DEPLOYING MULTIPLE SENSOR APPLICATIONS IN A NETWORK

by

SUDHIR CHANDER REDDY KONDAM

B.Tech, Jawaharlal Nehru Technological University, India, 2008

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

TinyOS is an open-source component based operating system designed for highly memory constrained wireless embedded sensor network. TinyOS includes interfaces and components for communication management, routing and data acquisition tools to be refined further for custom applications.

This project aims at developing a system which detects overlapping paths for data collection in different applications in the network and utilizing that information for efficient data acquisition. This prevents a reconfiguring the entire network of wireless sensor nodes (called motes) for each new application request. The application for initial or first data acquisition request tries to build the tree architecture on motes in the network where each node in the tree knows its immediate parent and children. The application builds the tree routed at the base station for the initial request and each intermediate node sends data to its parent when the data request is made. Each base station can request Light, Temperature and Passive Infrared sensory data from all or a subset of motes present in the system.

When a new base station comes and connects to the network through a mote/node in the tree, the system reconfigures only those parts of the tree built in the initial phase which do not overlap with the tree required for the new base station as the root, all the other overlapping parts of the tree are left unchanged. We present experimental result to illustrate the efficiency of the approach.

Table of Contents

List of Figures

Acknowledgements

I would like to take this opportunity to thank my Major Professor and advisor, Dr. Gurdip Singh for his immense support, inspiration and guidance during my report work and the entire Masters program.

I would also like to thank Dr. Mitchell Neilsen and Dr. Doina Caragea for being in my committee and providing their valuable cooperation during my project and the entire Masters program.

Finally, I thank my parents and brother for all their invaluable support and encouragement.

Dedication

I would like to dedicate this work to my advisor Dr. Gurdip Singh, for his tremendous and immense support and guidance without which this project wouldn't be possible.

CHAPTER 1 - Introduction

1.1 Introduction

In the world of automation, each device is expected to work in an unsupervised environment. One such application is a monitoring service. Thanks to the world of sensors and other associated technologies, now one can monitor attributes by sitting in another part of the world. The ability to sense real world attributes such as temperature, visual light and motion has revolutionized the field of energy conservation, healthcare management and various other walks of life.

This project is one such effort to develop infrastructure for energy conservation and object monitoring by providing a monitoring system to detect light, temperature and motion. The data obtained from monitoring can be utilized not only for energy efficiency but can also be used for surveillance.

1.2 Wireless Sensor Networks and TinyOS

Wireless Sensor Networks (WSN's) consist of spatially distributed sensors (called motes) to monitor various environmental attributes such as temperature, visual light intensity, humidity, sound and object motions which communicate using built-in wireless radio transmission modules. Most sensor network applications consist of a multi-hop algorithm where each mote forwards its data to reach the base station.[14]

TinyOS was specifically designed for wireless sensor networks with component based software design and minimal power and resource consumption while implementing sophisticated protocols and algorithms which are severely constrained in tiny, low-powered motes comprised in WSN's.[3]

1.3 System Overview

Systems used for the approaches discussed in this report are composed of a "data requesting application", a "base-station" and set of motes with sensors forming a network.

Users of these systems connect to the network of motes with sensors (TelosB) using a "*data requesting application*", a Java program. This data requesting application running on a PC

connects to a "*base-station*" attached to it. Each user provides a *moteID* to which the base-station connects, along with a list of other moteIDs from where the data is to be fetched as input to the data requesting (java) application.

A "*base-station*" is a TelosB mote used as an interface to interact with the network of motes and is also programmed to act as a communication bridge between the PC and the network. The mote to which a base-station connects is called a "dummy base-station".

A "*dummy base-station*" is like any other mote on the network except that when it receives a data acquisition request from a base-station, it initiates the network building algorithm and establishes a path between itself and a mote from where data is to be fetched. Each mote in the network is basically a TelosB sensor equipped with various sensors whose readings are sent when requested.

1.4 Motivation

In wireless sensor network, the amount of time and the number of messages spent in re-configuring the network of motes for each new base-station/data requesting application are crucial. The goal of each system is to build a path from data requesting mote or a base-station to the mote from where data is to be acquired with minimal use of messages and in minimum time.

In Approach-1, for each data acquisition request from a new base-station coming onto network, the entire network is re-structured by the system. This results in loss of vital network knowledge already present at motes and is accompanied with re-transmission of messages sent previously for building network. As a result there is an increase in time for getting response to the request sent by the new base-station.

Hence, there is a necessity for developing an approach with which can take requests from users using an application and re-configure the network by using the network knowledge already present with less number of messages and less amount of time to build a path from motes from where data is to be acquired to the dummy base-station for a new data requesting base-station coming in to previously built network.

The system developed for Approach-2 uses components similar to the components used in Approach-1 system for external and internal system interactions but aims at resolving the above stated problem by reducing the number of messages sent for re-configuration of the network thus reducing the amount of time spent for getting response to the request sent by the

new base-station. Approach-2 in this project uses an algorithm which exploits and utilizes the network knowledge present at mote for re-structuring only a necessary part of network whenever a new base-station connects to the network and requests the data from certain set of nodes.

CHAPTER 2 - TinyOS

2.1 TinyOS

TinyOS is an operating system developed for low powered and highly constrained wireless sensor networks by University of California, Berkeley. The open source code of TinyOS is written in a language called nesC, a dialect of C language used for developing embedded sensor applications as a set of tasks and process cooperating together with each other concurrently. Its component based architecture helps users in minimizing the code to be deployed on the low powered motes of wireless sensor network while providing rapid innovation.[1]

TinyOS component library in provides drivers for various sensor boards and other default tools necessary for developing wireless sensor network applications.[2] Its supplementary tools are generally front ends developed in Java and other shell scripts which provide interfaces for other languages like .NET to interact with sensor applications. Each TinyOS application is built using a number of software components which are statically linked using the interfaces which define those components. Being a component based development environment allows TinyOS in increasing code reusability between various applications. Components in a TinyOS application post tasks which will be run by operating system in later point of time these tasks generated are non-preemptive and are run in FIFO ordering while large instructions are run using a set of tasks.

2.2 nesC

nesC stands for **n**etwork **e**mbedded systems **C**. nesC is a language similar to C language which encompasses various concepts and models in TinyOS. Different components are wired (assembled) to form a single nesC application. Components in nesC have a set of tasks thus providing concurrency internally. Interface of a component defines the functionality provided by that component and also set of functions called events that must be handled by the interface user. User application using interfaces provided by respective components to interact with the component with the help of set of functions called commands they provide. Each user statically wires components together to allow interactions between the components. This static wiring improves the robustness and efficiency of the code which is tested at compile time itself.[5]

CHAPTER 3 - TOSSIM

3.1 TOSSIM

Debugging a distributed wireless sensor network with large network sizes, number of events and with large number of wireless messages in the system is pretty tedious job. Presence of a simulator for simulating these events in network along with capturing of the messages generated in system reduces the effort required to debug the application. TOSSIM is one such discrete event simulator developed by University of California, Berkeley used for simulating distributed wireless sensor network applications in TinyOS.

TOSSIM allows users to compile source code of any TinyOS application and simulates it for a large network of motes on any standalone computer. This allows the user to efficiently debug the application and analyze the behavior of the distributed application to a fine grain and thus scaling application to a large number of real-mote networks. TOSSIM provides user's control over the simulation with an option to insert a debug message packet which drives the simulation of the system being tested to show the user specified behavior.

TOSSIM provides users with two radio models one model allows users to specify the probability of corruption per bit while another model allows every bit to be received without corruption. Users can use these radio models for efficient debugging of their sensor application.[6]

3.1.1 Limitations of TOSSIM

TOSSIM aims at providing simulation of the TinyOS application with a highly fine grained network interaction. This sometimes forces TOSSIM not to take all real world situations into consideration and assume something while simulating the application to keep the instance simple and efficient.

TOSSIM does not provide a mechanism to run two different source codes or applications in the same instance of simulation. Thus user has to implicitly define different behavior in the same code for required set motes.

TOSSIM lacks in estimating the power consumption which can affect the radio communication range and signal strength of a mote.

TOSSIM works with a discrete event queue where each interrupt is not preempted as interrupts are generated periodically at 8 MHz (clock frequency of TelosB) but in real world interrupts can occur when mote is busy running some code which could result in system crash.[6]

3.2 TinyViz

TinyViz is an application developed in Java to provide users with a GUI to visualize and control the actual environment of wireless sensor networks on a standalone system. TinyViz provides a visual presentation of the simulation instance ran by TOSSIM with all communication done using a socket connection with TOSSIM. [16]

TinyViz framework provides a number of plug-ins for visualizing and controlling simulation. One such plug-in is “radio links” component which can be used to depict the sent and received radio signal pictorially. Plug-ins include “radio message” plug-in used for inspecting the message packets sent or received by any mote in the network. TinyViz depicts a TOS broadcast using a circle around a mote and a directed arrow for a unicast allowing users with better visual experience. One can monitor the network behavior for the application by controlling and changing the radio signal strength using “radio model” plug-in of TinyViz while the application is still running. Ability to allow dragging and dropping of motes or selecting default layout in “layout” drop down allows user to test application for different network topologies. TinyViz also provide a number of other plug-ins to monitor various mote behaviors including ADC readings at motes.

The main TinyViz class is a jar file, which is located in tools/java/net/tinyos/sim/tinyviz.jar which can be attached to any running simulation. TinyViz also provides users with an option to introduce delays in simulation with the help of slider; one can configure how long the delay in handling the events occurrences in TOSSIM. TinyViz engine uses an event-driven model allowing easy mapping with TinyOS’s event-based execution. Event bus provided in TinyViz allows it to read simulation events and pump them to respective active plug-in outputs. TinyViz have provisions to add user defined custom plug-ins to default list of plug-ins.

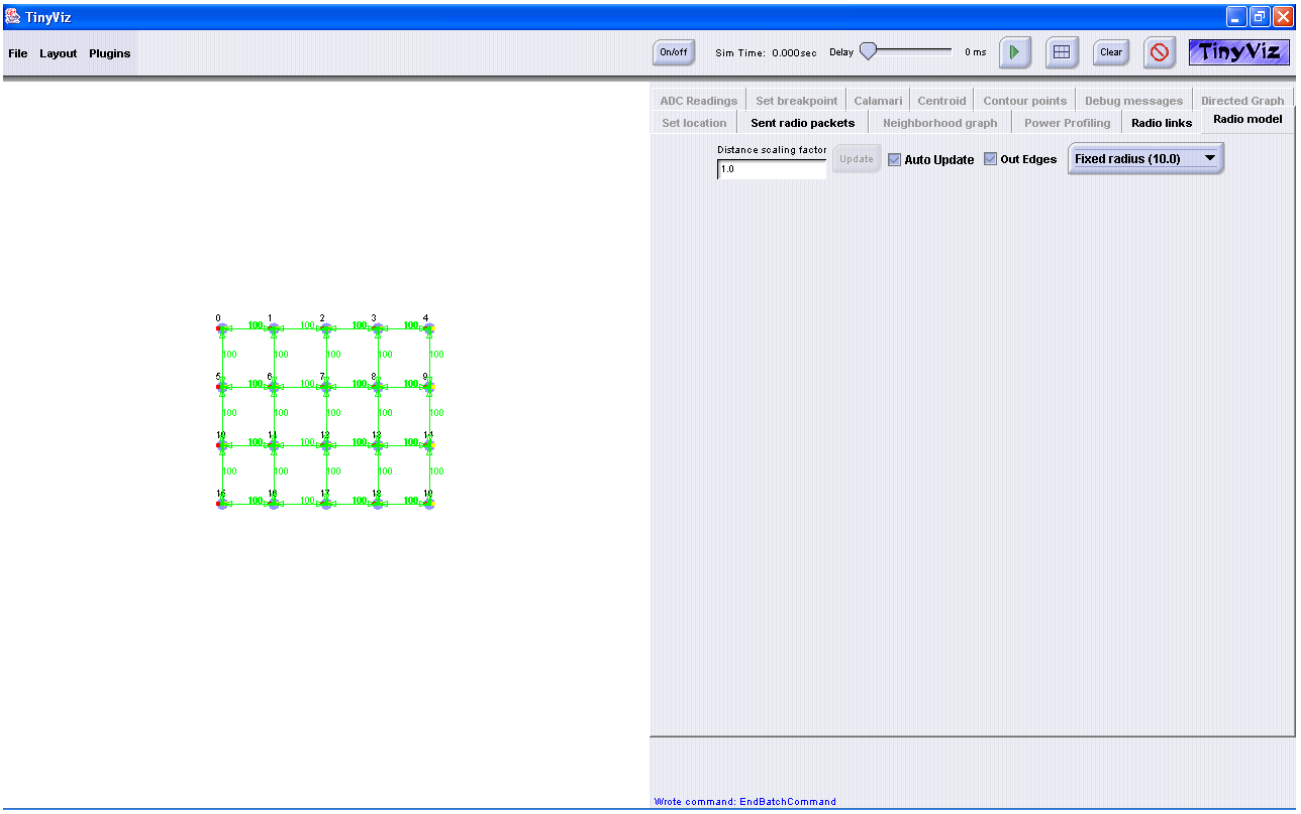

Figure 3.1 Snapshot of TinyViz GUI

In the above snapshot we can see how a TinyViz application window looks like. On the right side of the application window we can see all the plug-ins provided by TinyViz to the user. On the left we can see a network of motes with green lines representing the radio signal reach ability of mote. The top left portion holds the plug-in menu where user can activate and deactivate plug-ins. It also holds layout menu which user can use to specify the layout to be used. The file menu on the top left corner provides users to save the current topologies and also load previously stored topologies. On the right top corner, we can see the button for start and pausing simulation and slider to introduce delays buttons.

CHAPTER 4 - Hardware Components

4.1 TelosB Mote

Figure 4.1 TelosB mote

Image Source: [8]

TelosB sensor is one of the many sensor motes used in wireless sensor network applications. It was developed by University of California, Berkeley and was contributed to wireless sensor network research. TelosB houses a powerful TI MSP430 microcontroller with 8 MHz clock frequency designed to work under heavily power constrained WSN's. It also provides users with 10kB of RAM used only for programming and other data storage where programming is done using the UART interface present on board. TelosB sensor board is equipped with an IEEE 802.15.4 compliant RF transceiver which can speed up to 2.4GHz radio. Board also features an onboard embedded antenna for receiving and transmitting signal. TelsoB sensor bundle comes with an optional pre-installed environmental sensor suite for light, temperature and humidity sensor. In order to provide low power consumption when running on external batteries (AA), TelsoB mote was designed to stay most of the time in sleep mode and do a quick wake up from sleep when required.[17]

4.2 EasySen WiEye Sensor Board

Figure 4.2 EasySen WiEye Sensor Board

Image Source: [10]

EasySen WiEye is a sensor board developed by EasySen crop, used along with TelosB sensor boards in various surveillance and other security applications. WiEye connects directly with TelosB using the 16 pin extension connector present on TelosB sensor board and has ability to work in low-powered environmental setup. WiEye sensor board provides a powerful long range passive infrared (PIR) sensor, a visual light sensor and an acoustic sensor on a single chip to be used in various security and object tracking applications. Long range PIR sensor present on WiEye can detect human presence from a distance of 20-30 feet and presence of other objects from a distance up to 50-150 feet, both with a detection angle ranging from 90° to 100°. [18]

4.3 Other Components

Other hardware components used for development of test bed for the system include use of USB hub, USB Cables, Plexi Glass Board and Velcro.

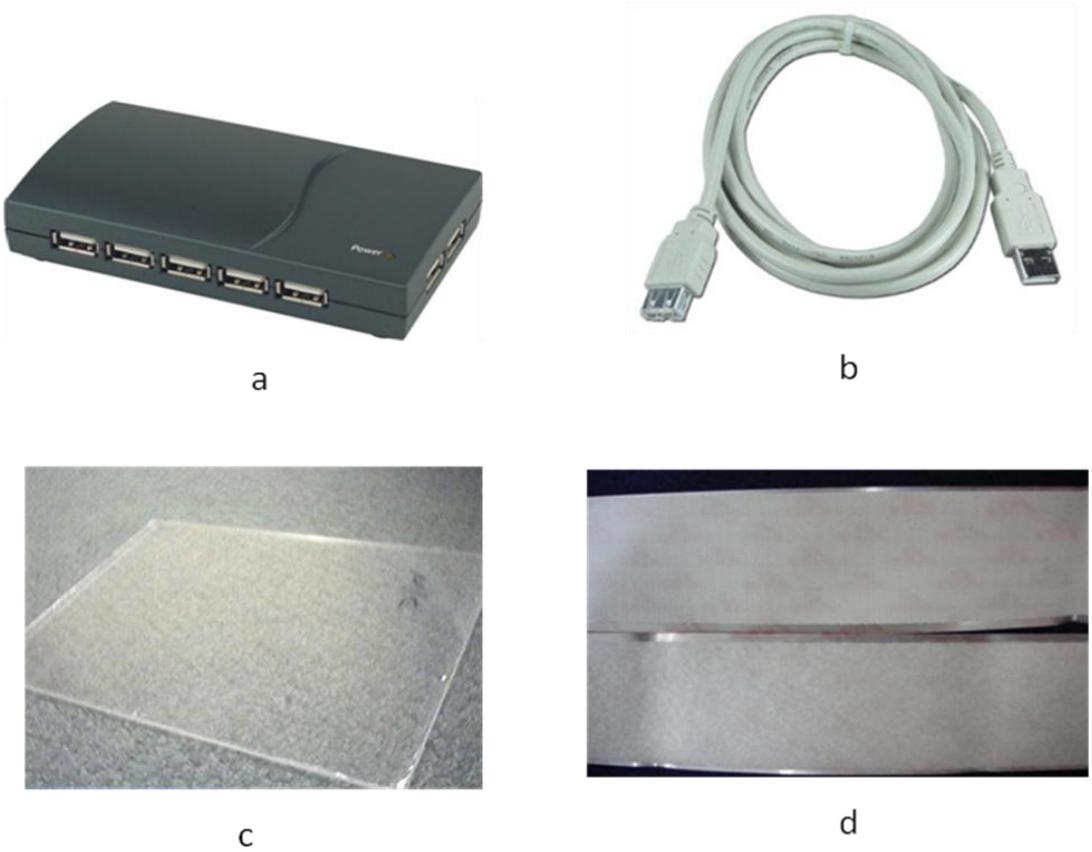

Figure 4.3 a: USB hub , b: USB cable, c: Plexi glass board and d: Velcro

Image source: [11], [12], [13] and [14]

Plexi glass board was used to house TelosB motes with the help of Velcro. USB cables were used to power TelsoB motes with power distribution done using USB hubs.

4.4 Test Bed Setup

The test-bed for the system was created using 2 Plexi glass boards with each plexi glass board housing a total of 8 TelosB motes in a mesh of 2 rows and 4 columns. All motes were attached to board using Velcro. A USB hub was used to distribute power to all motes through USB cables (shown in below given figure).

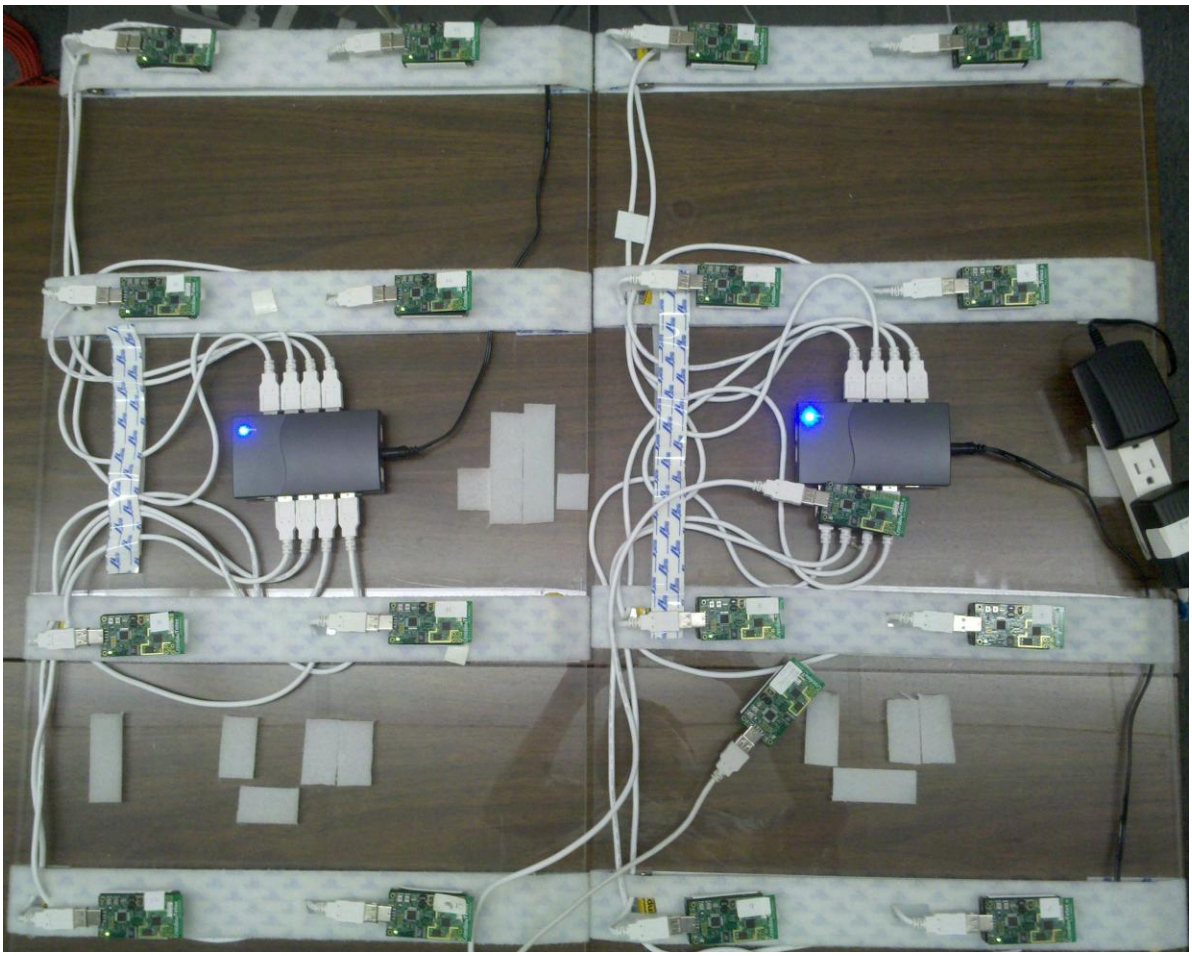

Figure 4.4 Test bed

CHAPTER 5 - System Architecture

5.1 System Architecture

The architecture of the system used for Approach-1 and Approach-2 consists of a network of 16 motes connected on 2 test beds, Base-station and a Desktop or Laptop. Users connect to the base-station which is attached to the laptop or a desktop executing a Java application. Each user provides a moteID to which a base-station connects, along with a list of moteIds from where data is to be fetched. The mote to which base-station connects is termed as "dummy base-station". Once a dummy base-station gets a request from a base-station it starts tree building algorithm only if it has never participated in tree building algorithm or it has not already initiated the tree building algorithm.

Once the dummy base-station is done building tree in the network of motes it then sends the send data signal. Each mote checks for its ID in the list of motes which has to send data. A mote sends its data to its parent for that dummy base-station request if it was asked to send its data. Once data message reaches the dummy base-station it forwards that message packet to appropriate base-station.

The system allows multiple base-stations to connect to the same dummy base-station or to different dummy base-stations with same or different mote lists for data acquisition. When a new base station comes and connects to the network through a mote in the tree, the system reconfigures only those parts of the tree built in the initial phase which do not overlap with the tree required for the new base station as the root. All overlapping parts of the tree are left unchanged thus making an optimal use of network.

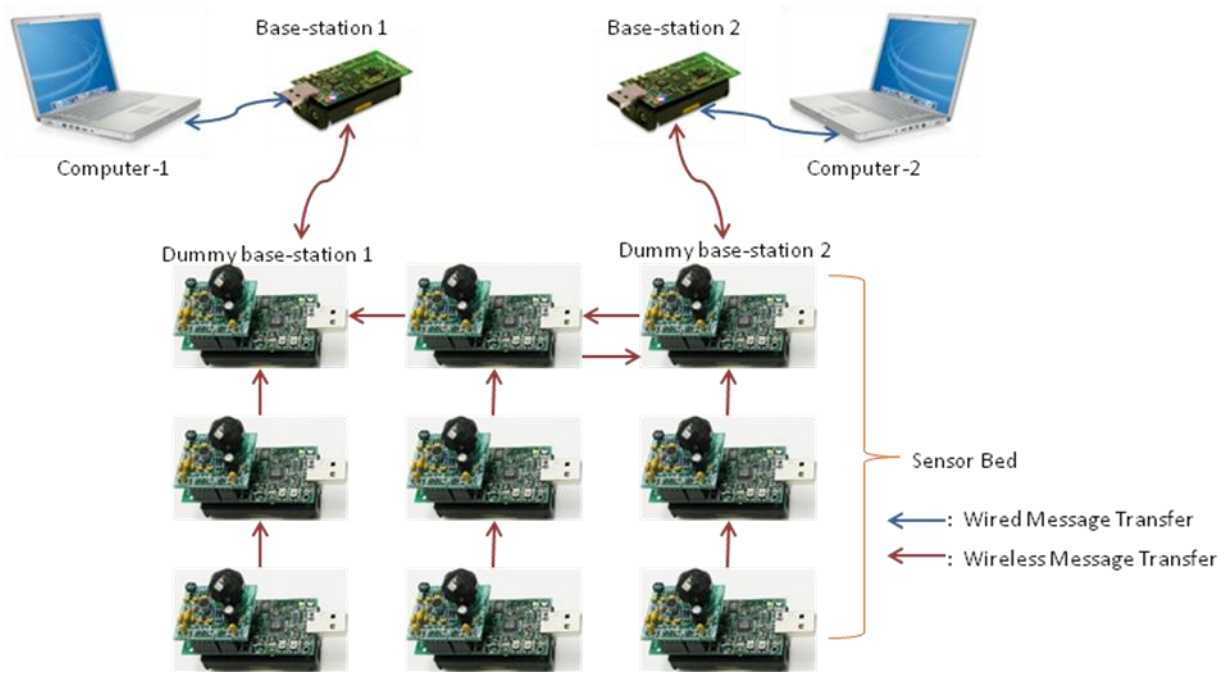

Figure 5.1 An instance of system

Image Source: [7], [8] and [9]

5.2 Algorithm

The algorithm developed for system with Approach-2 assumes that parameters myroutingtable, sources, ackcount, routecount, children, PHASE and motesrcs are initially set to 0 while all other flags present are set to FALSE. The algorithm assumes that Base-station ID(s) connecting to the system do not match with moteIDs present in network.

Below given are the steps which the proposed algorithm is composed of:

1. If a mote receives a Type-0 message from a base-station it will set cmpsource to TRUE i.e. it's a dummy base-station. Mote also stores the base-station ID which sent Type-0 message and list of moteIDs from where data is to be acquired along with updating their motesrcs entries with base-station's ID. Dummy base-station before starting neighbor detection process checks
    a. If it has already done its neighbor detection (i.e. in PHASE=3) then it just broadcasts send data message a Type-10 message along with recently connected base-station ID and list of motes and which it asked to acquire data from. It will also send an acknowledgement to base-station a Type-12 message.

b. If dummy base-station has not done its neighbor detection process (i.e. in PHASE=0) it starts the process by broadcasting a message with its ID a Type-1 message. This message also contains the base-station ID to which dummy base-station is connected. It now sets its PHASE=2 and starts a timer called Timeout timer which will be fired if no acknowledgement is received for Type-1 packet sent. At last mote will send an acknowledgement to base-station a Type-12 message.
c. If dummy base-station is in neighbor detection process (i.e. 0<PHASE<3) then it will not acknowledge the Type-0 message from base-station.

2. If a mote receives a Type-1 message then
    a. If mote has already received a Type-1 message (i.e. in PHASE > 0) it just do nothing.
    b. If mote is in PHASE=0 then it sets source of Type-1 message as its parent and sends an acknowledgement a Type-2 message back to the source. It now starts neighbor detection process by broadcasting its ID a Type-1 message. It then starts Timeout timer for acknowledgements and sets its PHASE=2;
3. If a mote receives a Type-2 message then
    a. If mote has received a Type-2 message from this source earlier it discards the packet.
    b. If mote never received a Type-2 message from this source it updates list of children with the source ID and increment its ackcount by 1.
4. If a mote doesn't get any reply for Type-2 message sent previously, then Timeout timer is fired and mote sends a message to its parent with its ID a Type-3 message indicating present mote is leaf node in the tree. After sending a Type-3 message it sets its PHASE to 4.
5. If a mote receives Type-3 message then
    a. If it's a dummy base-station and if it dint get a Type-3 message from this source earlier then it will increment routecount counter by 1. If dummy base-station mote got Type-3 messages from all of its children then it will broadcast a message with its ID, base-station's ID and list of motesIDs

from data is to be acquired a Type-10 message. It will now set its PHASE=4.

b. If it's not a dummy base-station and dint get a Type-3 message from this source earlier then it will increment its routecount counter by 1. If it got Type-3 messages from all of its children then it will send a Type-3 message to its parent and setting its PHASE=4.

6. If a mote receives a Type-10 message from its parent then
    a. If motes already got the pair of dummy base-station and base-station with Type-10 message then it discards the packet.
    b. If mote saw this pair of dummy base-station and base-station with Type-10 message for the first time and
        i. If its ID is present in list of motes which have to send data then it sets Send flag to TRUE, stores dummy base-station's ID and base-station's ID in SRCS then broadcast the Type-10 packet with its ID as source. If Send flag is set then mote will start a Timer which fires periodically to send data to its parent a Type-11 packet.
        ii. If its ID is not present in list of motes which have to send data then it will just broadcast the Type-10 packet with its ID as source. Mote will also update the motesrcs for the mote ID in the list for data acquisition with the source of Type-10 packet.
7. If a mote receives a Type-10 message from its child then
    a. If mote dint see the dummy base-station and base-station pair earlier then it will update Srcs with the new pair and it updates Child with the source (a back-link to child represents that new base-station is in the system).
        i. If mote's ID is listed in list of motes in Type-10 message then it will set Send flag to TRUE and start the timer which will send the data periodically to parent.
        ii. Else it will just update the motesrcs of moteID present in the list with new source.
        iii. Once mote is done doing updates it will broadcast obtained Type-10 message for further propagation.

8. If a mote receives a Type-11 message then it will forward it to ID(s) obtained from motesrc[childID].

5.3 Message Format

Any wireless sensor network application relies on wireless messages sent over radio for efficient interactions between the motes which include data transfer between motes. TinyOS applications developed for any interaction in WSN's utilizes TOS messages. TOS uses active messages which are defined in tos/system/types/AM.h. Each active message carries fields for a destination address, group ID, message type (the AM handler ID), length and payload. The maximum payload size is TOSH_DATA_LENGTH set to 29 bytes by default and can be set to 36 byte.[19]

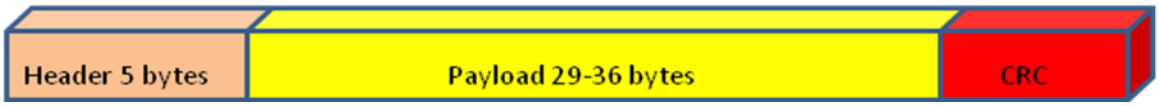

Figure 5.2 TOS Message Structure

Destination address in TOS message is 2 bytes and have some reserved addresses:

0xFFFF : Radio broadcast address specified by TOS_BCAST_ADDR.

0x007E : UART address specified by TOS_UART_ADDR

Local address of the mote is specified using constant TOS_LOCAL_ADDRESS.

The system uses the message structure defined in ClientMoteMsg.h as given below for interactions between motes.

```
typedef struct ClientMoteMsg
{
    uint8_t Id;          //used for specifying sender of packet.
    uint8_t src;         //used for specifying base-station ID.
    uint8_t data[5];     //used for sending various sensory data.
    uint8_t motes[5];    //used for specifying moteIDs for data acquisition.
    uint8_t num;         //used for specifying packet number.
} ClientMoteMsg;
```

The system with Approach-2 uses src and data [0] for specifying dummy base-station and base-station pair in Type-10 message. System utilizes data [4] always for specifying message type.

5.4 Wiring

The operation of linking an interface used by a component to an interface provided by another one is called “wiring”. Implementations section of a configuration file wires modules or other components interfaces. nesC uses arrows(->) to determine relationships between interfaces. The left side of the arrow binds an interface to an implementation on the right side i.e. the component that uses an interfaces is on left , and the component provides the interface is on the right for E.g. BlinkM.Timer -> SingleTimer.Timer line is used to wire the Timer interface used by BlinkM to the Timer interface provided by SingleTimer. BlinkM.Timer on the left side of the arrow is referring to the interface called SingleTimer while Timer on the right side of the arrow is referring to the implementation of Timer present in SingleTimer. nesC supports multiple implementations of the same interface . Various components used in the system are as follows:

GenericComm	: a component for sending and receiving messages.
HamamatsuC	: a component for Light intensity sensor.
ADCC	: a component for Passive Infrared sensor.
DemoSensorC	: a component for Temperature sensor.
cc2420RadioC	: a component for Radio control.
RandomLFSR	: a component for generating random number.
LedsC	: a component for controlling Leds.
TimerC	: a component for Timers.

CHAPTER 6 - Implementation

6.1 Environment Setup

The basic requirement to implement the system is to have TinyOS-1.x installed on the machine to be used for programming the mote and to act as base-station.

Following are the steps to be followed for installing Tinyos-1.x:

1. Download all the files from http://www.tinyos.net/dist-1.1.0/tinyos/windows/micaz-installer/ under file named TinyOS.
2. Run setup.exe and follow the installation steps as guided by the application.
3. Once finished open Cygwin and change to /opt/tinyos-1.x/ to start working on TinyOS.

Following are the steps to be followed for installing TinyViz:

1. Make sure TinyOS is installed and working correctly.
2. Open /opt/tinyos-1.x/tools/java/net/tinyos/sim/MakeFile.
3. Go to the line saying net/tinyos/packet/*.class \
4. Insert below given three lines below line in line given in step3.
    a. net/tinyos/message/avrmote/*.class \
    b. net/tinyos/message/micaz/*.class \
    c. net/tinyos/message/telos/*.class \
5. Run “make” command in /opt/tinyos-1.x/tools directory.

6.2 Code deployment

Following are the steps to be followed for deploying code on mote:

1. Open cygwin and go to /opt/tinyos-1.x/apps/n_ClientMote
2. Run “make telosb install.<moteID>”. Where moteID is the unique identifier for each mote.

Following are the steps to be followed for deploying code on base-station mote:

1. Open cygwin and go to /opt/tinyos-1.x/apps/n_BStationMote
2. Run “make telosb install.<moteID>”. Where moteID is the unique identifier for each base-station mote.

Note: Mote should be connected to USB port of device and recognized as device.

6.3 Executing Code

Each user interacts with a java application called SendRcvApp.java to communicate with base-station. The system used Message Interface Generator (MIG) a tools for generating Java classes that corresponds to Active Message types that are used in mote applications.

MIG reads in the nesC structs definitions for message types used in mote applications and generates java class for each message type by taking care of details of packing and unpacking fields in the message's byte format. MIG is used in conjunction with the net.tinyos.message package.

The java application for users uses MoteIF, a Java interface for sending and receiving messages to and from motes. This interface will communicate with the serial forwarder and registers a handler to be invoked when a packet arrives. Application also initializes with PrintStreamMessenger which indicates where to send status messages, as well as optional Active Message group ID.[19]

Following are the steps to get the system working:

1. Start cygwin and move to /opt/tinyos-1.x/tools/java/net/tinyos/sendrcv1.
2. Connect the base-station mote to machine and run "motelist" command to get the communication port# at which base-station will listen.
3. Run "export MOTECOM=serial@<COM port#>:57600" command.
4. Run "java net.tinyos.sendrcv1.SendRcvApp".
5. Enter the base-station ID which is connected to machine, dummy base-station ID and list of motes from where data is to be acquired. Now wait for the data to be printed on the screen.

d6.4 Pseudo code

Pseudo code for implementing tree building algorithm:

```
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr m)
{
        atomic    packet = (ClientMoteMsg*) data.data;
        *packet = *((ClientMoteMsg*) m->data);
        srcmote=packet->Id; //source for the packet
```

```
if(myPhase==1 && packet->data[4]==9)
{
        stop sending acknowledgement for Type-2 packet;
        pack->Id=TOS_LOCAL_ADDRESS;
        pack->data[4]=1;
        broadcast signal i.e. send Type-1 message;
        start Timer for detecting acknowledge from children;
}
else if(myPhase==0 && !cmpsource && packet->data[4]==1)
{
        myPhase=1;
        myRoutingTable[0][0]=TOS_LOCAL_ADDRESS;
        myRoutingTable[0][1]=srcmote;
        packet->data[1]=srcmote;
        packet->data[4]=2;
        sends acknowledgement for Type-1 to parent;
        call Leds.redOn();
}
else if(packet->data[4]==2 && myPhase>0)
{
        if(children[srcmote]==0) //dint see this child earlier
        {
                Stop timer for timeout;
                ackcount++;
                myPhase=2;
                children[srcmote]=12;
                pack->data[1]=src2;
                pack->data[2]=ackcount;
                pack->data[4]=9;
                send acknowledgement for Type-2 packet to parent;
        }
```

```
        }
        else if(packet->data[4]==3 && myPhase>=2)
        {
                if srcmote my child
                {
                        routecount++;
                        pack12->data[1]=src3;
                        pack12->Id=TOS_LOCAL_ADDRESS;
                        pack12->data[4]=4;
                        sends acknowledgement to child for Type-3 packet received.
                        checks if (ackcount==routecount)
                        {
                                Tree building is done send Type-10 message packet
                        }
                }
        }
}
```

pseudo code for processing Type-10 message:

```
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr m)
{
        atomic    packet = (ClientMoteMsg*) data.data;
        *packet = *((ClientMoteMsg*) m->data);
        srcmote=packet->Id; //source for the packet
        if(packet->data[4]==10)
        {
                if new dummy base-station and base-station pair then
                {
                        if message from children[] then
                                add srcmote to child[] i.e. backlinking
                                broadcast received packet Type-10.
```

```
                    else if message from parent then
                        broadcast received packet Type-10.
                    Update motesrcs of moteID present in packet->motes[] as srcmote
                }
            }
        }
```

CHAPTER 7 - Execution and Performance

7.1 Execution Phases

7.1.1 Phase-1

In this phase all motes are initialized and the base-station sends message to connect to dummy base-station after user gives the dummy base-station ID. User also gives the list of motes to get data from.

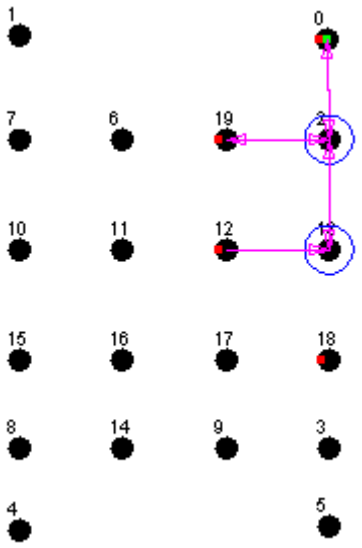

Figure 7.1 Phase-1 of execution

In the above screenshot from TinyViz we can see that the base-station 0 is connecting to the dummy base-station 2. As 2 got a Type-0 message from 0, it sent acknowledgement for that basically a Type-2 message and then it broadcasts the Type-1 message.

7.1.2 Phase-2

In this phase each mote which receives a Type-1 message will send an acknowledgement to Type-1 message i.e. a Type-2 message. Each parent will wait until all of its children are in phase-3 and parent got Type-3 messages from all children.

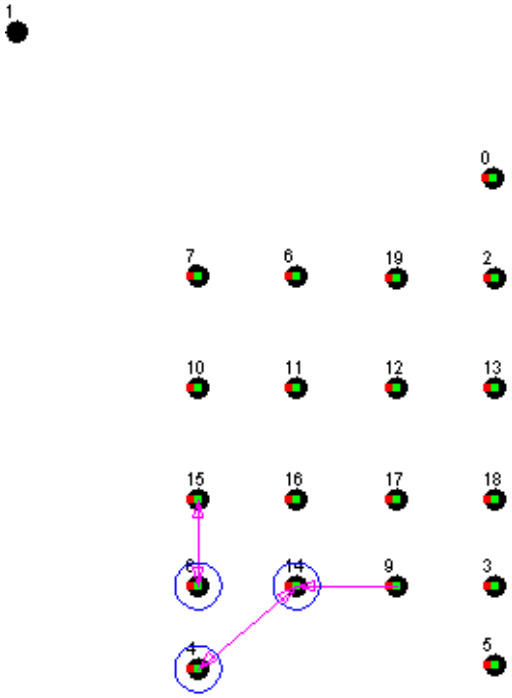

Figure 7.2 Phase-2 of execution

In the above screenshot we can see that the mote 15 sent a Type-1 message to the mote 6 and in turn mote 6 sent an acknowledgement to mote 15 and then broadcasted Type-1 message. Now the mote 15 is waiting for mote 6 to send back the Type-3 message and mote 6 is waiting for timeout to happen. Once mote 6 timeouts it sends a Type-3 message to mote 15 and then it propagates to the root.

7.1.3 Phase-3

In this phase all the motes got Type-3 messages from their children and they sent Type-3 message to their parent eventually reaching root of the tree i.e. the dummy base-station. Once the dummy base-station gets a Type-3 message from children it broadcasts the Type-10 message. Similarly each mote which receives the Type-10 message will send acknowledgement for Type-10 packet to the parent and will broadcast the same Type-10 packet. Each mote will also update motesrcs for those moteIDs listed in list of motes for data acquisition with the present motes parent ID.

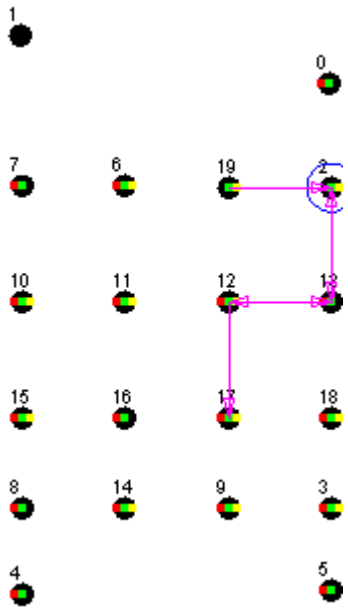

Figure 7.3 Phase-3 of execution

In the above screenshot we can see that 2 the dummy base station broadcasted Type10 message and each of its children are sending acknowledgement to the dummy base-station i.e. their parent. Similarly mote 13 and mote 12 broadcasted the Type-10 message which it got from its parent.

7.1.4 Phase-4

In the phase the motes which are listed in list of motes for data acquisition will start sending the data after they are done broadcasting Type-10 message. Each data packet is sent a Type-11 message each mote which receives a Type-11 message it will forward the type-11 message to the ID which is obtained from motesrcs[srcID]. Where srcID is ID of mote that sent the Type-11 message.

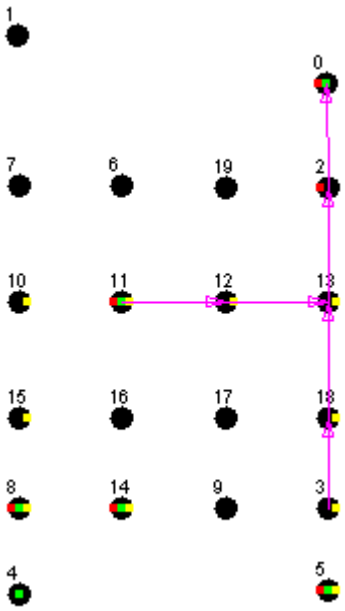

Figure 7.4 Phase-4 of execution

In the above screenshot we can see mote 11 sending its data which is forwarded by mote 12 to mote 13 with the help of motesrc[11].

7.1.5 Phase-5

In this phase new base-station connects to a mote. As tree is already built rooted at dummy base-station of phase-1, system reconfigures only those parts of the tree which is built in the initial phase and which does not overlap with the tree required for the new dummy base station as the root. New dummy base-station will now broadcast Type-10 message packet as sent in section 7.1.3 i.e. Phase-3 but with new dummy base-station and base-station Id pair. If this Type is received by its child the behavior would be same i.e. if child notices new dummy base-station and base-station pair it will process it as described in section 7.1.3 otherwise if this message is received by parent then parent links itself to the child which sent that Type-10 message.

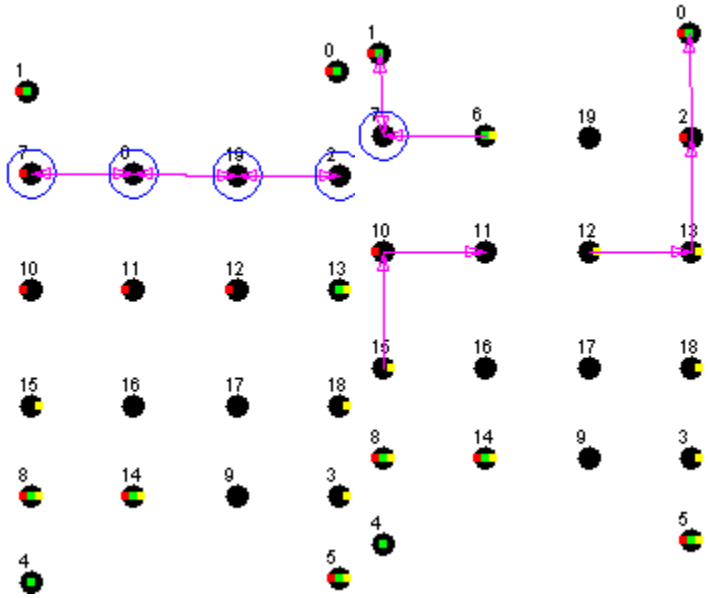

Figure 7.5 Phase-5 of execution

In the above given screenshot we can see that mote 1 a base-station connected to mote 7 but as mote 7 is not in phase-0 it sent an acknowledgement to mote 1 and broadcasted a Type-10 message which is received by mote 6. As mote 6 received a Type-10 message from its child it checks the dummy base-station and base-station pair and find it to be (7,1) which is different from previous pair (2,0) thus it makes a link to mote 7 and broadcasts it which eventually reaches mote 2 which will broadcast it and from there children of mote 2 which dint see the (7,1) pair earlier in Type-10 message will receive and forwards it to its children thus leaving other parts of the tree unchanged. Thus only message from child to parent causes reconfiguration between them and not in whole network. If a mote find its ID listed in motes list for data acquisition it will start sending data only after it forwards the Type-10 packet to its children.

7.1.6 Phase-6

This phase is the final phase where all the motes which were requested by both the base-stations to send data will start sending their data which will be forwarded by the intermediate nodes based on motesrc entry for id of the mote which is sending the data.

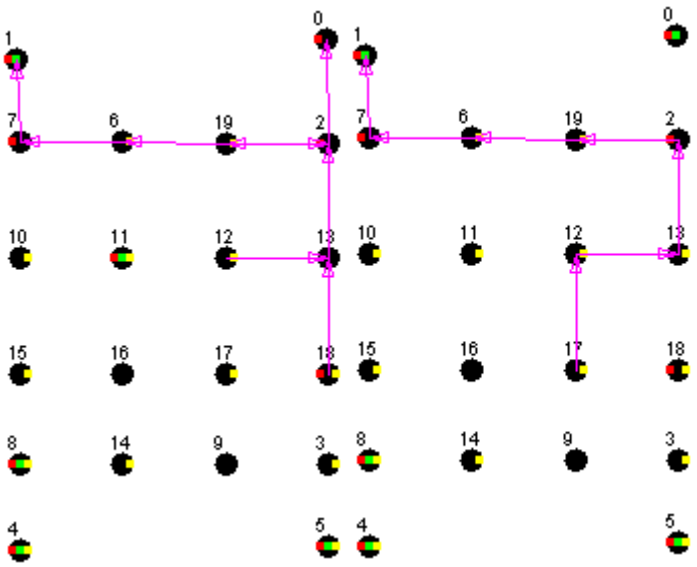

Figure 7.6 Phase-6 of execution

In the above screenshot we can see that mote 17 sending data to mote 7 (in right one) and mote 11 and mote 3 sending data to mote 2 and mote 7 respectively at same time (in left one).

7.2 System Performance

Approach-1 and Approach-2 were tested in many ways to analyze the performance and results for various test cases below given are some performance details. Each test case was made to run 10 times and total number of messages obtained and total time spent was averaged over the number of runs made. Each run of the system consisted of 18 motes in the network.

7.2.1 Scenario-1

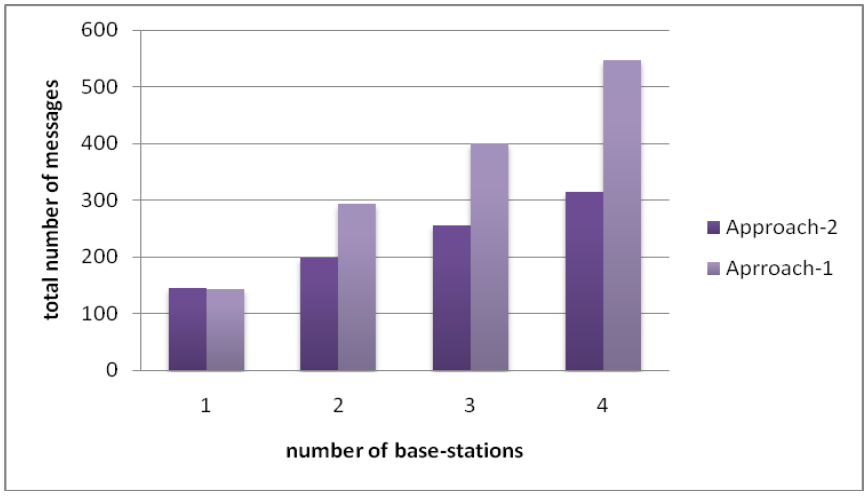

Figure 7.7 Scenario-1 message graph

Scenario-1 is the case where system with both approaches runs with different base-stations connecting to different dummy base-stations and requesting data from different motes. The above graph Figure 7.7 represents that the system with Approach-2 used considerably less number of messages for re-configuring the network than the system with Approach-1, whenever a new base-station comes and connects to the system. From the above graph Figure 7.7 we can notice that Approach-1 for 2 base-stations for building the tree, a total of 293 messages were used which is 47% more than the number of messages used by the Approach-2, which used only 199 messages for 2 base-stations.

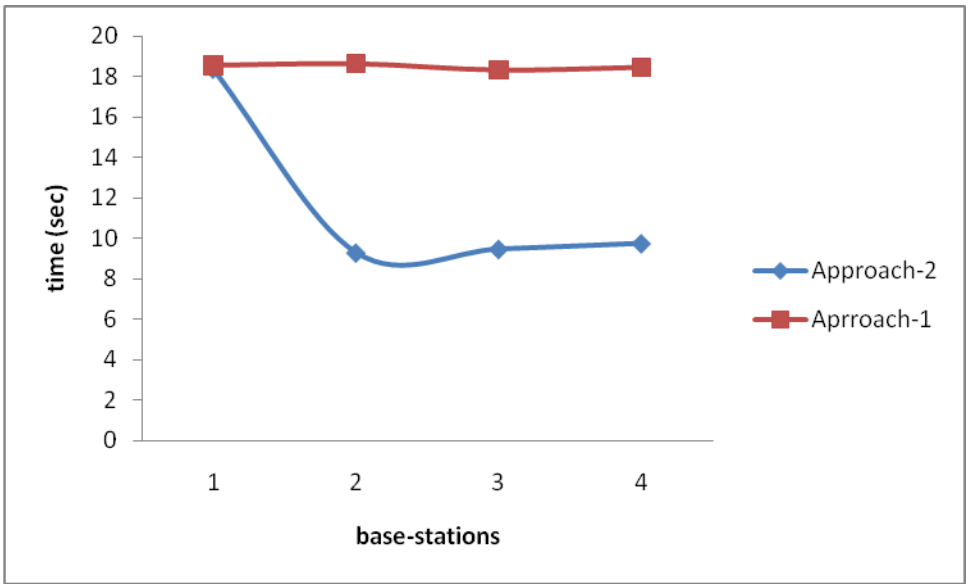

Figure 7.8 Scenario-1 time graph

From the above time graph (Figure 7.8) for scenario-1, we can see that the system with the Approach-2 used less time for getting response (data from desired mote) to the data request sent than the Approach-1 system. It can also be noticed from graph that the Approach-2 takes an average of 9.5 seconds to re-configure the network and get response to the request from a new base-station, connecting to the system (after network was built by one base-station which came previously) but the Approach-1 system used approximately same time for re-configuration which was used for building the initial network.

Goal of the Approach-2 of using less number of messages and time for re-configuration is achieved.

7.2.2 Scenario-2

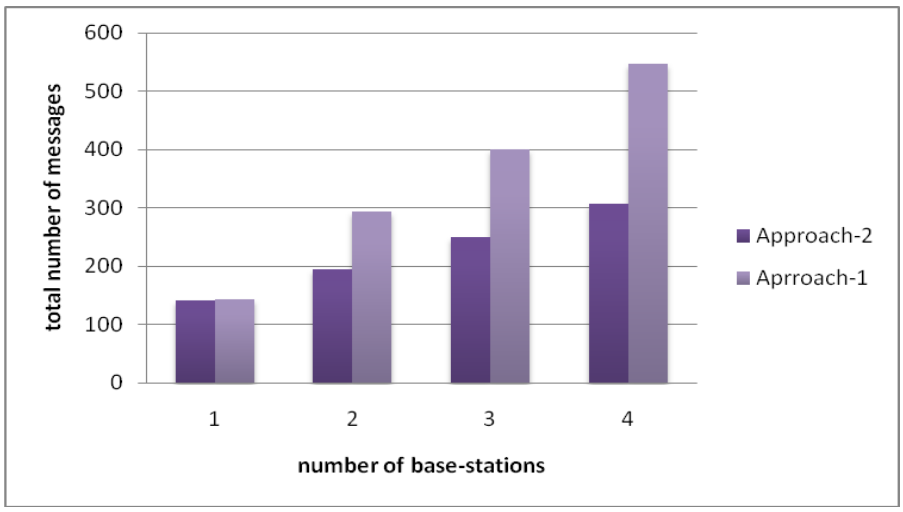

Figure 7.9 Scenario-2 message graph

Scenario-2 is the case where the system runs with different base-stations connecting to same dummy base-station and requesting data from different motes. From the above graph Figure 7.9 generated with test results it was observed that the Approach-2 system used only 193 messages for building and re-configuration of network for 2 base-stations while the Approach-1 system used a total of 295 messages which is 52% more than the number of messages used by the Approach-2 system.

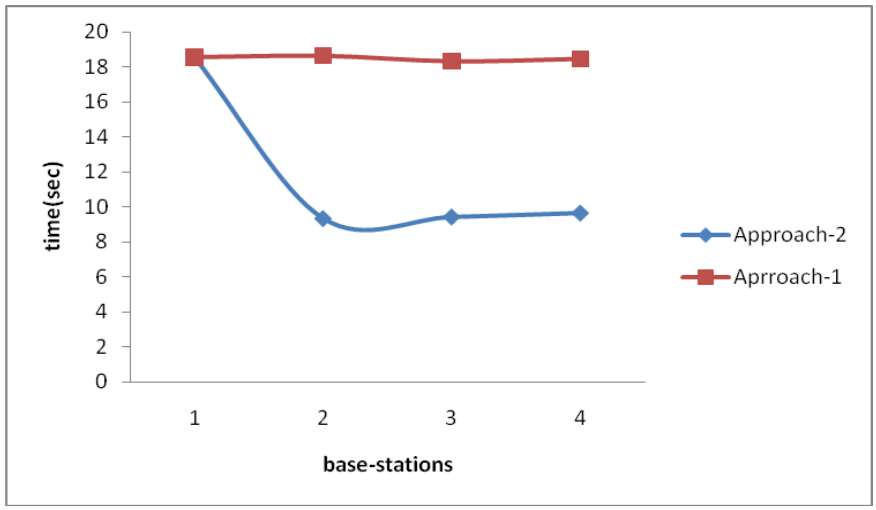

Figure 7.10 Scenario-2 time graph

From the above time graph (Figure 7.10) for scenario-2, we can see that the Approach-2 system takes an average of 9.4 seconds to re-configure the network and get a response to the

request of new base-station, connecting to the system (after network was built by one base-station which came previously) but whereas the Approach-1 system used approximately same time for re-configuration which was used for building the initial network.

Goal of the Approach-2 of using less number of messages and time for re-configuration for this scenario is achieved.

7.2.3 Scenario-3

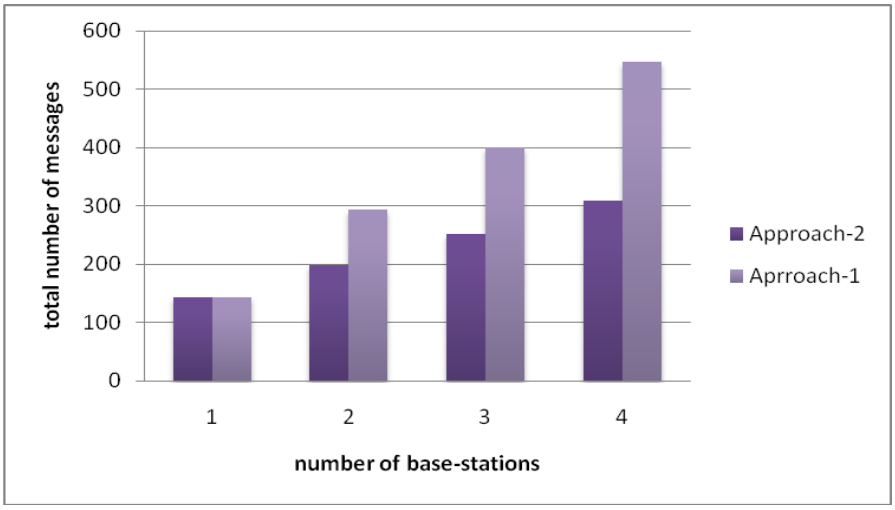

Figure 7.11 Scenario-3 message graph

Scenario-3 is the case where the system runs with different base-stations connecting to different dummy base-stations and requesting data from the same motes. From the above graph Figure 7.11 plotted with the system performances for given scenario it was seen that the Approach-1 system used a total of 400 messages for 3 base-stations which the Approach-2 system used considerably less number of messages for re-configuring the network totaling to only 252 messages. Thus the Approach-1 uses 58% more number of message than the Approach-2 for same network setup.

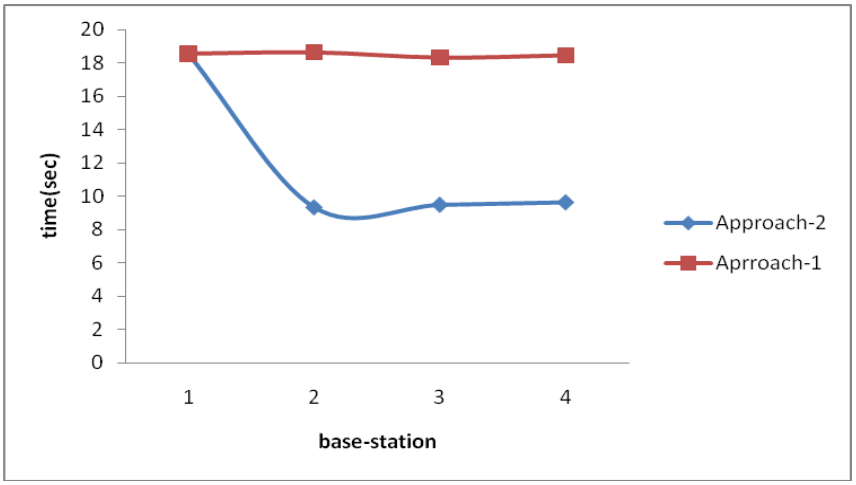

Figure 7.12 Scenario-3 time graph

From the above time graph (Figure 7.12) for scenario-3 we can notice that the Approach-2 system uses an average of 9.4 seconds for re-configuration of the network for a new base-station data acquisition request after network was built initially by other base-station.

Goal of the Approach-2 system of using less number of messages and time for re-configuration for this scenario is achieved.

Performance of the Approach-2 system depicted in terms of message and time graphs for Scenario-1, 2 and 3 above are bit equivalent or same because if the 2 base-stations are connected to different dummy base-station and request data from different or same motes then motes from where data is acquired should be notified that data was requested from them by new base-stations coming onto the network. As dummy base-station doesn't have knowledge about the location of motes in the network it will send a notification message over the network.

If base-stations request data from same motes by connecting to different dummy base-stations then we have to adjust network so that data from requested mote also reaches the new dummy base-station. Due to these reasons the performance of the Approach-2 is nearly same for scenario 1, 2 and 3.

Performance of the Approach-2 changes only when base-stations requests data from same motes by connecting to the same dummy base-stations as we already have the network configured to present dummy base-station so only messages used for connection of new base-station to dummy base-station are considered.

7.2.4 Scenario-4

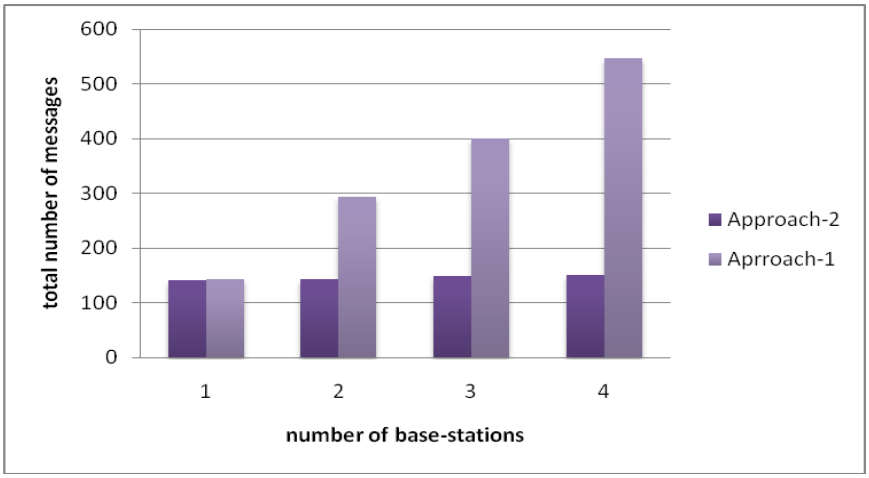

Figure 7.13 Scenario-4 message graph

Scenario-4 is the case where the system runs with different base-stations connecting to same dummy base-station and requesting data from same motes. From the above graph Figure 7.13, we can see that the Approach-2 uses nearly no new messages to re-configure the network but the Approach-1 system uses the same number of messages for each new base-station connecting to network.

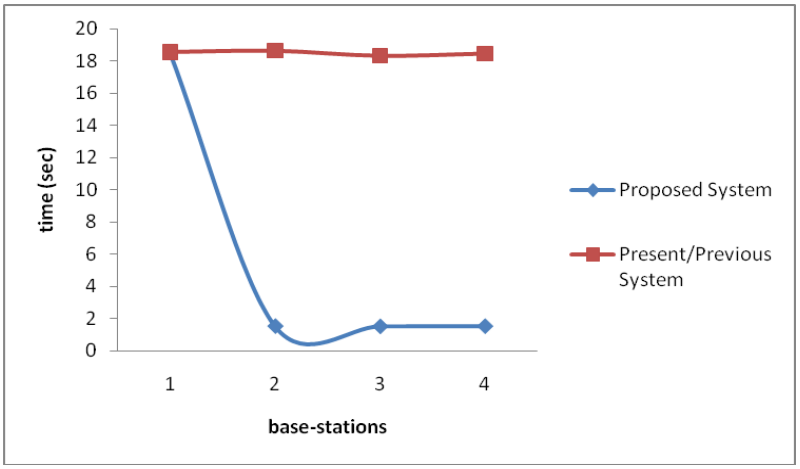

Figure 7.14 Scenario-4 time graph

From the above time graph (Figure 7.14) for scenario-4 we can see that the Approach-2 will use an average of 1.5 seconds for the re-configuration of the network for a request from new base-station as only connection from dummy base-station to the new base-station is to be established. In this scenario whole network is not re-configured or notified about new data acquisition motes as list of motes from where the data is requested by new base-station is same as the list of motes from where old base-station requested data thus achieving the system goal.

7.3 Tests on Length of Data Path

The length of the path chosen by a node to send its data to the dummy base-station was also measured during the testing phase. It was observed that the length of the path used for sending the data depends on the timers present in each mote, which are used to forward the messages after holding them for a random amount of time during the network development phase when a new base-station comes on the network. Both Approach-1 and Approach-2 were compared based on the length of the path used a data sending mote. For a linear (bus) arrangement of the motes, the length of the path used for sending data was same in both approaches.

When the motes on the network were arranged in a mesh topology it was found that the length of path used for sending data by a mote mainly depends on ids of its neighboring motes. It was found that the mote used a shorter path to dummy-base-station when the motes in its neighborhood had similar Ids. The reason for this behavior is that both approaches use timers to forwards message while building the tree and these timers are fired based on their Id along with a random wait where random number(generated by RandomLFSR component) for time is also dependent on Id of the mote. Hence lower the Id of mote, faster is the forwarding of message for building the tree.

It was noticed that whenever a new base-station connects to the network, the position of that dummy base-station is important in both approaches as the tree or network formed will be in the oriented in that direction as the motes present in that direction get message to set itself as child and start its sub-tree development.

For Example:

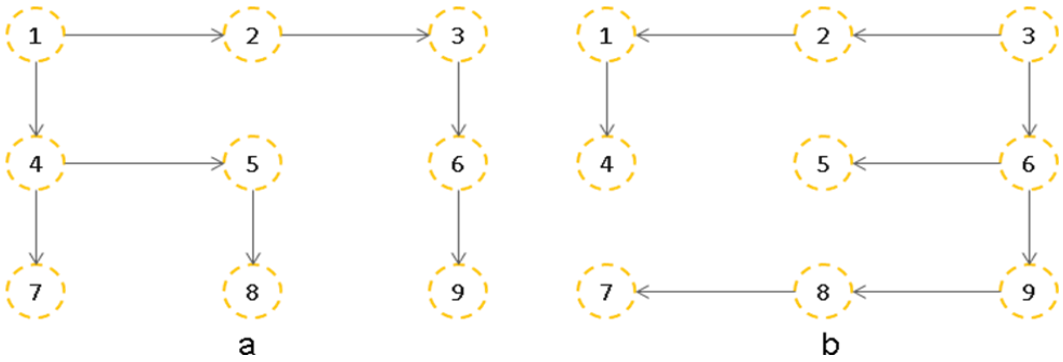

Figure 7.15 a) Network for base-station-1 b) Network for base-station-2

In the above Figure 7.15 a, we can see an instance of the network formed when base-station-1 connects to mote 1 and asks data from motes 8 and 9. It takes 4 and 5 hops for data to reach to the base-station from 8 and 9 respectively. In Approach-1, when base-station-2 connects to the network at mote 3 and requests data from motes 5 and 6, the entire network is re-configured (as shown in Figure 7.15 b) thus taking 3 and 2 hops for data from 5 and 6 respectively to reach base-station-2 but for data from 8 and 9, it now takes 6 and 5 hops to reach base-station-1. In Approach-2, when base-station-2 connects to the network at mote 3 and requests data from mote 5 and 6 network is just configured to get data from 5 to 3 and thus 5 and 6 takes 5 and 2 hops to reach base-station-2 and base-station still receives the data from 8 and 9 using same hop distance. If both base-stations connect to same dummy base-station then the length of the path used to send data will be same in both approaches.

Hence we can conclude that the location of the base-station connecting to the system is important in varying the length of the path used for sending data in both approaches.

CHAPTER 8 - Conclusion

This project aimed at developing a system which detects overlapping paths for data collection in different applications in the network and utilizing that information for efficient data acquisition. This prevents reconfiguring to the entire network of wireless sensor nodes (called motes) for each new application request. The algorithm used in the system was tested for all possible interactions of a data requesting application with network of sensors and was found that the system performance was consistent and efficient in exploiting the overlapping paths in the system for a number of runs. It was also noticed that the systems with Approach-1 used 45% number of messages more than the system with Approach-2 along with higher response time. The goal of exploiting the previously present network knowledge for future re-configuration of the network was also accomplished.

CHAPTER 9 - Future Work

The system with Approach-2 presented can be utilized in future for storing data onto databases for better analysis of the sensory data obtained from sensors over a time period. System can also be used for real-time monitoring with the help of a GUI.

Java application used for interaction with users can also be modified for creating an API which naïve users can use to acquire data from the sensors without having to write lengthy nesC code which can be later on utilized in other applications to be developed in java or in any other programming language.

System can be extended further with visual monitoring and movement monitoring using accelerometers to provide efficient visual surveillance. One can also employ the optimizing algorithms to form new routes in the Approach-2 to reduce the message latency in the system.

References

TinyOS

1) http://en.wikipedia.org/wiki/TinyOS
2) http://www.tinyos.net
3) http://www.sensorsmag.com/networking-communications/tinyos-operating-system-design-wireless-sensor-networks-918

nesC

4) http://nescc.sourceforge.net/
5) http://nescc.sourceforge.net/papers/nesc-ref.pdf

TOSSIM

6) http://www.cs.berkeley.edu/~pal/pubs/nido.pdf

Computer image download from

7) http://laptops2010.blogspot.com/2009/11/apple-mac-laptops.html

TelosB mote image download from

8) http://persnl.cis.ksu.edu/testbed.htm

TelosB with EasySen WiEye image download from

9) http://www.easysen.com/WiEye.htm

EasySen WiEye image download from

10) http://www.easysen.com/WiEye.htm

USB hub image download from

11) http://persnl.cis.ksu.edu/testbed.htm

USB cable image download from

12) http://persnl.cis.ksu.edu/testbed.htm

Plexi glass board image download from

13) http://persnl.cis.ksu.edu/testbed.htm

Velcro image download from

14) http://persnl.cis.ksu.edu/testbed.htm

Wireless Sensor Networks

15) http://en.wikipedia.org/wiki/Wireless_sensor_network

TinyViz

16) http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson5.html

TelosB datasheet

17) http://www.willow.co.uk/TelosB_Datasheet.pdf

EasySen WiEye datasheet

18) http://www.easysen.com/support/WiEye/DatasheetWiEye.pdf

TOS Message Structure

19) http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson6.html