading will then be received by base station displayed on the PC.

Here it is assumed that the network should be dense enough to communicate with all the sensor devices. There will be only one node i.e., the base station which will initiate the algorithm.

The network information is collected by applying the below algorithms. The reasons for using them are also explained:

1.4.1 Message Processing Algorithms

Two algorithms one for each of route request and route reply message processing is defined. The description of these algorithms is explained in chapter 7.

Route Request Message Algorithm:

This algorithm is developed to retrieve the path information for the destination client node. This algorithm helps in reducing the number of messages generated and reaching the destination using minimum number of hops.

Route Reply Message Algorithm:

This algorithm is developed to process the route reply message generated by client and send to the base station. This will help in sending the acknowledgement using a unicast reply message.

1.4.3 Assumptions

The user should be aware and be able to identify all the sensor devices within the network from which the temperature information has to be collected.

1.5 Ad hoc Routing Protocol

This protocol allows the nodes to route messages through their neighbors to nodes which they cannot directly communicate with. This is done by finding the routes along which messages can be passed. This protocol avoids messages getting looped around by not regenerating the same message by the same node more than once. It also finds the path with minimum number of hops to the destination. This protocol also handles changes in the routes periodically and can create alternate routes if the existing route causes any failure.

When a node wants to send a message to another node, which may not be its neighbor it simply broadcasts a Route Request (RREQ) message.

This message contains several fields such as source, destination, hop count and sequence number. When a neighbor receives a Route Request message, if they know the route to the destination, it generates a unicast Route Request message or if it doesn’t know the destination, it generates another broadcast message to the neighbor. If the node itself is the destination, it can send a Route Reply (RREP) message to the source. This message contains the details of all the nodes through which it needs to be sent to reach the source node.

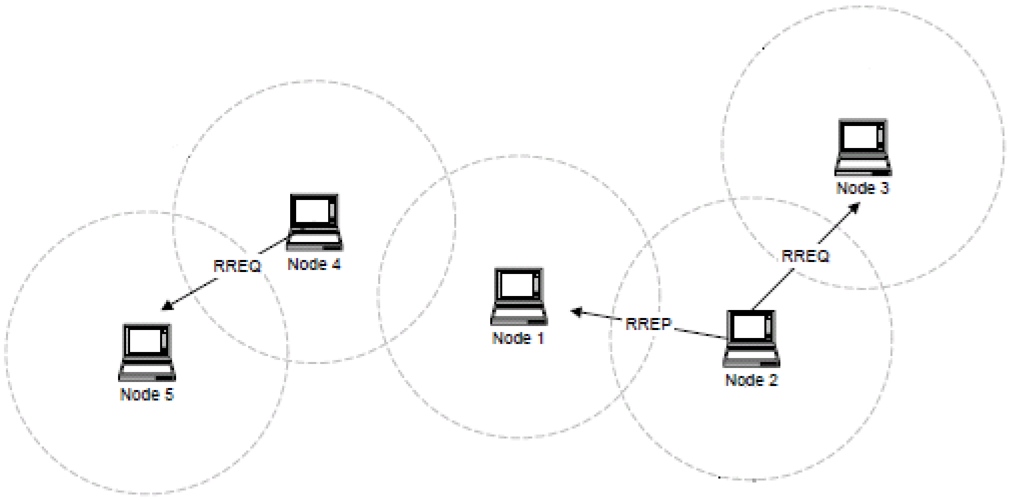

Figure 1-1: Ad hoc Network

Chapter 2 - TinyOS

2.1 Introduction to TinyOS

TinyOS is an open source operating system which is designed for wireless embedded sensor networks [4]. This can be used to deploy component based applications. TinyOS uses a component-based architecture which helps in implementing new applications and enables in rapid innovation, while compacting the size of the code which is the main criteria for the devices used in sensor networks that requires a severe memory constraints. The component library of TinyOS includes network protocols, distributed services, sensor drivers, and data acquisition tools – all of which can be used as-is or be further refined for a custom application. Some of the tools include an interface to generate messages, Message Interface Generator (MIG), an interface to simulate the TinyOS based applications, TOSSIM [5] etc. Many of these tools can be customized and modified to suit many real-world sensor applications.

2.2 NesC

NesC [6] is an extension to the C programming language designed to embody the structuring concepts and execution model of TinyOS. It is primarily intended for embedded systems such as sensor networks. A NesC application consists of one or more components linked together to form an executable [7]. A component provides and uses interfaces. These interfaces are the only point of access to the component and are bi-directional. An interface declares a set of functions called commands that the interface provider must implement and another set of functions called events that the interface user must implement. For a component to call the commands in an interface, it must implement the events of that interface. A single component may use or provide multiple interfaces and multiple instances of the same interface. Interfaces may be provided or used by components. The provided interfaces are intended to represent the functionality that the component provides to its user; the ‘used’ interfaces represent the functionality the component needs to perform its job.

2.3 TelosB Mote

TelosB mote is a low power wireless sensor module and open source platform designed to enable experimentation for the researchers [10]. It has integrated Light, Temperature and Humidity sensors. Data collection and programming is done via USB interface. It is capable of gathering sensor information and communicating with other motes within the wireless sensor network.

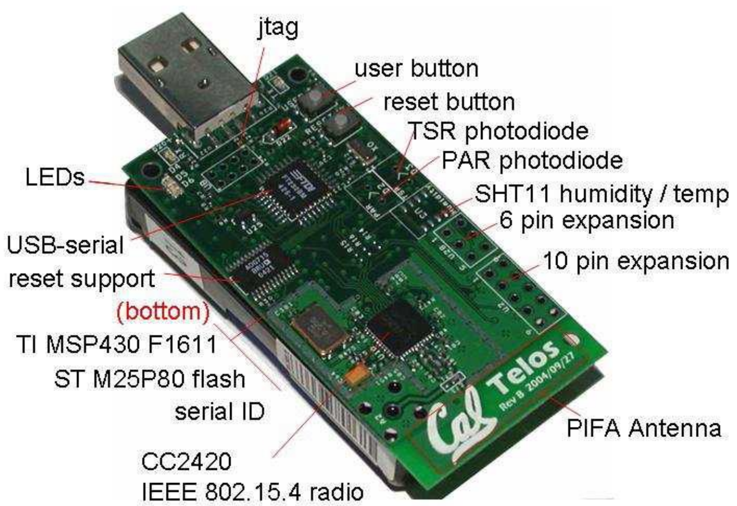

Figure 2-1: TelosB Mote

Image Source [11]

2.3.1 Testing on Motes

A test bed has been set up consisting of a set of boards of 3 feet height and one feet width. Motes are arranged on the board in a grid like structure. The effective communication range is around 75 to 100 meters. So the experiment has been performed by decreasing the transmission power to reduce the communication range between motes. setPower (uint8_t power) command is used to set the communication range which is present in CC2420Control interface. Power value lies between 1 and 31. This project is tested by setting the transmission power to 1.

Chapter 3 - Design Approaches

This chapter describes the two approaches which were considered in solving the problem. Before describing the approaches, the section below describes about each component used for developing this project which will help in understanding the approaches.

3.1 Components

Ad hoc based routing architecture has been extended to implement the data collector application [2] [3]. Each node in the network is replaced with a sensor mote. Here instead of having each mote initiating the Route Request (RREQ), one designated mote called Base Station is used. This is responsible for sending and receiving messages from the client motes. An SNMP interface is used to query the data from the Base station. PC engine is used as a storehouse and interface for the user to request and display the information received from the base station. Below is the detailed description of each component used in data collector application:

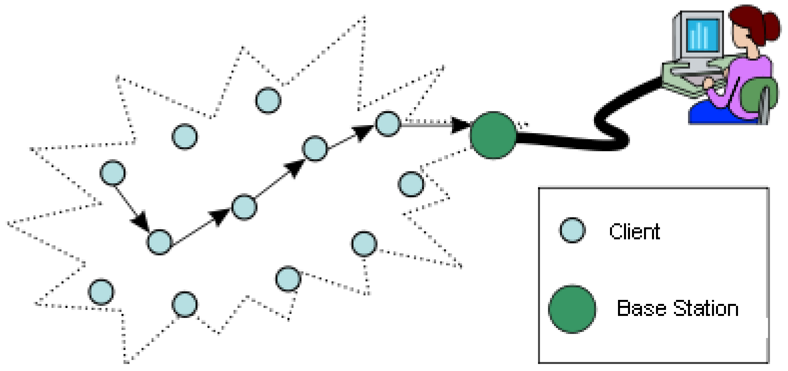

Figure 3-1: Clients and base station setup in WSN

Image Source [12]

The data collector application is divided into four components 1) Base Station 2) Client 3) SNMP Interface 4) PC Engine. Below is the description of each component. The communication and implementation details of these components are described in Chapter 5.

3.1.1 Base Station

This is a wireless sensor mote connected to the computer (PC Engine). It acts as an interface gateway between PC Engine and the set of sensor devices within a building. It uses the radio interface to communicate with the client nodes and uses UART interface to provide serial communication with the PC Engine. It is responsible for forwarding requests to the client nodes generated by PC and receiving responses from the client nodes and forwarding to PC. TelosB mote is used to represent the base station.

3.1.2 Client

This component is responsible for getting the sensing information on a regular timer basis from the sensing device present on the wireless sensor node. Multiple client nodes are kept in different areas within a building. This component is responsible for processing messages either received from neighboring client or base station. The client node constantly collects the neighbor information upon processing the route request and reply messages. It also shares its neighbor information whenever it sends the request and reply message to the neighboring nodes. TelosB motes are used to represent client nodes.

3.1.3 PC Engine

This component is responsible for accepting the queries generated by the user and converting the request to a message format that is understood by the base station. The message is then send to base station through serial communication. The base station will also send the reply messages received from the client node to the PC Engine. The PC Engine constantly gathers and develops the complete knowledge of the sensor nodes present in the building. This information is stored in a file. The PC engine also runs the SNMP daemon which is responsible for any SNMP requests that are sent to this daemon.

3.1.4 SNMP Interface

Simple Network Management Protocol (SNMP) is a widely used protocol for monitoring the health and welfare of network equipment. In this project SNMP is used to monitor the temperature information of all the client nodes within a network remotely. Another PC is used to to run the SNMP software. This PC will communicate with the PC Engine and request the temperature information of client motes. The PC engine will answer the queries related to the

SNMP requests. The PC engine will return all the entries present in the neighbor table. The PC which has an SNMP interface will then the display list of clients with their temperatures. In this way, the SNMP Interface will be used to get the information from a remote machine. Chapter 4 describes about SNMP in more detail.

3.2 Approaches

This section describes the two approaches which are used to implement the data collector application.

3.1.4 Approach 1

The first approach uses the broadcast messages to send data from the base station to the client motes. When the user enters the mote id from which to get the temperature data, then the PC engine will generate a route request message which is sent to the base station. The message structure for the request message is described in Chapter 8. Upon receiving the request message by the base station, base station will broadcast the route request message including its mote id in the path list. The client motes within the vicinity of the base station will receive the message. On receiving the route request message, the client motes will then again broadcast the message to all the client motes by appending the client mode id details in the path list. Appending the client mote id is required in the request message so that when the destination receives the message it can send the unicast message through that path from which it received the message. If it sees its own mote id in the path list then the client will drop the message. When the destination client mote receives the request message, it will retrieve the path information from the request message and generate a new reply message including this path information.

In this approach, the reply message generated by the client mote will be a unicast message instead of broadcast message that is sent to the base station. So, the intermediate number of messages will be reduced while sending the reply message. But the request message that is generated by the base station and the client motes is a broadcast message. The request messages generated will be more as the intermediate client motes will rebroadcast each message again. So, there will be lot of duplicate messages that will be generated. This approach should be tuned so that the duplicate messages can be discarded.

3.1.4 Approach 2

The second approach also uses the broadcast messages to send data from the base station to the client motes. But the broadcast message will be sent only when it doesn’t have any information about the client mote. This approach will use the SN and HC fields present in the message structure (Chapter 8) to reduce the number of message and send the reply message through minimum number of hops to reach the destination.

The user enters the mote id of the client to get the temperature data. The PC engine will generate a route request message with the client mote id and the path information which it retrieves from the neighbor table. The neighbor table contains the information about the client motes and the neighboring node id from which to reach that client. PC engine will then send this constructed message to the base station. Upon receiving the request message by the base station, the base station will determine whether to broadcast or unicast the message based on the details present in the request message. The description for the fields that are set during the packet process is explained in Section 5.1 of Chapter 5. The client motes within the vicinity of the base station will receive the message. When the client mote receives the request message it will see whether the message is already received by the client mote. This will be determined by the use of sequence numbers. Its usage is explained in Section 5.3.1 below of Chapter 5. If it already received the message, then it will update any new neighbor information from the request message and will simply discard the message. With this mechanism, duplicate messages generated will be avoided. Also whenever the client mote or the PC engine receives a request message, they will collect the information of the neighboring motes within the network and stored in a table called neighbor table. The client motes will also remember the number of hops required to reach the destination. This will be determined by the hop count. Its usage is explained in below Section 5.3.2 of Chapter 5. When the destination client mote receives the request message, it will retrieve the path information from the request message and generate a new reply message including this path information.

Chapter 4 - SNMP

4.1 Introduction

SNMP is a network protocol which is used mostly in network management systems to monitor network attached devices [13]. It defines a set of standards for network management that allows management information to be exchanged between SNMP agents and managers. SNMP provides management data in the form of variables on the managed systems, which describe the system configuration. These variables can then be queried by managing applications.

An **SNMP agent** is a software program that implements the SNMP protocol and allows a managed node to provide information to a Network Management Station and accepts commands from the NMS.

The **SNMP manager** is responsible to allow the Network Management Station to send the commands to the managed node also to get the information from the managed node whenever it wants to retrieve the data.

A **Network Management Station (NMS)** are the network devices that run special software to allow it to manage the regular management nodes.

4.2 Purpose

SNMP provides administrators with a network monitoring mechanism to retrieve the data provided by the data collector application remotely. Using SNMP, the temperature information can be retrieved remotely and can use this information to correct network related issues. Because of the simple design of SNMP, it make easy for a user to program the variables that needs to be monitored remotely. It is also easy for the protocol to be updated, so that it can meet the needs of the user.

4.2 Usage in this project

This section describes how SNMP and its components are represented in the data collector application. NET-SNMP is an open source application tool which we can use to implement all the components for this project [14].

A managed device is a network node that implements an SNMP interface which provides access to the node specific information. In the data collector application, the PC engine provides all the network information. This is considered as the managed device.

The SNMP agent is the one which runs the SNMP software module which has all of the management data. Here, the management data is the neighbor table to retrieve the complete network information. In order to create a SNMP module for the data collection application, we need to generate a Management Information Base (MIB) file [15]. The structure of this file is given in Section 8.3 of Chapter 8. From this MIB file, a SNMP MIB module is generated which is described in Section 4.3 of Chapter 4. Once the MIB module is generated, the SNMP daemon should be restarted. The PC engine is responsible for running the SNMP daemon. Now, the network information that is generated by the data collector application can be used to provide the data remotely.

To access the management data of data collector application remotely, a Network Management System (NMS) is used. In this project, the PC that represents the SNMP interface is responsible for providing the Network Management System.

4.3 MIB Module

The Management Information Base (MIB) describes the structure of the management data of a device or an application [15]. It uses a hierarchical namespace structure containing the object identifiers (OIDs). The NET-SNMP daemon (agent) needs to be extended to implement the custom module for the data collector application. MIB modules usually make use of the API provided by the NET-SNMP and are therefore written in C programming language. To implement the MIB module, three files are necessary: MIB definition file, C header file and C implementation file. Writing MIB modules is a long and a cumbersome process. To avoid this, mib2c tool can be used to generate the template code for extending the agent. mib2c tool uses the MIB definition file to produce the two C code files. This produces the template code which needs to be updated with the relevant code to provide the functionality for retrieving the information from NMS.

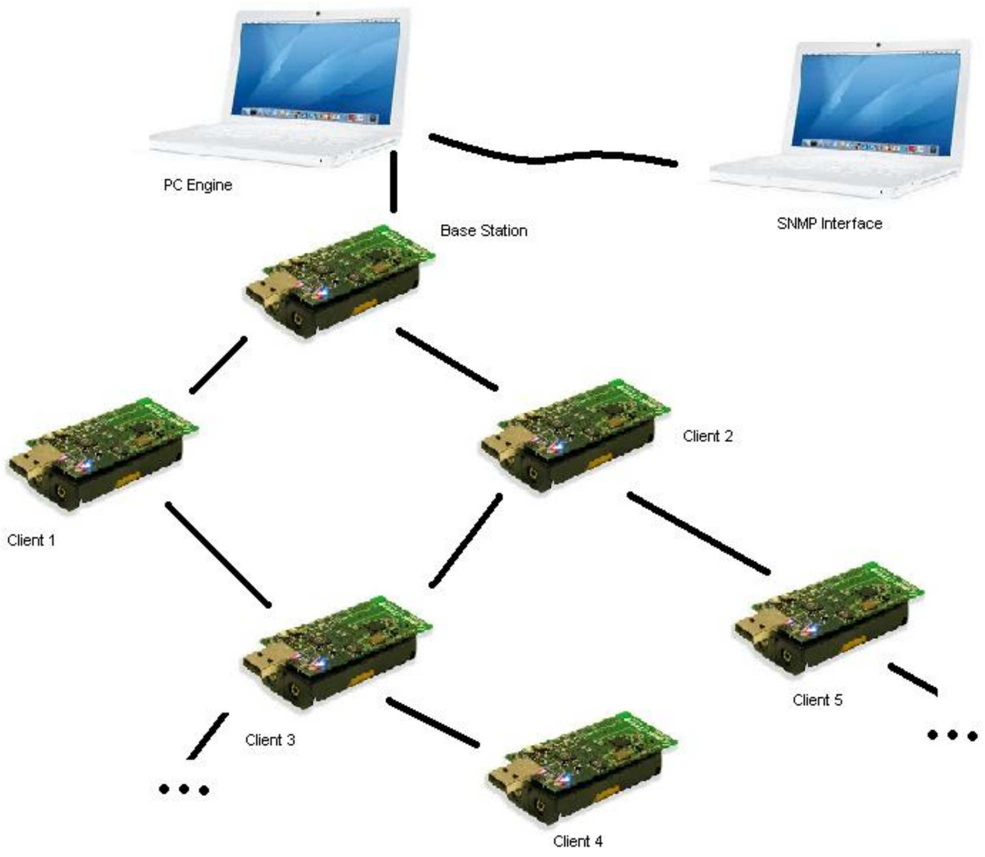

Figure 4-1: Network Setup with components

Chapter 5 - Communication Overview

5.1 Functionality and Packet Flow

This section describes the packet flow and the fields that are set in the request and reply messages during each step of Approach 2.

Client motes in the wireless sensor network are arranged in different locations within a building. These client motes have sensors which can sense the climatic conditions (temperature, humidity and pressure) and retrieve the sensor values. For each sensor mote, there are two interfaces: radio and UART. The radio interface is used to communicate with other sensor motes. UART interface is used for serial communication between a mote and PC. The overall environmental changes within a building can be collected by having a central gateway mote (base station) which communicates with all the sensor motes located in the building. The base station is connected to a computer (PC Engine) to forward the information that is received from the radio interface to the UART interface and vice versa.

The user (at PC Engine) enters the destination client mote id 4 in the example below. Once the destination client id is entered, the PC engine will generate a route request message. The message structure is described in Section 8.1 of Chapter 8. The message id is set to 104 for route request message and destination mote id is set to 4. The hop count is initially set to 0 and will be incremented by the client mote whenever the message is forwarded to the next neighboring mote. The sequence number is controlled by PC engine. This value is initialized to 0 and will be incremented for each new request message generated. The broadcast flag is set to 0, if the destination client entry is present in the neighbor table. Otherwise the broadcast flag is set to 1. The neighbor table entry will be in the form of a tuple <destination Id, next Hop Id, Hop Count>. When the application starts there are no entries present in the neighbor table. So the broadcast flag will be set to 1. Data id is set to 201 to retrieve temperature data. Path count represents the number of motes that has been traversed by this request message so far and Path contains the list of mote ids that has been traversed. The PC engine will set the path count to 1 and the path list contains <1>.

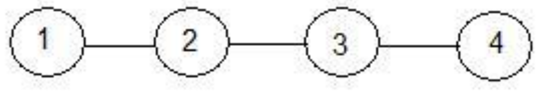

Figure 2-1: Motes

Once the request message is generated it will be sent to the base station through the UART interface to obtain the environmental attribute i.e., temperature of a particular mote within the building. The base station either sends a broadcast message or a unicast message based on the broadcast flag set by the PC engine. This message will be sent through the radio interface.

The neighboring client motes within the vicinity of the base station will receive the message and do similar processing as done by PC engine. Upon receiving the request message from the base station, client mote 2 will increment the hop count from 0 to 1. The sequence number will be unchanged. The broadcast flag will be set to 0, if the entry for the destination mote is present in the neighbor table. It is similar to the processing done in PC engine. The data id will remain unchanged. The path count will be incremented from 1 to 2. And the path list will include the client mote id and will be <1,2>. Client 2 will again broadcast the request message. The base station present within the vicinity of the client 2 will also receive the message, but it will discard the broadcast message as it only processes the received reply messages. Mote 3 will receive the request message and does the similar processing as done by mote 2. Now the path count will be 3 and the path list contains <1,2,3>. Client 3 will again broadcast this message. Client 2 will drop the request message as it already processed the route request message with the same sequence number present in the request message. Client 4 also receives the request message. Since the destination mote id is 4, Client 4 will create a route reply message. Current temperature reading will be filled in the dataval field. The path count and path list fields will be copied from the request message. Client 4 will then send the unicast reply message to the neighbor mote Client 3, which will be retrieved from the path list <1,2,3>. Client 3 will perform similar processing and forwards the reply message to the client 2. Client 2 will again forward the message to the base station with id 1 by seeing the path list <1,2,3> in the reply message. The base station receives this route reply message and forwards it to the PC engine connected through the UART interface.

The PC Engine processes this reply message and stores the sensing data for that particular sensor mote. This way, the user can request any sensor motes (Client) sensing information. The information obtained about the complete network is stored in a data file called NeighborTable in the format:

(<destination>, <nexthop>, <hopcount>, <temperature>)

5.2 Retrieving Data remotely

Once the PC engine receives the client mote information, this information will be stored in a local data file to access remotely. The MIB module for the data collector application is built as described in Section 4.3 of Chapter 4. The SNMP daemon is run on the PC Engine to handle the SNMP requests using the command ‘snmpd.exe’ as below.

```
$ ./snmpd.exe
```

This will listen to all the requests sent to the UDP port 161.

From the remote machine, the neighbor and sensing data can be retrieved by sending the SNMP request message to the daemon using the below command:

```
$ ./snmpwalk.exe –v 1 –c private <PC Engine IP Address> .1.3.6.1.4.1.8072.2.6
```

where **“.1.3.6.1.4.1.8072.2.6”** is the object identifier (OID) to identify the neighbor table entry (See Section 8.3).

This will display the destination, next hop, hop count and temperature values for each entry present in the neighbor table. This way, the complete knowledge of sensing data provided by sensor motes within a building can be viewed remotely.

5.3 Issues

5.3.1 Duplicate Messages

When the request (RREQ) message is generated and broadcasted, the message can be forwarded through different paths before reaching the destination mote. The intermediate motes upon receiving the request message will regenerate the broadcast message. The motes may also receive the same request message which they have already processed. If the mote doesn’t keep track of the messages which they have already processed, then the same message will be processed and regenerated more than once. This will lead to generating heavy message traffic which is not feasible for the sensor motes which have a very low energy resource. To avoid this, sequence number has been added in the message header (See Section 8.1 of Chapter 8). This field will be incremented for each message generated by the base station. Whenever a client mote receives a request message it compares the sequence number present in the message with the sequence number stored locally within the mote. If it receives the message with the sequence number less than or equal to the stored sequence number then it will update any new neighbor information from the request message and will simply discard the message. From example described in Section 5.1 of Chapter 5, Client 2 upon receiving the route request message from Client 3, will update the neighbor table with the Client 3 information and will discard the message as the message is already received from the base station with sequence number 0. This processing will be handled by the route request message algorithm described in Section 7.1 of Chapter 7.

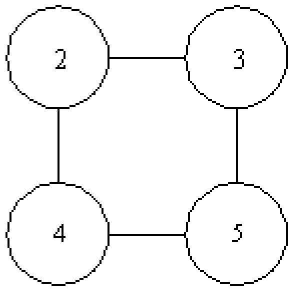

Figure 5-3: Motes arranged in grid

5.3.2 Path with minimum number of hops

Each client mote can receive the information about the other motes within the network from multiple neighboring motes. Each neighbor mote might report a different path to the same destination. If there is no mechanism to find the path with minimum number of hops to reach destination mote, we might end up using a route that traverses through large number of hops. To find the path with minimum number of hops to a destination mote, hop count is added to the message header (See Section 8.1 of Chapter 8). This field is incremented by the client mote whenever the request (RREQ) message is forwarded to next hop. In the Figure 5-2 above, if the base station is present in the vicinity of the Client’s 2 and 3 and requests for the Client 5 information then the Client 3 will report that Client 5 is away by one hop. But the Client 2 might also report that the Client 5 is away by 2 hops via the Client 4. The PC engine will update the neighbor table entry that the destination Client 5 is away by 1 hop via the Client 3 instead of path via 2. In this way, the neighboring mote remembers the number of hops to reach each mote within the network. So, only the mote which has the minimum number of hops away will be stored in the neighbor table. This will be updated whenever it receives the route to a mote with minimum number of hops. This processing will be handled by the route request message algorithm described in Section 7.1 of Chapter 7.

5.3.3 Message Collisions and Loss

When the client mote generates a broadcast message, there is possibility that the neighbor client mote will also generate the broadcast message at the same time. When both the neighboring client motes generate the request message at the same time, there is a possibility that the message will be lost due to collision. To avoid this, a random back off timer value is used. The timer is set whenever the client mote needs to broadcast a request message. When the timer times out, then the broadcast message is sent. With this the client motes can hold the broadcast message for random time interval and send the broadcast message at different time instants. So, the occurrence of collision can be reduced.

The reply message generated by the client might be lost on its way to the base station, and then there is no way for the base station to determine that the message is lost. For this, a timeout interval is kept on the base station so that when the base station doesn't receive the reply message from the client mote, it will regenerate the message after a fixed time interval. If the base station doesn't receive the reply message after three timer intervals then it will assume that the client mote is removed or died and will stop requesting the data from that client mote.

5.3.4 Count to Infinity Problem

This section describes how count to infinity problem [8] is handled in this project. Count to infinity problem arises when there is any node failure and tries to get the alternate route to the failed node. In this project, the route entry for a node will be updated only when it receives the request and reply messages from a neighboring node. This is triggered only when the user requests information from the particular destination through base station. The request message that is sent to the neighboring nodes doesn't include the information about the nodes that are neighbors next to the destination. For example, from Figure 9-1, node 6 will only have a route entry to node 2 with next hop as node 2. Node 6 doesn't store any alternate routes to node 2. If node 2 fails then node 6 will not have any alternate route to node 2 and there is no chance to update the hop count so that it reaches to maximum value. So, if there is any node failure, other nodes will not send any alternate path about the failed node in the request message. Only the nodes which are in active communication in the network will be included in the route request and route reply messages. So, there will be no situation where the client updates the neighbor table with the false information of the node that got failed.

Chapter 6 - Flow Charts

Below is the flowchart for the route request message processing at the base station and at the client.

The user generates the route request at PC by entering the destination mote id. The PC engine performs a look up in the neighbor table for the destination mote and next hop mote id is retrieved. The request is then sent to the base station. On receiving the message, the base station either broadcasts or unicasts the message based on the broadcast flag set in the request message. Below is the flow chart for processing done at basestation:

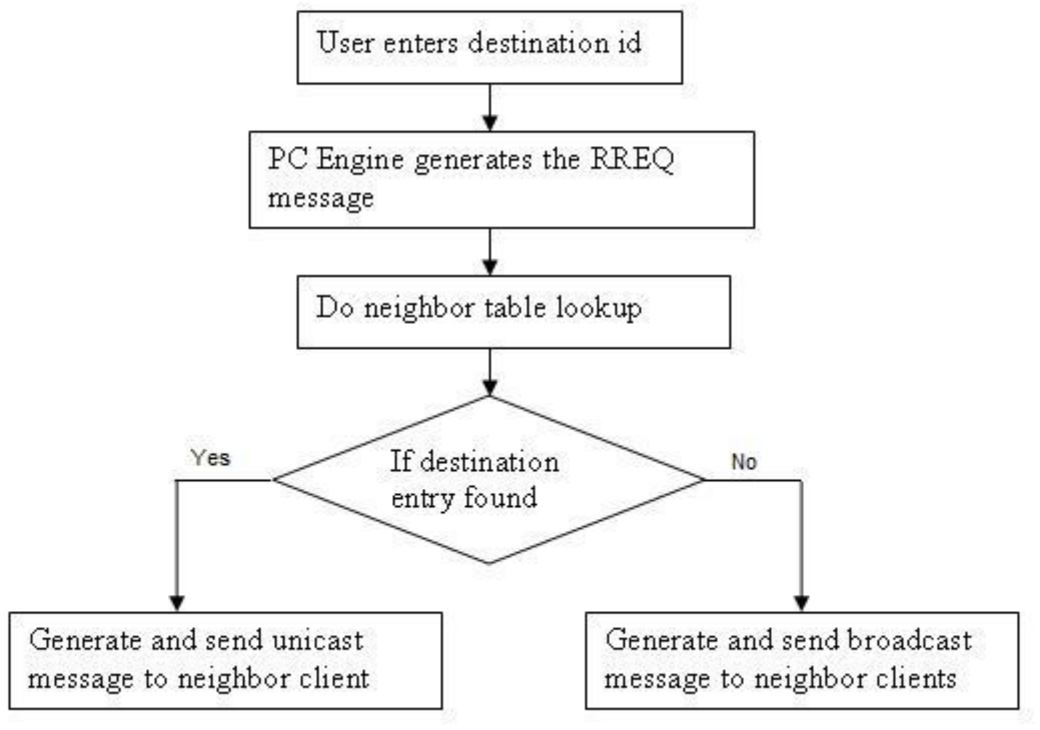
User enters destination id
PC Engine generates the RREQ message
Do neighbor table lookup
If destination entry found
Yes
No
Generate and send unicast message to neighbor client
Generate and send broadcast message to neighbor clients

Figure 6-1: Flow chart at base station

The client processes the route request message received from another client or the base station. On receiving the route request message, the sequence number in the message is compared with the sequence number stored locally. If the received sequence number is less than or equal to the stored sequence number then the client simply drops the message. Next the client checks the hop count for each mote in the path field present in the route request message and compares with that in the neighbor table present locally in the client. If there is any better route observed then the neighbor table is updated. Next, the destination mote id is compared with the client's mote id. If both are same then the route reply message is constructed by copying the required fields from route request message to the reply message. The route reply message is forwarded to the next hop to send to the base station. Otherwise, if the request message is destined for other mote, the client performs a neighbor table lookup and obtains next hop mote id for the destination mote. If the entry is present, the client generates a unicast route request message and forwards to next hop mote, Otherwise the client generates and sends a broadcast route request message. Below is the flow chart for processing done at the client side:

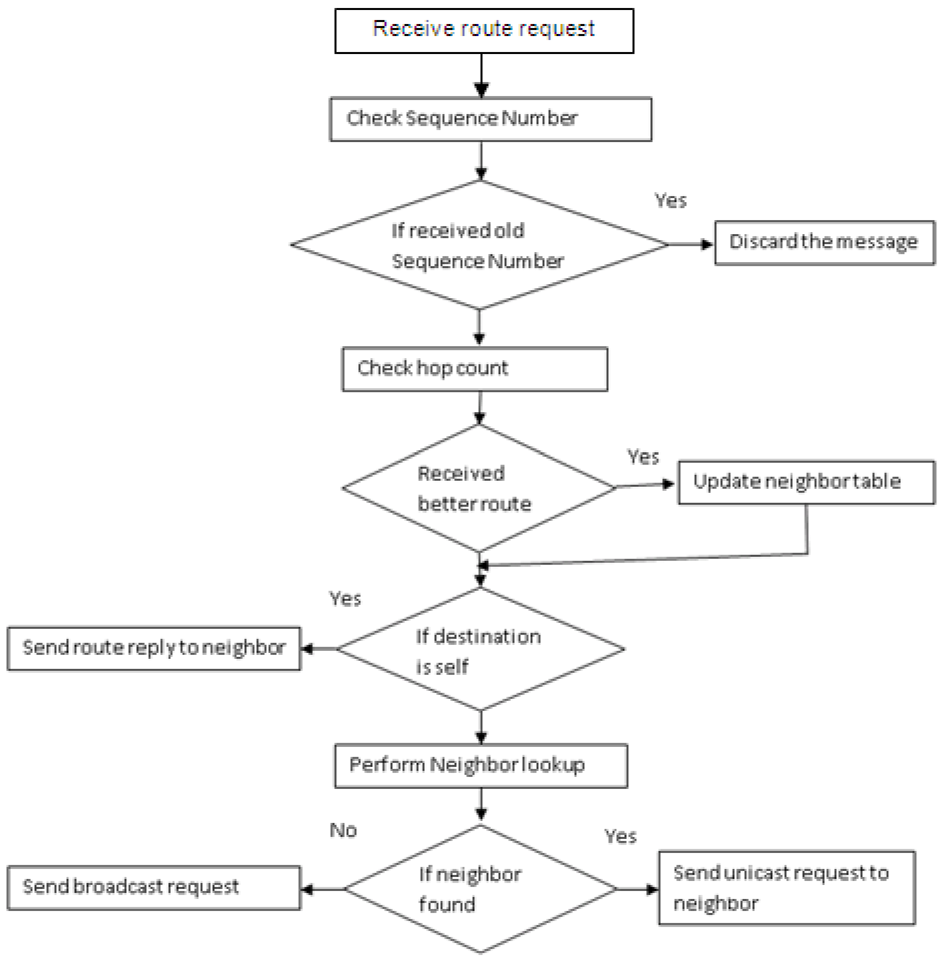

Figure 6-2: Flow chart at client

Chapter 7 - Algorithms

The following algorithms are used to implement Data Collector Application. These are implemented in nesC, an extension to the C programming language.

7.1 Route Request Message Algorithm

The algorithm to process route request message is implemented in nesC language. This algorithm will be run on both the client and the base station. This will be used by the receive () routine present in both the base station and the client. The following are the steps that are involved with this algorithm:

Algorithm Steps: (At PC connected to Base station)

1. User enters the destination client id to retrieve the temperature reading from a particular location. PC engine receives the input.
2. PC will look for the client mote id in the neighbor table to see if the entry already exists.
3. If entry is present in the neighbor table
    a. Set the broadcast flag to zero and next hop id is set to the next hop field present in the neighbor table for the destination mote. This request message is send to the base station mote.
    b. Otherwise, If entry is not present in the neighbor table, the set the broadcast flag to 1 and send to the base station.
4. Base station receives the route request message from PC through UART interface. If broadcast flag is set to 1, then the message is sent as a broadcast. If the broadcast flag is set to 0, then the message is sent to the neighboring client mote which is away by one hop.
5. If route request message is received by the base station
    a. Drop the message as the base station is responsible for only generating the route request message but not receiving and processing the request messages.

Algorithm Steps: (At Client)

1. Receive the request message from the neighboring client or from the base station, check the path list to make sure that the request message is not again received which it already processed. Traverse through the path list to look for the client mote id.
2. If the message contains the id of this client
    a. The request message is generated by itself and again received by this client mote through a different path. Then update the neighbor table with any new entries and discard the message.
3. If the destination mote Id is itself, get the next hop Id from the path list that is present in the request message received.
    a. Generate a route reply message and copy the path and path list fields from the request message to the reply message. Forward the message to next hop id to send it to the base station.
4. If the destination id is set to a different mote id, then
    a. Compare the sequence number that is stored in the client mote with the one that is present in the received request message. If the received sequence number is less than the existing sequence number then the client received an old message.
        i. The path list might contain the mote information that is not available with the client mote. Update this path information in the neighbor table. Once the neighbor table is updated the request message is discarded.
    b. If the client received message with the sequence number greater than the existing sequence number then
        ii. A neighbor table lookup is performed to look for the next hop details for the destination mote
5. If a next hop entry is present for the destination mote then
    a. Unicast route request message is generated. This unicast message is sent to the next hop mote id to forward it to destination mote.
6. If next hop entry is not present for the destination mote then

        a. A new route request message is generated. The client's information is included in the path list and this message is broadcasted in order to send to the destination mote.

7.2 Route Reply Message Algorithm

This algorithm is used to process route reply message. It is implemented in nesC language. This algorithm will be run on both the client and the base station. The pseudo code for the algorithm is as follows:

Algorithm Steps: (At PC connected to the Base station)

1. Receive the route reply message from the base station received through UART interface. Get the mote id from where the reply message is received.
2. Retrieve the path information from the path field present in the reply message.
3. Add new entries that are obtained from the path list to the neighbor table. If there are any entries with less hop count to a particular mote, then update the hop count and the next hop details for those entries in the neighbor table.
4. Print the temperature for the received mote id.

Algorithm Steps: (At Client)

1. Receive the route reply message from the neighboring client through radio interface. Retrieve the path list from the reply message.
2. If there are any new client mote entries present in the path list then
    a. Add the new entries into the neighbor table maintained by the client mote. Otherwise, If there are any entries with a minimum hop count to a particular destination then
        i. Retrieve the next hop id from the path list and update the hop count and next hop id in the neighbor table.
3. To forward the reply message towards the base station, retrieve the next hop information from the path list and generate a new route reply message. The fields for the new reply message are filled from the reply message that was received from its upstream neighbor.

```
4. Retrieve the next hop id from the path list and send the route reply message to the
   next hop id.
```

7.3 Neighbor Table Update Algorithm

The algorithm to update the neighbor table when the route request and reply message is received is implemented in nesC language. This algorithm will be run on both the client and the base station. The pseudo code for the algorithm is as follows:

Algorithm Steps:

```
1. Upon receiving the route request or route reply message, retrieve the path list from
   the message.
2. Retrieve the first client mote entry present in the path list
    a. If the client mote entry is non-zero value, get the first entry of the neighbor
       table.
        i. If the destination id present in the neighbor table entry is non zero then
            1. Compare the client mote entry in path list with the destination
               id the neighbor table. If both are equal then
                a. Compare the hop count in neighbor table entry with the
                   one in the path list. If it is greater than the received hop
                   count, then the neighbor table entry is updated with the
                   latest hop count and the next hop entry.
        ii. If the neighbor entry in the neighbor table is zero then
            1. The received mote id present in the path list is a non existing
               entry. Add a new entry in the neighbor table. Set the hop count
               and the next hop id.
    b. Repeat step a by retrieving next entries in the neighbor table until there are no
       entries that needs to be traversed.
3. Repeat step 2 by retrieving next client mote entry in the path list until the end of the
   list is reached.
```

7.4 Neighbor Table Lookup Algorithm

Algorithm Steps:

1. Get the client mote id for which to retrieve the next hop information.
    a. If the client mote id is non-zero value, get the first entry of the neighbor table.
        i. If the destination id present in neighbor table is non zero then
            1. Compare the client mote id with the destination id the neighbor table. If both are equal then
                a. Retrieve the next hop id field from the neighbor table If the next hop id is non-zero then return the next hop id.
        ii. If the neighbor entry in the neighbor table is zero then
            1. The client mote id is not present in the neighbor table. Then the local flag fount is set to zero and returned.
    b. Repeat step a by retrieving next entries in the neighbor table until there are no entries that needs to be traversed.

Chapter 8 - Implementation

8.1 Message Structure

Below are the message formats which will be used to send the route request message and route reply messages

Route Request (RREQ)

MSGID	DSTID	HC	SN	BCAST	DATAID	PC	PATH
(1)	(1)	(1)	(1)	(1)	(1)	(1)	(20)

Figure 8-8-1: Message Structure for route request message

Route Request (RREP)

MSGID	DSTID	DATAID	DATAVAL	PC	PATH
(1)	(1)	(1)	(1)	(1)	(20)

Figure 8-2: Message Structure for route request message

1 MSGID – ID of the message

2 DSTID – Destination Mote ID

3 HC – Hop Count from the base station, to track the best path

4 SN – Sequence Number generated by the base station, to track duplicate messages

5 BCAST – Flag to determine broadcast or unicast message

6 DATAID – ID to retrieve the value, currently has only one value.

7 PATHCOUNT – Number of motes from the base station to current mote.

8 PATH – List of mote IDs present between the base station and current mote.

Field in braces represents the length of the field in bytes.

8.2 Data Structures

The following message structures are used to generate and process route request and route reply messages at the base station and the client motes.

```
typedef nx_struct route_req_msg
{
  nx_uint8_t  msgId;
  nx_uint8_t  dstId;
  nx_uint8_t  genId;
  nx_uint16_t hopCount;
  nx_uint16_t seqNum;
  nx_uint8_t  bcastFlag;
  nx_uint8_t  dataId;
  nx_uint8_t  pathCount;
  nx_uint8_t  path [MAX_HOPS];

} __attribute__((__packed__)) route_req_msg_t;

typedef nx_struct route_rep_msg
{
  nx_uint8_t  msgId;
  nx_uint8_t  dstId;
  nx_uint8_t  genId;
  nx_uint8_t  dataId;
  nx_uint16_t dataVal;
  nx_uint8_t  pathCount;
  nx_uint8_t  path [MAX_HOPS];

} __attribute__((__packed__)) route_rep_msg_t;
```

Message IDs and Data IDs constants:

```
#define MSG_ID_ROUTE_REQ    104
#define MSG_ID_ROUTE_REP    105
#define DATA_ID_TEMP        201
```

8.3 MIB File

The following MIB has been implemented. mib2c tool is used to generate the C files from the below MIB file. These C files are integrated to net-snmp code and compiled to generate a new snmp daemon (snmpd). This daemon is then run to perform get, set and walk operations remotely.

```
netSnmpTinyOS MODULE-IDENTITY
   DESCRIPTION      "Corrected notification example definitions"
   ::= { netSnmp 4 }

netSnmpTinyOSTables        OBJECT IDENTIFIER ::= { netSnmpTinyOS 2 }

netSnmpNeighborTable OBJECT-TYPE
   SYNTAX           SEQUENCE OF NetSnmpNeighborEntry
   MAX-ACCESS       not-accessible
   STATUS           current
   DESCRIPTION       "This table contains the details of neighbor table entry"
   ::= { netSnmpTinyOSTables 1 }

netSnmpNeighborEntry OBJECT-TYPE
   SYNTAX           NetSnmpNeighborEntry
   MAX-ACCESS       not-accessible
   STATUS           current
   DESCRIPTION      "A row describing a given neighbor table entry"
   INDEX   { dstId }
   ::= {netSnmpNeighborTable 1 }

NetSnmpNeighborEntry ::= SEQUENCE {
        dstId          OCTET STRING,
        nxtHop         OCTET STRING,
        hopCount       OCTET STRING,
        temperature     OCTET STRING

}
dstId   OBJECT-TYPE
   SYNTAX              Integer16
   MAX-ACCESS      read-only
   STATUS              current
   DESCRIPTION     "The destination id of neighbor entry."
   ::= { netSnmpNeighborEntry 1 }
nxtHop         OBJECT-TYPE
   SYNTAX              Integer16
   MAX-ACCESS      read-only
   STATUS              current
   DESCRIPTION     "The next hop id of neighbor entry."
   ::= { netSnmpNeighborEntry 2 }

hopCount       OBJECT-TYPE
   SYNTAX              Integer8
   MAX-ACCESS      read-only
   STATUS              current
   DESCRIPTION     "The hop count for neighbor entry."
   ::= { netSnmpNeighborEntry 3 }

temperature       OBJECT-TYPE
   SYNTAX              Integer16
   MAX-ACCESS      read-only
   STATUS              current
   DESCRIPTION     "The temperature reading for neighbor entry."
   ::= { netSnmpNeighborEntry 4 }
```

8.4 Timers

8.4.1 Table Cleanup Timer

An entry in the neighbor table is added whenever a new neighbor entry is retrieved from the request and reply messages. To have up to date entries in the neighbor table, the old entries needs to be cleaned up on a regular timer basis. For a test bed of sixteen motes, the clean up timer is set to 33 seconds. This is because the base station will request the data from each client mote with a maximum interval of two seconds. So it will take 32 seconds to request from all the client motes. And also the back off timer value will be added when broadcast messages are sent. With a threshold range of 10ms to 160ms, an average value of 90ms is considered as the back off timer value. By considering this value, the clean up timer is set to 33 sec. The Timer component in nesC language is used to set the timer in the client motes. And on the PC connected to the base station, a POSIX API call is used to set the timer:

```
ret = select(maxfd + 1, &rfds, NULL, NULL, &cleanUpTimerValue);
    if (ret == 0) /* Timeout has occurred */
    {
            neighborTableCleanup (); /* Remove each entry in neighbor table */
    }
```

8.4.2 Back off Timer

This timer is used to hold the broadcast message before sending the message. It is used to avoid collisions between neighboring client motes. The threshold value for this timer is a random value in the range from 10 milliseconds to 160 milliseconds for a test bed of 16 motes. The Timer component in nesC language is used to implement this timer on the client motes.

8.5 Pseudo Code

Receive.receive code is triggered whenever a message is received by the radio. And message is send using send () routine. Also the timer component fires event to read from ADC.
Below is the pseudo code for processing received messages from neighbor (either from the base station or the client)

```
/* receive from the Neighboring mote, could be base station */
```

```
event message_t* Receive.receive (message_t* bufPtr, void* payload,
                         uint8_t len)
{
        route_req_msg_t *   pRouteReqMsg;
        /* Get the first byte */
        uint8_t          msgId = * (nx_uint8_t *) payload;
        /* If message is route request */
        if (msgId == MSG_ID_ROUTE_REQ)
        {
                message_t        routeRepPkt;
                route_rep_msg_t * pRouteRepMsg;

                /* If the message is route, get the route message from the second byte */
                pRouteReqMsg = (route_req_msg_t *) payload;

                /* If message looped around and came back, discard the message */
                if (findSelfId (pRouteReqMsg))
                        /* Do Nothing, Discard the Message */
                        loopAboutToTrigger = 1; /* true */
                else
                        call NeighborTableUpdate (pRouteReqMsg);

                /* received an older (less seq num)/duplicate (same seq num) message */
                if (pRouteReqMsg->seqNum <= latestSN)
                        /* Do Nothing, Discard the message */
                        isDiscard = 1; /* true */

                /* If the destination is self */
                if (pRouteReqMsg->dstId == TOS_NODE_ID)
                {
                        if (isDiscard == 0 && loopAboutToTrigger == 0)
```

```
{
      /* Get the NextHopId, which sits as last entry in path list indexed by path count */
      nextHopId = pRouteReqMsg->path [pRouteReqMsg->pathCount - 1];
      pRouteRepMsg = (route_rep_msg_t *)(call Packet.getPayload(
                                     &routeRepPkt, NULL));

     /* create a unicast route reply message to the base station and send
        to NextHop */
     pRouteRepMsg->msgId = MSG_ID_ROUTE_REP;
     pRouteRepMsg->dstId = BASE_STATION_ADDR;
     pRouteRepMsg->dataId = DATA_ID_TEMP;
     pRouteRepMsg->dataVal = temperature;
     pRouteRepMsg->pathCount = pRouteReqMsg->pathCount + 1;

     for (i = 0; i < pRouteReqMsg->pathCount; i++)
       pRouteRepMsg->path[i] = pRouteReqMsg->path[i];

     pRouteRepMsg->path[i] = TOS_NODE_ID;
     alreadySent = 1; /* true */

     call AMPacket.setDestination (&routeRepPkt, 2);
     pRouteRepMsg->genId = TOS_NODE_ID;
     call CC2420Packet.setPower (&routeRepPkt, 2);
     call UartSend.send [id](addr, &routeRepPkt, sizeof (route_rep_msg_t));
     latestSN = pRouteReqMsg->seqNum;
     if (call AMSend.send (nextHopId, &routeRepPkt, sizeof (route_rep_msg_t))
                        == FAIL)
     {
       busy = TRUE;
     }
    }
```

```
}

/* If message is not discarded */
if (alreadySent == 0 && isDiscard == 0 && loopAboutToTrigger == 0)
{
  latestSN = pRouteReqMsg->seqNum;
  /* Update the Neighbor Table */
  neighborTableUpdate (pRouteReqMsg);
  /* generate a new route request */
  pRouteReqMsg_2 = (route_req_msg_t *) (call Packet.getPayload
                                        (&pkt, NULL));
  pRouteReqMsg_2->msgId = MSG_ID_ROUTE_REQ;
  pRouteReqMsg_2->dstId = pRouteReqMsg->dstId;
  pRouteReqMsg_2->seqNum = pRouteReqMsg->seqNum;
  pRouteReqMsg_2->hopCount = pRouteReqMsg->hopCount + 1;
  pRouteReqMsg_2->pathCount = pRouteReqMsg->pathCount + 1;
  for (i = 0; i < pRouteReqMsg->pathCount; i++)
    pRouteReqMsg_2->path[i] = pRouteReqMsg->path[i];

  pRouteReqMsg_2->path[i] = TOS_NODE_ID;
  pRouteReqMsg_2->genId = TOS_NODE_ID;
  call CC2420Packet.setPower (&pkt, 2);
  call UartSend.send [id](addr, &pkt, sizeof (route_req_msg_t));
  if (found == 0)
  {
    pRouteReqMsg_2->bcastFlag = 1;
    if (call AMSend.send (0xffff, &pkt, sizeof (route_req_msg_t))
                         == FAIL)
      busy = TRUE;
  }
  else
```

```
    {
      pRouteReqMsg_2->bcastFlag = 0;
      if (call AMSend.send (nextHopId, &pkt, sizeof (route_req_msg_t))
                          == FAIL)
        busy = TRUE;
    }
  }
  /* Reset Flags */
  alreadySent = 0;
  isDiscard = 0;
  loopAboutToTrigger = 0;
}
else if (msgId == MSG_ID_ROUTE_REP)
{
  route_rep_msg_t * pRouteRepMsg_R;
  message_t         routeRepPkt;
  route_rep_msg_t * pRouteRepMsg;
  pRouteRepMsg_R = (route_rep_msg_t *) payload;
  /* Update the neighbor table */
  neighborTableUpdate (pRouteRepMsg);
  pRouteRepMsg = (route_rep_msg_t *)(call Packet.getPayload(
                                        &routeRepPkt, NULL));
  /* Lookup next Hop towards the base station */
  /* create a unicast route reply message to the base station and send
     to NextHop */
  pRouteRepMsg->msgId = MSG_ID_ROUTE_REP;
  pRouteRepMsg->dstId = pRouteRepMsg_R->dstId;
  pRouteRepMsg->dataId = pRouteRepMsg_R->dataId;
  pRouteRepMsg->dataVal = pRouteRepMsg_R->dataVal;
  pRouteRepMsg->pathCount = pRouteRepMsg_R->pathCount;
  nextHopId = pRouteRepMsg_R->path [loc - 1];
```

```
    for (i = 0; i < MAX_HOPS; i++)
      pRouteRepMsg->path[i] = pRouteRepMsg_R->path[i];

    pRouteRepMsg->genId = TOS_NODE_ID;
    call CC2420Packet.setPower (&routeRepPkt, 1);
    call UartSend.send [id](addr, &routeRepPkt, sizeof (route_rep_msg_t));
    if (call AMSend.send (nextHopId, &routeRepPkt, sizeof (route_rep_msg_t))
                        == FAIL)
      busy = TRUE;
  }
  return bufPtr;
}
```

Chapter 9 - Test and Analysis

Different tests have been performed to analyze the performance of the application with different topology sizes. These tests show the results based on the number of messages generated and the average hop count to reach a destination mote. This is compared with the broadcast approach. From the test results it has been observed that there is a performance improvement with the second approach. The tests are described as follows

9.1 Impact of number of motes on messages

This test case is used to analyze the total number of messages that are generated in collecting the environmental information from all the client motes. The test is run so that the base station will continuously requests for the temperature data from the client motes one after the other. If the base station doesn't receive the information from some of the motes, the request messages are sent for the remaining client motes in the next iteration. Once the base station receives the complete information from all the client motes, the test run is stopped. Now the total messages generated to collect the complete network is calculated. This test case helps in understanding how many messages are required to collect the data from the client motes.

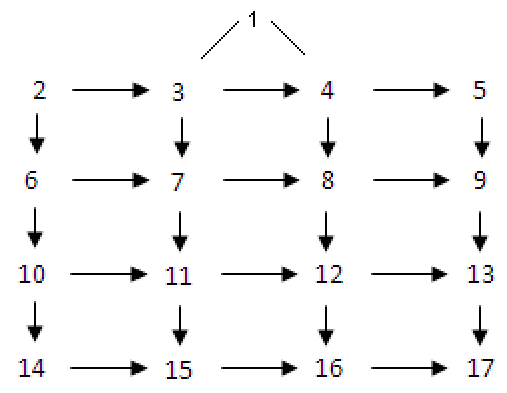

Figure 9-1: Motes arranged in grid structure

The client motes are arranged in a grid like structure as above with ids from 2 to 17. The base station with id 1 requests the client motes 2 through 17. The intermediate client motes which are on the path to the destination will also generate and forward these messages to neighboring motes. The messages generated by each client mote are aggregated to get the total number of messages generated. Once the neighbor information is gradually collected, the number of broadcast messages generated will be reduced and will be forwarded as unicast messages.

Here, there is a possibility that a strong signal messages might be received which are away by more than one hop and also the weak signal message might be dropped.

The x axis shows the different topology sizes which was used to compare the number of messages generated. The y axis represents the total number of messages generated and sent by all the motes. The <blue> and <red> line represents the graph for the number of messages with the Approach 1 and Approach 2 respectively.

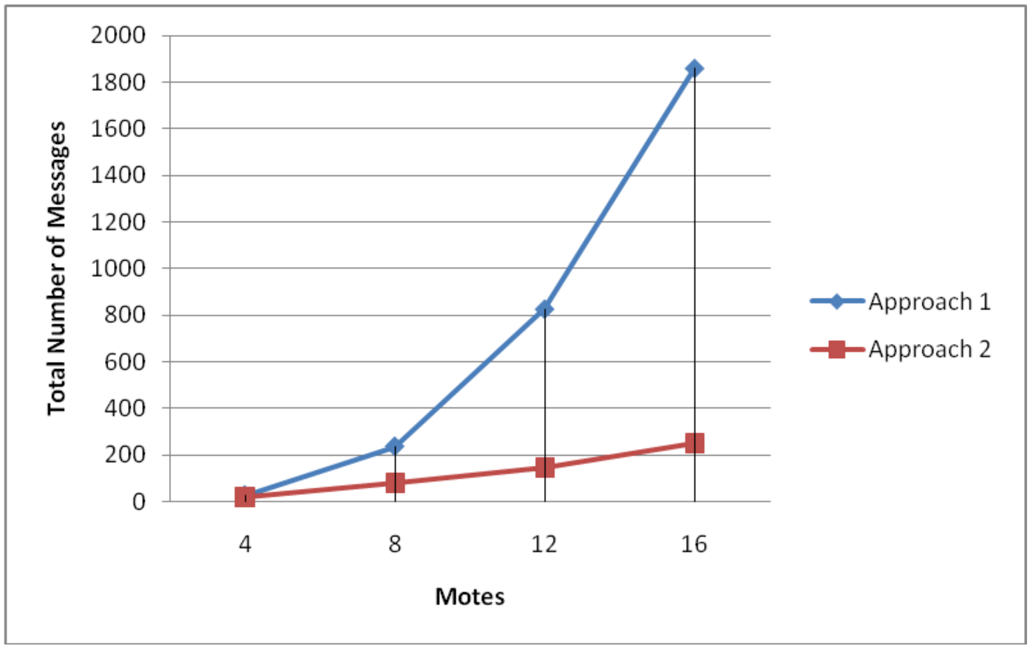

Figure 9-2: Messages Vs Motes

In the first approach, the broadcast messages will be sent for each request and these messages will be regenerated during each hop. Messages will be discarded by the client mote when it receives the message which it had already processed. The request message that traversed through different path will still be regenerated. Here, there is no control to avoid duplicate messages and the number of messages generated will be more. The second approach avoids the duplicate messages using sequence number and reduces the number of messages generated. From the above graph, it can be seen that the total number of messages generated by Approach 2 is hugely reduced when compared with the Approach 1. The difference in number of messages is more when topology size is increased. Clearly from the graph, it can be observed that the Approach 2 performs better than Approach 1 in terms of total number of messages generated.

9.2 Iterations Vs Messages

This test case is used to analyze the number of iterations it took to collect the information from all the client motes. This test is run with motes arranged in grid topology as shown in Figure 9-1. The test is repeated for five times and an average value of the number of iterations is obtained. The x axis shows the different topology sizes which are used to get the total number of iterations it took for building the complete network. The y axis represents the average number of iterations. The <blue> and <red> line represents the graph for the number of iterations with the Approach 1 and Approach 2 respectively.

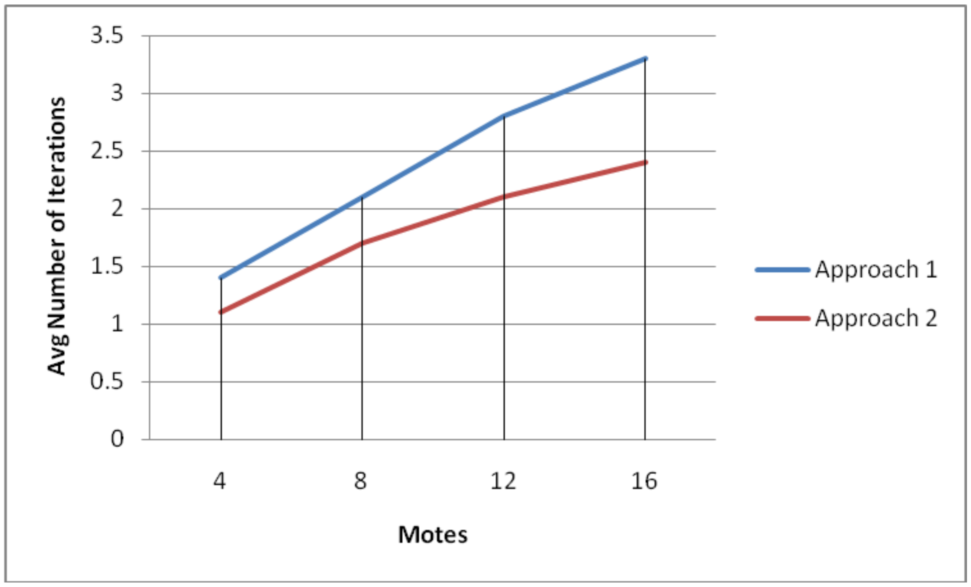

Figure 9-3: Iterations Vs Motes

From the above graph it can be observed that the Approach 2 performs better than Approach 1 in terms of average number of iterations. The number of iterations increases slowly with increase in the number of motes. This is because as the number of motes increases there are chances of collision and weak signal scenarios which might lead to losing the messages. Those messages are requested in the next iteration. Both the approaches can be compared and seen that the number of iterations in Approach 2 is lesser than the Approach 1. This is because, in broadcast approach, there will be more number of messages that gets generated for each request and there will be message loss which leads to more number of iterations. But in Approach 2, there will be less number of messages generated with the use of sequence number and hop count and it requires less number of iterations to collect the information from all the motes.

9.3 Hop Count Vs Motes

This test case is used to analyze the average number of hops reported for all the client motes in the network. The x axis shows the different topology sizes which are used to get the average number of hops reported for building the complete network. The client motes are arranged in a grid like structure as shown in Figure 9-1. The y axis represents the average hop count in each topology size. The <blue> and <red> line represents the graph for the average hop count with the Approach 1 and Approach 2 respectively.

The test case is run similar to the one described in Section 9.1 of Chapter 9. For a topology size of 16 motes, the test is run so that the base station will continuously requests for the data from the entire client motes one after the other. PC engine will retrieve the hop count information from the received reply message. Once the base station receives the complete information from all the client motes, the test run is stopped. If the base station doesn't receive the information from some of the motes, the request messages are sent for the remaining client motes in the next iteration. Once the complete network information is collected, the average sum of the reported hop counts is taken and compared with the broadcast approach. In the broadcast approach, client mote generates multiple reply messages for the same request following different path to the base station. So, the average sum of these paths is taken as the hop count for each destination mote. Once all the hop counts are obtained, the average hop count is calculated.

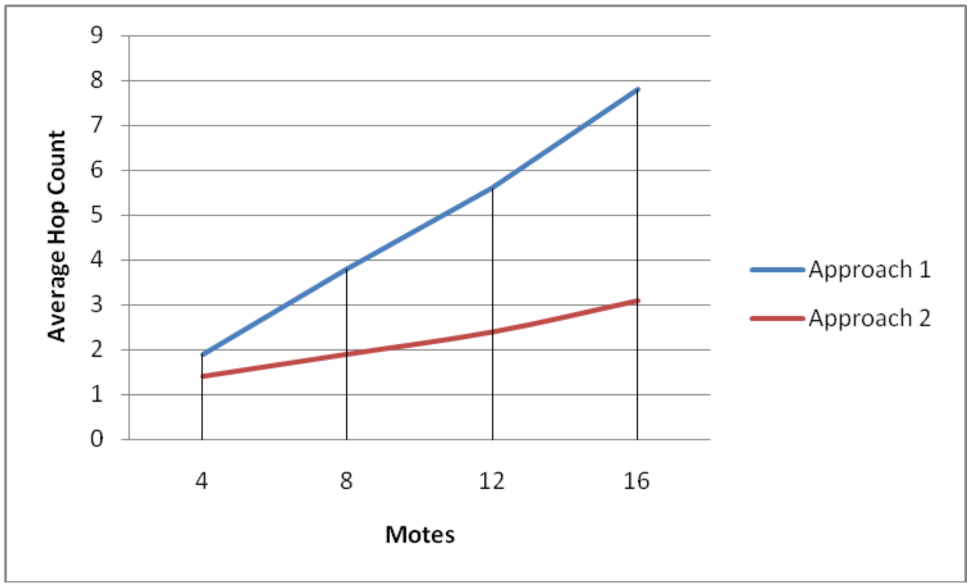

Figure 9-4: Hop count Vs Motes

The above graph summarizes the average hop count reported for different topology sizes. Approach 1, as it uses the broadcast approach it does not remember about the request message that it recently processed. When it receives a duplicate request messages, Approach 1 assumes that the request message will be a new message that needs to be replied. The duplicate request messages might traverse through longer path to reach the destination. The reply message will be sent through the same path through which the request message is received. The hop count for most of the reply messages generated by a node observed as around 8. So the average hop count using Approach 1 will be around 8. Where as in Approach 2, as the client constantly collects the neighbor information from the request messages, it will update the better routes to the base station for each request message and will send the reply message in the best route. So, in the second approach the average hop count will be much lesser than the Approach 1 and a huge improvement is observed when compared with Approach 1.

9.4 Messages Vs Back Off Timer

This test case is used to analyze how the total number of messages generated to retrieve the complete network information varies when changing the back off timer. This test is performed with 16 motes. The first test run is performed without setting the back off timer and 313 messages are generated to collect the complete network information. Without the back off timer, there will be significantly more number of messages generated because of the message collision when sending a broadcast message by two neighboring motes at the same time. The next test runs are performed by setting the back off timer with different values. It can be observed from the graph that there is an improvement in the number of messages generated but the change observed is very less. It is also observed that the difference in number of messages generated is observed similar after increasing the timer value from 200msec. To have a better response time to reply message 20 msec can be set as the back off timer value.

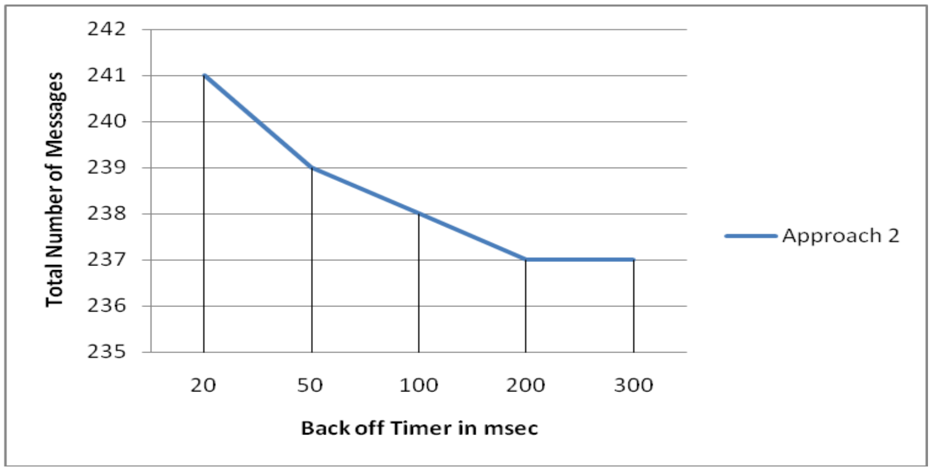

Figure 9-5: Back off timer Vs Messages

9.5 Topology Change

The number of messages generated to collect the complete network information is run for a linear topology. This test will help in understanding which topology will work better for the second approach. This test is compared with the number of messages generated with different topology sizes. Approach 1 and 2 performed similarly with a linear topology. The number of hops is also similar for both approaches. This is because the request message will be broadcasted by only one client mote at any point of time and it involves only one path to reach the destination. There will no scope to reduce the number of messages and hops. So both approaches perform similarly with a linear topology. From this and above test cases, Approach 2 will perform better when the client motes are dense enough which involves multiple paths to reach a destination.

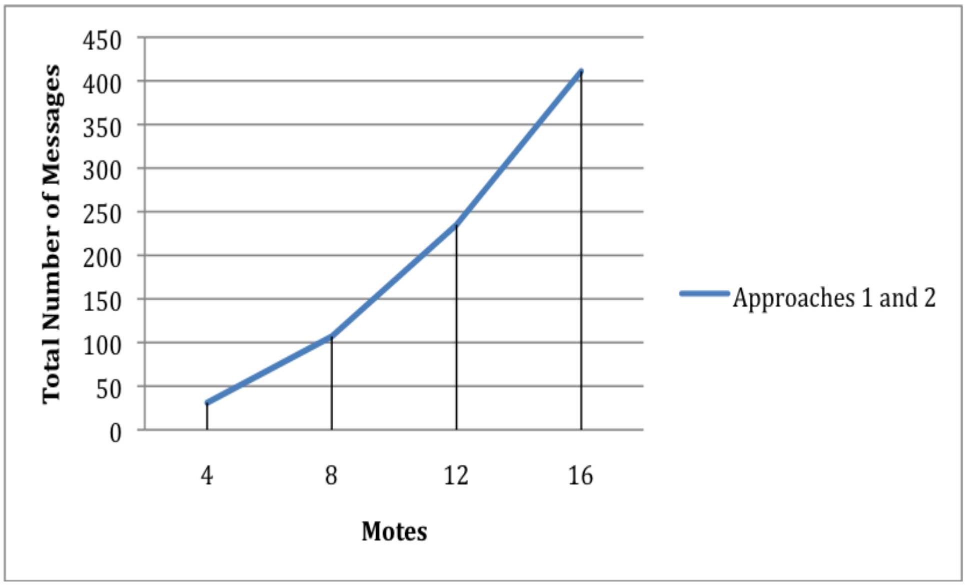

Figure 9-6: Messages generated in linear topology

Chapter 10 - Conclusion

This chapter summarizes the lessons that are learned during this project.

This project is involved in developing a data collector application which collects the environmental attributes of the sensor motes present within a large building. The ad hoc routing protocol concept is used to learn and distribute the complete network topology with the neighbor motes. The protocol is implemented in a way that it will reduce the message traffic and using minimum number of hops in reaching the destination while building the complete topology. The sequence number and hop count are used to reduce the duplicate messages and find the better route.

Route request, Route Reply, Neighbor table lookup and Neighbor table update algorithms has been proposed to process the request and reply messages efficiently and accurately. This application can be used to get the temperature changes within a building by arranging the client motes in different locations. Performance testing has been done to evaluate the number of hops and the number of messages that are involved in building the complete network information.

Chapter 11 - Future Work

TinyOS provides the message structure with which we can send the data of 30 bytes. So, the number of motes is limited by the data length and for successful message transmission the message size should be 30 bytes. There can be limited number of motes that can be arranged within the network. This has to be extended by increasing the default size and implementing the application with large number of motes.

This project can be extended to implement clusters by subdividing the network into multiple sub networks. In this way the number of messages that are generated will be reduced and increases the performance of the application. For this, the messaging architecture and the algorithms need to be changed.

References

[1] Römer, Kay; Friedemann Mattern (December 2004), "The Design Space of Wireless Sensor Networks", IEEE Wireless Communications 11(6):54–61, doi:10.1109/ MWC. 2004.1368897

[2] Perkins, C.; Belding-Royer, E.; Das, S. (July 2003). Ad hoc On-Demand Distance Vector (AODV) Routing. IETF. RFC 3561. Retrieved 2010-06-18.

[3] Reactive Protocols
http://www.olsr.org/docs/report_html/node16.html

[4] TinyOS-2.x Tutorial
http://www.docs.tinyos.net/index.php/TinyOS_Tutorials

[5] Philip Levis , Nelson Lee , Matt Welsh , David Culler, TOSSIM: accurate and scalable simulation of entire TinyOS applications, Proceedings of the 1st international conference on Embedded networked sensor systems, November 05-07, 2003, Los Angeles, California, USA [doi>10.1145/958491.958506]

[6] nesC
http://nescc.sourceforge.net/

[7] The nesC Language: A Holistic Approach to Networked Embedded Systems, D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, D. Culler, Proceedings of Programming Language Design and Implementaiton (PLDI) 2003, June 2003.

[8] Khaled Elmeleegy, Alan L. Cox, T. S. Eugene Ng, "Understanding and Mitigating the Effects of Count to Infinity in Ethernet Networks", to appear in IEEE/ACM Transactions on Networking, February, 2009

[9] Wireless Sensor Network
http://en.wikipedia.org/wiki/Wireless_sensor_network

[10] TelosB
http://www.xbow.com/pdf/Telos_PR.pdf

[11] Image Source: Download from
http://www.willow.co.uk/TelosB_Datasheet.pdf

[12] Image Source: Download from
http://en.wikipedia.org/wiki/File:WSN.svg

[13] SNMP

http://www.snmp.com/protocol/

[14] NET-SNMP

http://net-snmp.sourceforge.net/

[15] Management Information Base

http://www.snmplink.org/snmpresource/mib/