Master in Informatics Engeeniring
Dissertation
Final Report

Wireless Sensor Networks Monitoring Tool

Author:
Augusto Soares
augustos@student.dei.uc.pt

Supervisor:
Vasco Pereira
vasco@dei.uc.pt

4 September of 2013

FCTUC DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Abstract

A Wireless Sensor Network (WSN) typically consists of a large number of nodes performing specific tasks such as measuring or monitoring environmental conditions (temperature, luminosity, humidity, etc.). WSNs once deployed are intended to work for a long period of time. Possible scenarios are the ones where human presence is not possible or encouraged such as battlefield, in military application.

Due to the large number of nodes in WSNs, the management and the maintenance of the WSN is a challenging task. To ease the management of WSNs, a monitoring tool is of utmost importance to perform some management tasks such as visualize nodes resources usage (node status), send control message to the nodes and also to provide a general knowledge of the network as whole.

Beside of monitoring purpose, the monitoring tool can also be used prior to deployment of a new application on the network, for testing purposes, in order to guarantee that all application requirements are satisfied before deploying it.

There are some existing monitoring tools that provide data visualization and also perform control of the nodes on the network. However, none of them provides mechanisms to measure performance related metrics such as packet delay or throughput. This dissertation focuses on WSNs monitoring tools which provide the same features of the other existing tools and also provide mechanism to measure performance related metric.

A new WSN monitoring tool based on Octopus was developed which is able to measure packets delay and packet lost. The information about these two QoS parameters is available in a live fashion and also information about past metrics history.

Keywords

“Wireless Sensor Networks” “Wireless Sensor Networks monitoring tools” “Octopus” “Performance metrics” “new Octopus monitoring tool”

Table of Contents

List of Figures

List of Tables

Acronyms

CISUC	Centre for Informatics and Systems of the University of Coimbra
ETX	Expected Transmissions
FTSP	Flooding Time Synchronization Protocol
GUI	Graphical User Interface
ISM	Industrial Scientific and Medical
LCT	Laboratory of Communications and Telematics
MAC	Medium Access Control
OS	Operating System
QoS	Quality of Service
RBS	Reference Broadcast Synchronization
SQL	Structured Query Language
TCP	Transport Control Protocol
TEP	TinyOS Enhancement Proposal
TPSN	Timing-sync Protocol for Sensor Networks
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
WSNs	Wireless Sensor Networks

1. Introduction

This final report describes the activities and the works that have been done during the first and the second semesters by the author. For his one year dissertation entitled **“Wireless Sensor Networks Monitoring Tool”** to obtain master degree in Informatics Engineering.

1.1. Contextualization

Dissertation takes place at Centre for Informatics and Systems of the University of Coimbra (CISUC), in the Laboratory of Communications and Telematics (LCT), specifically under the LCT-Sense research group. The dissertation is supervised by Vasco Pereira.

1.2. Overview of Wireless Sensor Networks

Wireless Sensor Networks (WSNs) consist of a few or large number small embedded electronic devices which perform monitoring or tracking of physical or environmental conditions, depending on type of sensor coupled to the node it can perform range of measurements i.e. temperature, pressure, sounds, motions etc. These small electronic devices, also known as sensor nodes, are equipped with limited resources in terms of storage, computational resources and limited energy supply.

When the monitored area is relatively small, a few number of sensor nodes is enough to cover the area, in this case, the data collected by each node can be sent directly via radio communication to the central node or Sink node (uni-hop communication). A much larger area may require many nodes, some of them far away from the sink, need intermediate node to relay or forward their packets to the sink node (multi-hop communication). The sink node in turn sends received packet to a specific program running on a computer for visualization or used by other programs.

According to [1] WSNs can be divided in two to categories; Structured WSNs and unstructured WSN. In structured WSNs nodes are deployed in a pre-planned manner depending on the radio coverage and thus can reduce the used sensor nodes and are easy to maintain. On the other hand, unstructured WSNs nodes are deployed in an ad hoc manner and in large number, especially in hostile areas where human presence is not encouraged. Once deployed nodes are left unattended and expected to perform monitoring or tracking during their lifetime with few or no maintenance.

1.3. Motivation

WSNs have many applications such as environmental condition monitoring, military applications, healthcare or industrial uses. Once deployed, WSNs are intended to work for a long period of time without any type of human intervention. Typically, WSNs consists of few nodes to tens or hundreds of nodes or even thousands of nodes depending on size of the monitored area.

Nodes on the network are severely limited in terms of computational power, energy supply and storage and they communicate via noisy wireless links. To assure the proper use of this resources and the link quality a monitoring tools is necessary to visualize the node's resources usage and also communication among the nodes.

While some existing WSNs monitoring tools provide data visualization and provide mechanism to perform basic control of the node in the network none of them really concentrate on measuring performance metric of the WSNs. WSNs have many application, and different application has its own requirements. WSNs use in critical areas such as in military, healthcare and industrial scenarios require predictable performance related metrics such as packet delay, link throughput and message reliability.

WSNs monitoring tool with ability to measure performance related performance is necessary in WSNs application where Quality of Service is the main concern. This tool can be used prior to deployment of new application on the network to guaranty that the new application requirements are satisfied.

1.4. Goal

The main goal of the dissertation is to study and explore existing WSNs monitoring tools or to create a new one that provides the following features:

- Data visualization - visualize the data collected from the WSNs network i.e. sensor reading data.
- Allow user to perform basic control of the nodes in the network i.e. setting or altering sensor sampling period, control nodes duty cycle, etc.
- Provides mechanism to analyze and calculate performance related metric when Quality of service is the main concern of the WSNs application. I.e. packet delay, link throughput, massages reliability, etc.

1.5. Work done and objective fulfilled

In summary, the work done during this dissertation was studying existing WSNs monitoring tool and elaborate a new WSN monitoring application which is also capable of measuring the performance related metrics. The new WSNs monitoring tool developed is based on the existing Octopus monitoring tool. Time synchronization protocol was incorporated to it for time-stamping events and packet delay measurement and a MySQL database is used as the main storage for the new tool rather than logging events to a flat file as done in the previous Octopus. Other advantage of using a database is to allow users to find events of interest easily, and allow the retrieval of events and node history. The Octopus tool available only permits for the visualization of current events and data from WSNs network.

The work done fulfilled the main goals of the dissertation which was to create a new monitoring tool that was able to measure performance related metrics. The two metrics related measure by the new Octopus is the packet delay and the Packet lost. Besides monitoring live events, the application also provides features for retrieving information related to these metric for the past events and data.

1.6. Document structure

The rest of the document is structured as follow; chapter 2 presents the state of the art, which discuss about existing WSNs monitoring tools. Chapter 3 discuses the requirements of the WSNS monitoring tool to be develop. Chapter 4 presents Octopus in details the experimental setup of octopus in TinyOS [2]. Chapter 5 discusses in detail the alterations made to Octopus and the implementation of the new features and functionalities of the new Octopus. The work plan for the first and second semester, their list of activities and respective durations is discussed in chapter 6. Conclusion if found in chapter 7 and references are available in chapter 8.

2. State of the art

This chapter discusses the state of the art of WSNs monitoring tools by analyzing existing WSNs monitoring tools stating their main features and a comparison summary of these tools also is provided.

2.1. WSNs management overview

WSNs typically consist of many small wireless devices which equipped with one or more sensors to perform specific tasks such as measuring or monitoring environmental conditions (temperature, luminosity, humidity, etc.). These small wireless devices perform the sensing and the sensed data is sent via radio communication to a sink node, which in turn forwards it to the monitoring application typically running on the Computer to be processed and visualized to the user.

The nodes in the WSNs are limited in terms of storage, energy supply and computational power. To better utilize these limited resources of the nodes, a mechanism to monitor these resources is utmost necessary for the management of the WSNs and the performance of the WSN as a whole. Beside for monitoring the nodes resources the management application can also use to perform testing prior to deploy a new application on the WSNs.

According to [3] WSNs management systems can be classified in: central vs distributed control, and reactive vs proactive monitoring. In a centralized fashion all the collected data is sent to a sink node which in turn forwards it to a central manager which more powerful resources that can perform complex management tasks, reducing the processing burden on resource constrained node. The advantage of this approach is that the central manager has a global knowledge and thus it provides more accurate management decision. However this approach suffers the scalability issue and also implies a single point of data concentration and a potential failure. The distributed control was used to overcome the problems presented by the centralized control by using multiple sink nodes, each of which can be used to control portion of the network and may communicate with the other sink nodes to perform management functions. It solved the problem of single a single point of data concentration but it considered complex and difficult to manage. All the monitoring tools discuss later in this document fall into the centralized monitoring category.

Based on the approach used for monitoring and control, WSNs monitoring tools can be further divide into four categories:

- Passive Monitoring. System collects the information about the network for post-mortem analysis.
- Fault Detection Monitoring. The system collects the information of the network states to identify whether a fault has occurred.
- Reactive Monitoring. System collects the information of the network states to identify whether an event of interest have occurred and adaptively reconfigure the network.
- Proactive Monitoring. The system actively collects and analyses network states to detect past events and to predict future events in order to maintain the performance of the network.

2.2. Available WSNs monitoring tools

In this section the available WSNs monitoring tools are described along with their functionalities.

2.2.1. Mote-view [4]

Mote-View is a scalable software framework from crossbow technology for visualization and monitoring WSNs. According to Mote-view designers a typical WSNs consists of three tiers (Mote tier, Server tier and Client tier). Mote tier runs embedded firmware (TinyOS [2]) and sensor hardware's drivers, the server tier provides logger and a data base for the data obtain from the sensor nodes. Mote-view belongs to the Client tier that provides Graphical User interface for visualization of the sensor data.

Beside of visualizing the live individual sensor data, Mote-view can log sensor data to be visualized in later time. The tools also allow collecting the sensor reading data from several sensor nodes for graphing or comparison purposes.

Since Mote-view is a modular framework, modules within the Mote-view client split in to four layers, each layer in turn provides plug-ins that can be extended.

Mote-view currently only supports the crossbow products, i.e. the crossbow sensor boards and crossbow sensor motes.

Mote-view features:

- Visualizes the network topology and network statistic and also logging of sensor readings data and viewing the logged data.
- Displays live sensor-reading data and also plot the data over the span of time.
- Node color changes if no reading from the node during a specified interval.
- Color-coding note health can be used for other metrics such as throughput, Bandwidth and success/yield data packet received.
- Provides statistic function includes end-to-end data packet yield and a prediction for the feature and the RF link quality.

Below is an image of Mote-view user interface and several options for displaying data gathered by the application.

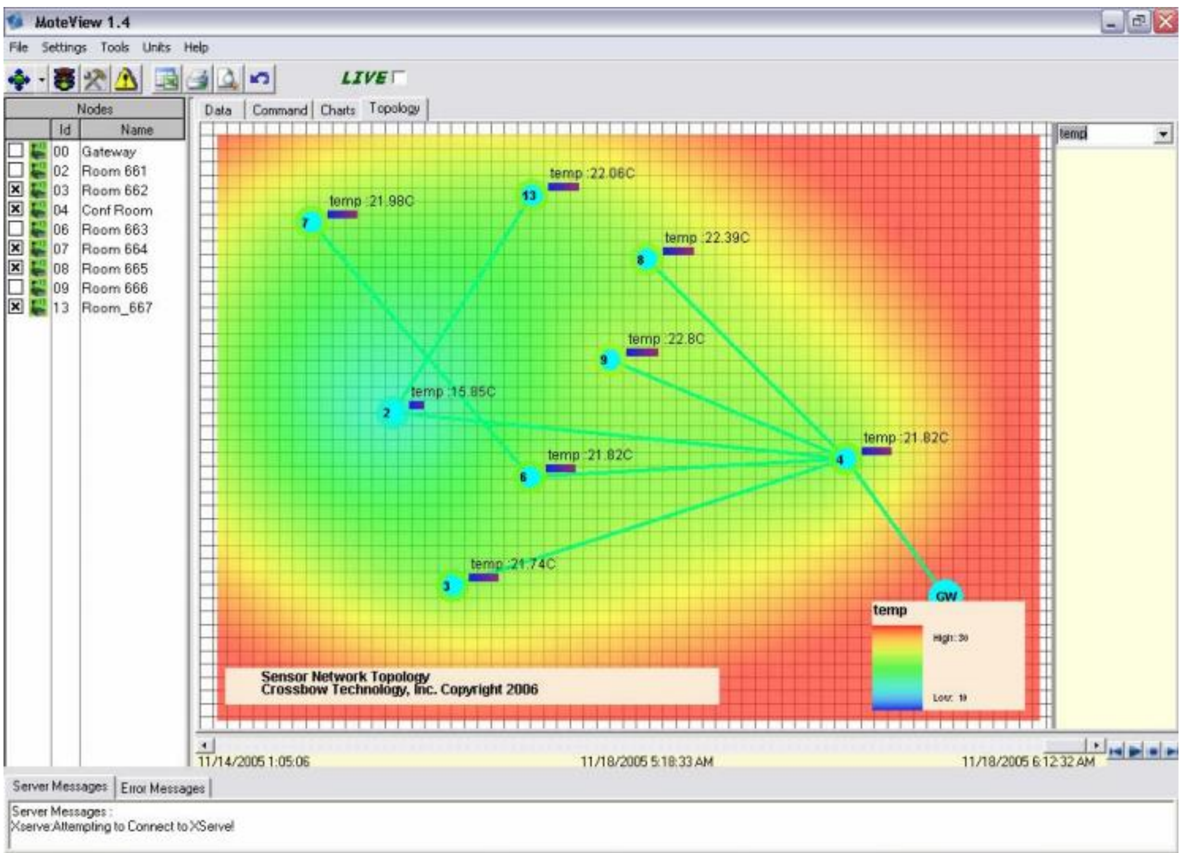

Figure 1 – Mote-view main screen [5]

The application provides a list of nodes on the network on the left column and several visualization options such as: “Data” which shows sensor reading for each node on the network, “Command” this tab allows user to control node’s LED light by turning it on or off. The Chart tab allows user to choose up to three nodes to plot their sensor reading value (temperature, light and Luminosity). Lastly, the image above is currently showing the logical network topology using its Topology tab. As shown above, nodes are shown in circle and lines to represent the link among them. Beside of the nodes and the links among them, user can also choose which sensor reading value to be displayed for each node.

2.2.2. Spyglass [6]

Spyglass is modular framework for monitoring WSNs, it uses several plugins to display various data such as node position, sensor reading data and network topology. The plugins can be extended to support more functionality.

Spyglass consists of three major functional entities: The sensor network, the gateway nodes located in the sensor network and the visualization software.

Data from the network is forwarded to the gateway node which forms a ring buffer to store the data packets, the application running on the gateway listen to a certain port for incoming visualization request from the visualization PCs via a TCP connection.

The visualization software consists of 3 components: graphical display canvas, tree-structured textual information of the network as a whole and line-based output for debugging purposes.

Spyglass functionalities:

- Displays the temperature and the relation between nodes and network and also allows recording activities in certain point of time for later playback in different speed (normal, fast forward and jumping to a certain point in playback).
- Uses plug-ins to paint on the canvas in its own layer via drawing primitive in the SpyGlass can avoid conflict between plug-ins and plug-ins can be extended as needed.
- Currently SpyGlass supports the several plugins to display the nodes and the network: temperature map plug-in, display node plug-in, the battery plug-in, topology plug-in.

SpyGlass user interface shown below can be grouped in three main categories: main windows on the upper left for displaying the logical network topology, right column is used to display tree-structured text information of the selected node (i.e. node position, sensor reading value). The lower windows show text-based output for debugging purposes.

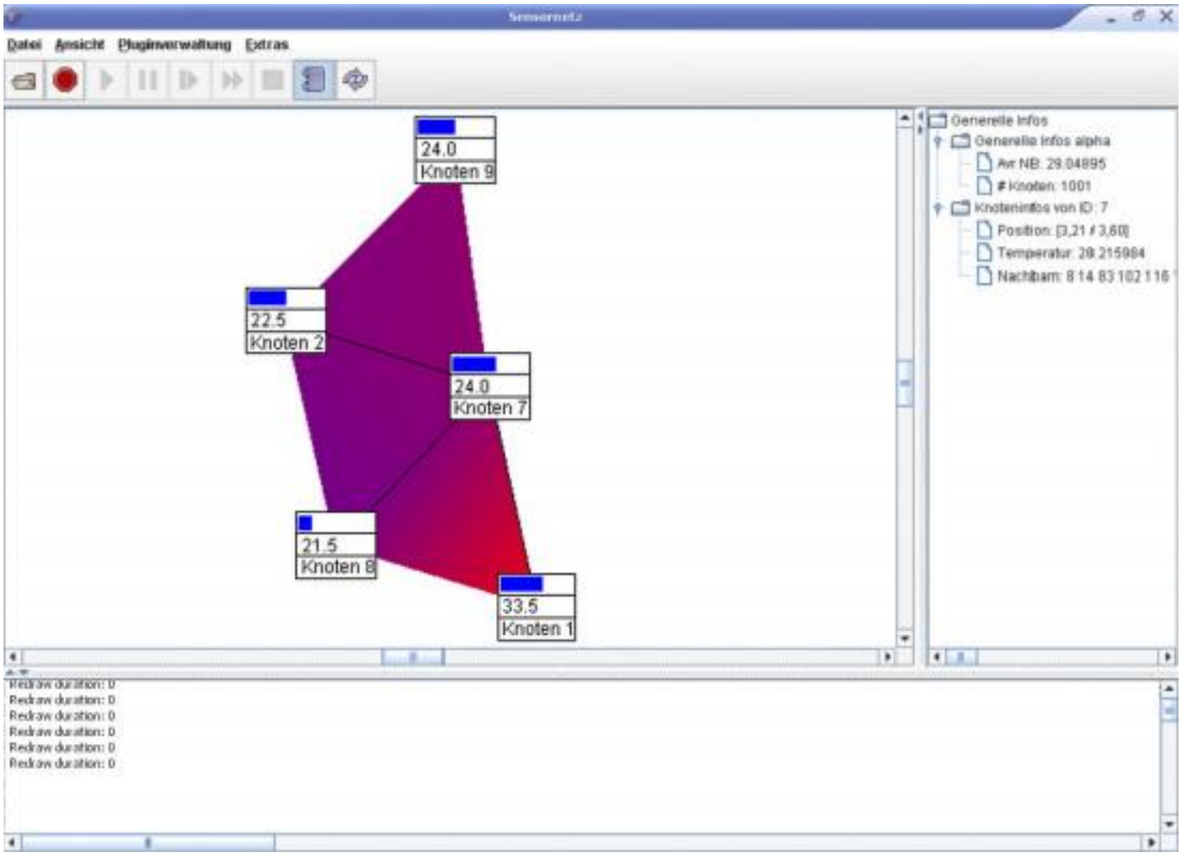

Figure 2 - SpyGlass user interface [6]

2.2.3. Octopus [7]

Octopus is WSNs monitoring tools which is independent of the routing protocol used in the WSN. It provides live information about the network topology and the sensor reading data. It also enables reconfiguration of the network and of the application by sending short commands to the sensor nodes over the air.

Octopus is both developed in nesC for the nodes (TinyOS 2) and Java in the application side that runs on the computer.

Octopus reconfiguration commands can easily extended such as modifying the sampling period, setting the duty cycle, changing between query, event or timer driven modes.

Octopus consists of 3 entities: GUI, gateway node and the emote nodes. GUI is a java based application serve as frontend of Octopus which displays live network topology, it connects to the gateway node via serial port on the host computer. The embedded portion of octopus relies on three main component octopus core, timer and sensor.

Octopus functionalities are:

- GUI for visualizing live sensor network topology
- Supports time, query, or event base driven sensor network.
- Allow user to broadcast or send control commands (sampling period, radio duty cycle, sensing thresholds) to specific node
- Allows generating alerts when the sensor reading data exceed user predefined threshold.
- Network chart is used to provide live data plots.
- Logs all the data received in a .csv file for post or further analysis.
- Enables developer to customize or extend GUI through modular APIs.

Octopus user interface below (the latest version) was taken from the experiment conducted in the first semester because the application was constantly improved and the new features were not available in the paper published by the authors.

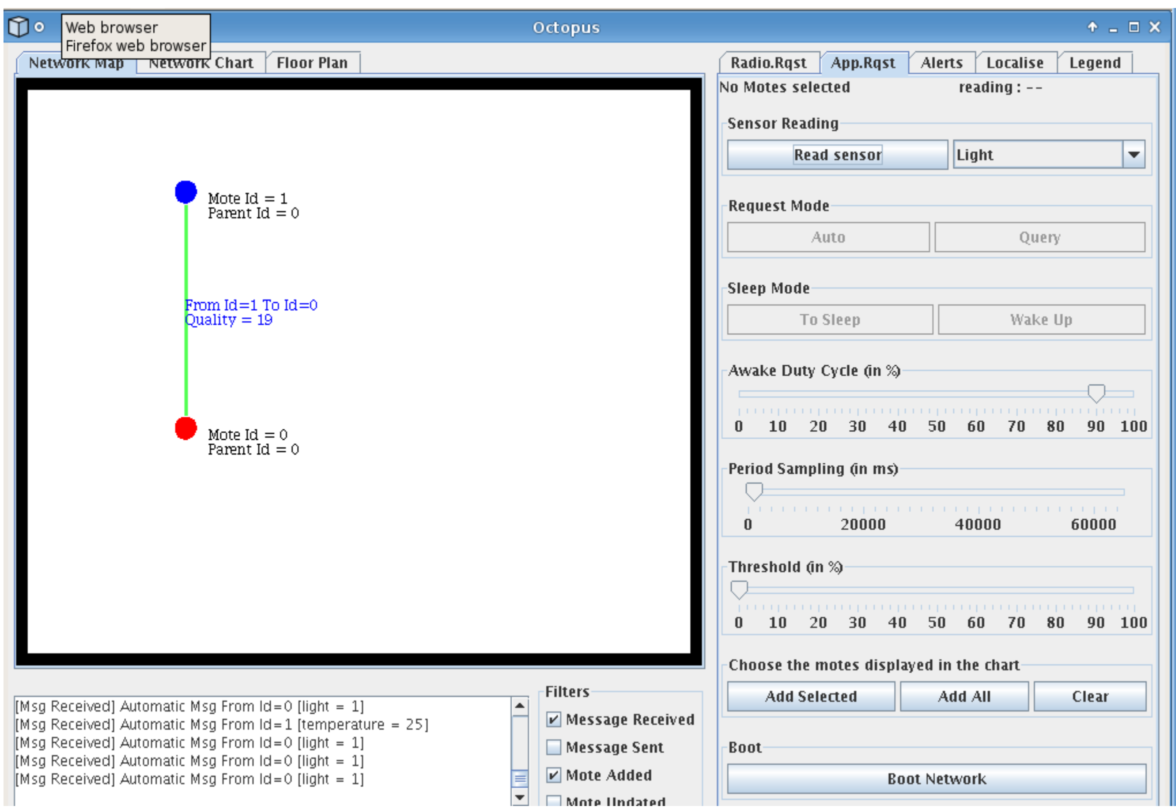

Figure 3 - Octopus user interface

Octopus user interface consists of several tabs that provide the features described above. "Network Map" displays the logical network topology, nodes are shown along with the lines representing the connection among the nodes, Sink node is colored red and the regular nodes are in blue. The "Legend" tab on the right can be used to show more information about node states and network topology (i.e. link quality, node's sampling rate). Users can choose any or all nodes displayed on the topology to be controlled by sending them the control message either from "Radio.Rqst" or "App.Rqst" tab. Also, the selected nodes can be added to plot their sensor reading values. The chart can be consulted in the "Network Chart".

"Floor Plan" allows user to upload an image of the floor, this is done in the "Localise" tab on the right, and after uploading the image the user needs only dragged the node to its specific location.

User can be alerted when an event of interest happens, an Alert can be setup in the "Alert" tab, and alert is implemented as pop-up windows that will be shown when the interest event has happened. (I.e. user is notified when the sensor reading value is beyond or below a determined threshold).

Most of the control message is send using the "App.Rqst" tab, the target nodes can be one or more, target nodes can be selected either in the "Network Map" or in the "Radio.Rqst". As shown above user can choose which sensor reading value to be displayed by console panel on the lower window, put node into sleep or awake mode, settings duty cycle period and add the nodes to plot their sensor reading values.

2.2.4. MonSense [8]

MonSense is modular WSNs monitoring framework for deploying, monitoring and controlling WSNs. It supports connections to various WSNs and also can interface with one or more gateways that forward information received by attached network.

The current implementation of MonSense in TinyOS and the telosB hardware platform, it targets two different user types: WSNs Customer and WSNs researcher. The application is used by the WSNs researchers to plan and deploy WSNs since they have deep understanding of the ad-hoc routing protocols, sensor hardware platform and the nesC code that runs on the devices, while the same application is used by WSNs customer for monitoring and controlling purposes.

MonSense functionalities are:

- For the deployment purpose Mon sense displays map of the testbed field and using the GPS to assist the deployment agent to select exact location of the node.
- The connections or routes among the nodes is represented by colored line for different connections parameters such as; the node is active or connected to the network, is node connected to the base station, is the node just send a packet, etc.

colored line is used to display the status of the network and facilitate the rapid perception of the entire network.

- Application allows control commands to be sent to the nodes, i.e. command to calibrate the sensors on the nodes.

MonSense user interface shown below consists of list of the nodes and map of the testbed and using GPS to locate nodes precise location.

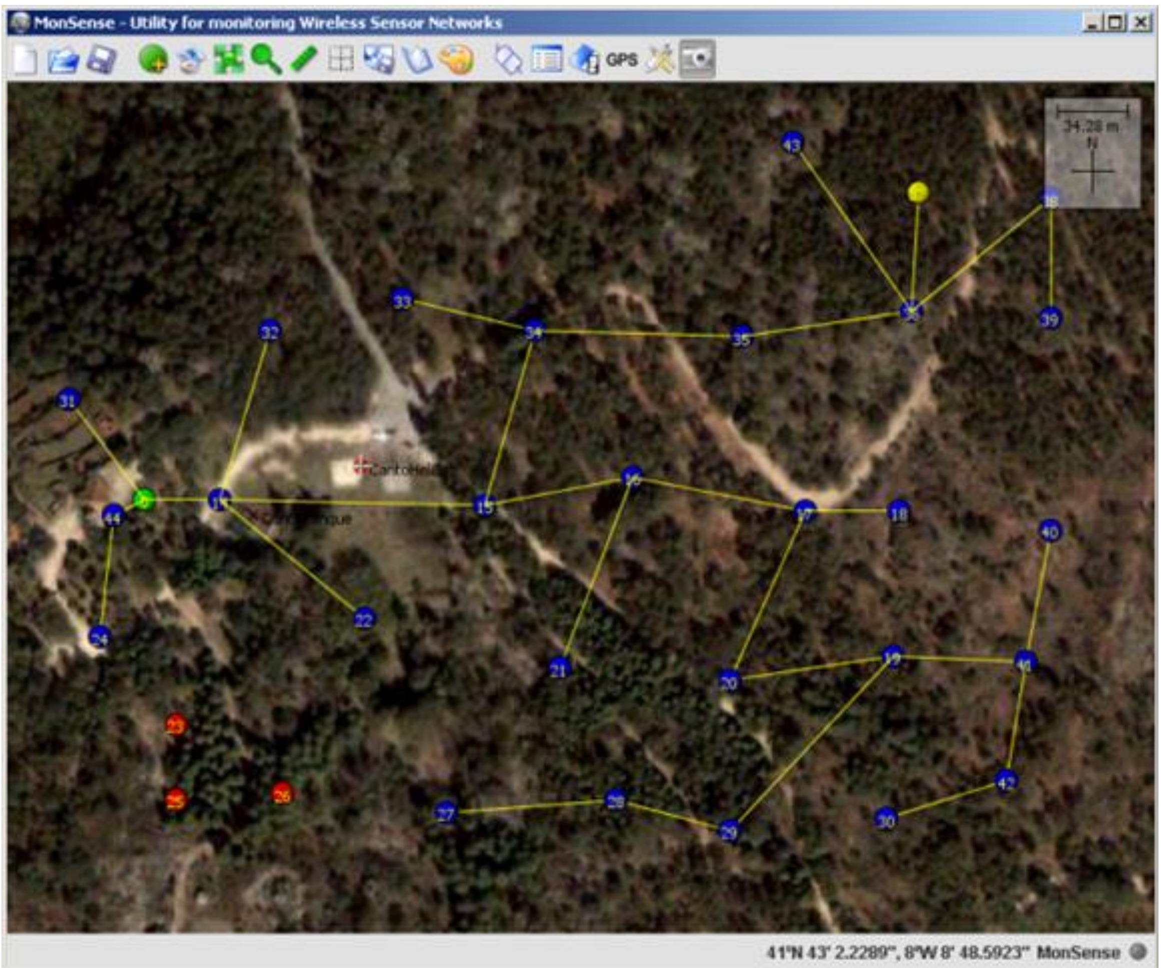

Figure 4 - MonSense user interface [8]

AS shown on the above image colors are used to describe connection among the node on the network: Nodes appear in red are node that have lost connection to the network, nodes in yellow means a network message has just been received from this node, nodes in green are nodes that are connected to the base station, blue nodes are nodes thats connected and heve sent a message recently.

2.2.5. NanoMon [9]

NanoMon is modular and flexible WSNs monitoring software, it offers flexible architecture and supports for various user requirements of sensor network applications in an adaptive manner.

NanoMon consists of following functional parts: wireless sensor networks, monitoring servers, databases and monitoring clients. Information generated by the WSNs nodes is forwarded to the monitoring server via one or more sink nodes trough serial connection, the monitoring server in turn forward the information received to the monitoring clients via TCP/IP connection for visualization and also forward to the databases for the later queries.

Software in NanoMon is comprised of 5 modules: connection module is responsible for managing the connection between the NanoMon and sink nodes also process the information request from monitoring devices. Packet processing Module parses the packet from received from the connection module to a well-defined packet (XML packet formats) for sending to monitoring clients and also sends the parsed to the management module to store in the database. The management module provides data access interface to and from the database. Sensor network abstraction module contains information of the sensors, nodes and the connection between them and allow user to choose between the live mode and the past information generated in a specified period. GUI integration module creates a GUI framework and integrates GUI plug-ins defined in the configuration file.

Users can flexibly scale and change the operation of NanoMon according to its specialized sensor network application requirements by using the configuration file.

NanoMon functionalities:

- Topology plug-ins for visualizing sensor node and the connection between them.
- Sensor data plug-in displays sensing value from the sensor node, data reception time and also searching particular sensor data according to input condition.
- Chart plug-in visualizes the sensor data using different types of charts.
- Sensor list plug-in visualizes live sensor nodes and their details information such as battery level, install location, ID and last sensing value.

Image below depicts NanoMon configuration file and the NanoMon user interface. The user interface can vary for different scenario of usage. User is free to choose modules and plug-ins to load according to his needs. All these choices are configured in the configuration file, prior to loading the user interface.

```
<?xml version='1.0'  encoding='euc-kr'?>
  <nanomon>
    <network name="Home Monitoring">
      <sensor caliclass="etri.data.cali.TempCalibration" name="Temperature Sensor" unit="C" value="1"></se
      <sensor caliclass="etri.data.cali.NullCalibration" name="Light Sensor" unit="LUX" value="2"></sensor
      <sensor caliclass="etri.data.cali.NullCalibration" name="Humidity Sensor" unit="%" value="3"></senso
      <sensor caliclass="etri.data.cali.NullCalibration" name="Infrared Ray Sensor" unit=" " value="4"></s
      <sensor caliclass="etri.data.cali.DecimalCalibration" name="Gas Sensor" unit="PPM" value="5"></senso
      <sensor caliclass="etri.data.cali.NullCalibration" name="Ultra Sonic Sensor" unit="cm" value="6"></s
      <battery caliclass="etri.data.cali.BatteryCalibration" unit="V"></battery>
      <plugin name="etri.main.chart"></plugin>
      <plugin name="etri.main.sdata"></plugin>
      <plugin name="etri.main.snode"></plugin>
      <plugin name="etri.main.topology"></plugin>
      <serial baudRate="19200" dataBits="8" flowIn="None" flowOut="None" parity="None" port="COM6" stopBit
      <database location="220.123.0.152" user="admin" passwd="1234"></database>
    </network>
    <network name="Parking Lot Monitoring">
      <sensor caliclass="etri.data.cali.NullCalibration" name="Infrared Ray Sensor" unit=" " value="4"></s
      <battery caliclass="etri.data.cali.BatteryCalibration" unit="V"></battery>
      <plugin name="etri.main.park"></plugin>
      <plugin name="etri.main.sdata"></plugin>
      <plugin name="etri.main.snode"></plugin>
      <serial baudRate="19200" dataBits="8" flowIn="None" flowOut="None" name="COM1" parity="None" port="C
      <database location="220.123.0.152" user="admin" passwd="1234"></database>
    </network>
```

Figure 5 - NanoMon configuration File [9]

As show in the image above, all the required components and plug ins must be defined on configuration to use for later in the UI, i.e. type of sensor, unit calibration and the some plugins to load for the user interface and other information that will be provided by this session user interface. Following is the correspondent User interface for the configuration file above.

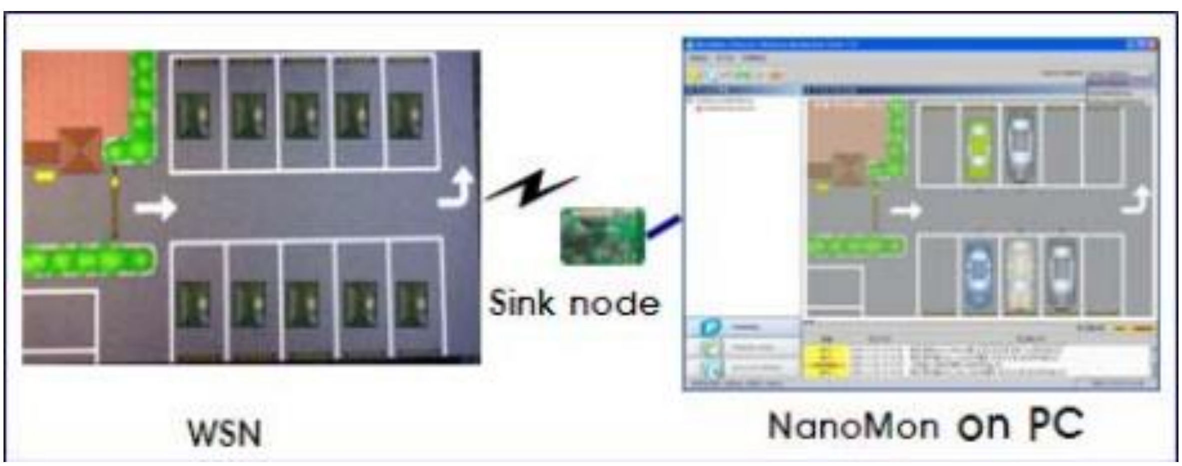

Figure 6 - A testbed and the NanoMon UI on the right [9]

2.3. Comparison table of WSNs monitoring tools

Monitoring Tools	***Data visualization and controlling***	***Live visualization and data logging***	***Data retrieval method***	***Type of data displayed***	***Generate Alerts***	***Source Code availability and extension***	***Supported Mote Platform***	***Operating System***
Mote-view	Data visualization and controlling	Live and data logging	Event and query driven	Nodes and network	yes	No	Mica series	TinyOS
Spyglass	Data visualization	Live	Time - driven	Nodes and network	no	No	Embedded Sensor Board ESB 430/2	C language
Octopus	Data visualization and controlling	Live and data logging	Time-driven and query-driven	Nodes and network	yes	Yes and support extension	TelosB, Mica Family and Tyndall 25	TinyOS
MonSense	Data visualization and controlling	Live and data logging	Time-driven	network	yes	no	TelosB	TinyOS
NanoMon	Data visualization	Live and data logging	Time-driven and query-driven	Nodes and network	No	no	Nano-24 ETRI-SSN MicaZ	NanoQplus [10]

Table 1 - Comparison of WSNs monitoring tools

Above table compares the functionalities provide by the WSNs monitoring tools discussed in this chapter. This comparison is necessary to determine which tool is offers more functionality by analyzing the following characteristics:

- *Data visualization and Controlling* – This functionality determines if the tool is only used to visualize the gathered data by the nodes on the network, or if it is also able to control the network. In the first case data flow only occur in one direction, from the WSNs to the Monitoring application running on the computer. On the other hand if besides data visualization, the tool also provides mechanisms to the user to control nodes on the network (i.e. change settings - altering Sensor sampling period, node duty cycle, and query the specific data from the nodes).
- *Live visualization and Logging* – Apart from visualizing the event that is happening in real time, the logging functionality is necessary to save the events for later analysis and also to consult an event that happened in the past.
- *Data retrieval method* – This functionality refers to the ways the collected data, node status, are send to the monitoring tool; "Time-driven" data are sent in a periodic fashion to the monitoring tool. "Query-driven" data retrieval method allows user to query at any time the data or information from the network whenever they desire.
- *Type of data displayed* – This characteristic is used to distinguish monitoring tools by the information that the tool provides. Most monitoring tools as shown in the above table provide data for both node and network level whereas MonSense primarily focus on the network level information as whole.
- *Alerts* - This functionality refers to the ability of the monitoring tool to notify a user when an event of interest has happened.
- *Source code availability and support of extension* – while most of the tools claim to support extensions, allowing the adding of more functionality to the tool, not all of them have source code available. Mote-view is a case of proprietary software that only supports only Crossbow products.
- *Supported mote Platform* – There are reasonable number of mote platforms available. The more platforms it supports is better. Even when some tools claim to be easily ported to a new mote platform, this can be quite difficult when the applications source code is not available.
- *Operating system* – the applications running on the nodes are programmed in a sensor operating system environment. Some operating system provide libraries to be used by other programs to communicate with the application running on the sensor node (i.e. TinyOS provides libraries for C, java and python program to communicate with the node's application)

2.4. Critical analysis of the monitoring tools

WSNs monitoring presented above were designed to use mainly for data visualization and to collect the network status (quality of the link between adjacent nodes) and nodes status such as energy level and sensor reading values. Mote-view, Octopus and MonSense provide a mechanism for the user to preform basic controlling such as query distinct sensor reading values and set or alter the sampling rate of the sensor node, the other tools only used for data visualizing or passive monitoring.

None of the existing tools provide a mechanism to calculate performance related metrics such as packet delay and throughput. The monitoring tool to develop will include the mentioned metrics and others to allow calculation of the performance.

The monitoring tool to be implemented in this dissertation will target WSNs in critical areas where performance of the network must be measured. Thus the tool must provide functionalities to measure the performance related parameters in real-time fashion.

3. Monitoring tool requirements

This chapter discusses the requirements of a WSNs monitoring tool that targets controlled performance networks. The monitoring will imply the display of the collected data and related performance parameters in real-time.

WSNs have been used in many areas such as environmental condition monitoring, health care, military applications and industrial plants. Different applications of WSNs require different requisites such as communication overhead, packet delay and task execution time, depending on the area where WSNs is used.

To assure the performance of the WSNs in the critical areas, a WSNs monitoring tool can be used to measure performance related parameter such as packet delay and the packet loss rate. Unlike most of the WSN monitoring tools discussed in the previous chapter, which only provide network monitoring and data visualization, the intended WSNs monitoring tool should also provide mechanism to predict the network performance by measuring some of the QoS parameters such as packet delay and packet lost.

The monitoring tool targets WSNs with controlled performance which allows network performance analysis in real-time fashion.

The monitoring tool In general, will provide following functionalities:

- Display the status of nodes in the network;
- Display data collected by the nodes;
- Display the logical network topology;
- Allow users to perform basic control of nodes in the network;
- Generate alerts based on the user predefined values;
- Allow the access to the history of each node.

Each of these functionalities will be described in more detail in the following sections.

3.1. Display network statistics, nodes status and the sensed data

The main task of the monitoring tool is to visualize the logical network topology, the status of nodes in the network and the sensed data. This is mainly important because it's the only way for the user to check the node status, node resource usage (such as battery level, memory usage) and data collected by the nodes (sensor reading values). The data collected by the nodes is displayed on the tool and also stored in the database for the later analysis.

Apart from obtaining the node status, the tool should also show the logical network topology, which displays all nodes and the interconnection (links) among them. This approach gives the user real-time network status and helps to determine if a node is active in the network.

A link between the nodes is used to determine the node connection, a node is active when there is a link from the node to the parent node otherwise the node is inactive (inactive node will be displayed without link).

3.2. Perform basic control of nodes in the network

There are times when the user needs to configure or alter node configurations such as sensor sampling period and setting node's duty cycle. This is necessary to avoid reprogramming the node just to alter a configuration value.

Node lifetime is dependent on the energy available in the node, since nodes are using limited energy supply (battery powered). Enabling node duty cycle can help to preserve energy on the node. Duty cycle allows a node to be in a quiescent estate to preserve energy.

For the flexibility reason control commands can be sent either in broadcast form or to specific node because sometimes control command is intend for all nodes on the network i.e. setting or altering sensor sampling period, whereas duty cycle control commands usually intended for specific node or group of nodes.

3.3. Performance related metrics

WSNs used in critical areas such industrial area, healthcare and military the network behavior must be predictable in order for some application to function properly. The monitoring tool can and should be used to measure the network performance in order to guarantee that the performance is the expected.

Following is the list of chosen metrics to measure, which will be able to determine the performance of the network:

- **Packet delay**. The packet delay is calculated by subtracting the “packet receive time” and the “packet send time”.
- **Packet lost.** The packet lost is calculated by subtracting number of “packet received” and the number “packet sent” during a period of time.
- **Packet delivery rate**. Packet delivery rate is calculated by the number of “receive packet” over time interval of one second.
- **Freshness [4].** Freshness is use to determine the node status and the last packet received from the node. This is useful to keep track of active node and can generate alerts when packet from a specific node has not been received beyond a defined time interval.

In order to perform the measurements of the above metrics the time synchronization is utmost important. The nodes time, the sink and the monitoring tool time must be synchronized to obtain correct results of the measurement. Apart from time synchronization the monitoring tool must provide a mechanism to log the history events or packet.

3.4. Other features that monitoring tool needs to provide

Beside than display network statistic, node status and allow user to perform basic control of nodes, the monitoring tool needs also mechanism to store the history values for post analysis and graphing purpose. The database is preferred to store these data rather than a flat file because it is easy to retrieve data of interest from the database via queries rather than analyzing the file manually.

The monitoring tool ability to generate alerts is necessary to alert the user of interest event has occurred or when the data collected by the nodes on the network exceed the threshold set by the user. The alert message will be displayed on the PC running the monitoring tool.

3.5. Definition of a standard interface with the sink

The monitoring tools will be divided into mote application and the PC application, the mote application as the name imply will be install on the node on the network, mote application is responsible for sending the data collected by the node to the PC application for visualization and also receive the control commands send from the PC application by the user. The mote application will be developed in TinyOS.

The PC application is the GUI application which displays the network statistics, nodes status and allows user to send control commands to the nodes on the network. The PC application will be interacting with a database to store and retrieve data. The PC application will be developed using high level programming language such as java which for the platform independent to be able to run in any operating system.

The communication between the PC application and the mote application will be done via TinyOS java tools. The message format and the field of the message will be configured in the mote application and the PC application.

3.6. Extending Octopus

After analyzing some of the existing WSNs monitoring tools discussed in the previous chapter, this chapter specifies the requirements of a WSNs monitoring tool with the ability not only for data visualization and to perform basic control of nodes on the network, but also that is able to measure network and node level performance.

Most of monitoring tools requirements specified in this chapter are a combination of the common features of existing monitoring tools which are considered essential and provided by all WSNs monitoring tools such as discussed in section 3.1, 3.2 and 3.4. While these commons and essential features are good to have, they are not enough for the purpose of the monitoring tool intended by this dissertation, which is developing a monitoring tool which is able to measure performance related metrics.

A comparison was made for the monitoring tools in section 2.3 of the previous chapter. This was done to compare their features and the possibility to extend an existing tool to leverage their features and add new functionalities according to the goal of this dissertation. With this in mind, comparison table was made along with the criteria of the comparison to decide which existing monitoring tool to be chosen as the base of the new monitoring tool.

By analyzing the comparison table the choice becomes clear. Octopus was chosen because it satisfies most the comparison criteria compare to other tools. Following is the description how Octopus feats those comparison criteria:

- Octopus provides data visualization as the other tools and also allows user to control nodes on the network, Octopus allows to control nodes for variety of tasks (i.e. setting, altering node's sampling period, node's duty cycle, changing radio communication channel, etc.) compare to other tool such as Mote-view which only allows to perform basic control i.e. controlling the node's LEDs.

- Octopus supports several motes platform as shown on the comparison table compare to other which either tested only for specific mote platform or limited to a company product (i.e. Mote-view only supports Crossbow products)
- Source code availability and support for extension, Octopus source code is made available by their developer under a GPL license for other developer to use and modify according to their need, furthermore Octopus made used of the protocols and the libraries which included actively maintained by TinyOS developer.

Octopus as discussed in the state-of-art chapter already provided some of the requirements listed in the previous section 3.1, 3.2, 3.4 and 3.5. Extending Octopus so it can measure performance related parameters requires adding new features i.e. *packets delay calculation, packet lost rate and the node freshness* or it last known activity. These three parameters are can be used as main factors to measure network performance in general:

- Packet delay – In some WSNs applications scenarios i.e. industrial area where WSNs are used for controlling and automations, the packet delay must be in a tolerable range to guaranty that actions are taken at timely. (I.e. commands send to nodes in the network must be arrived between expected times so thus corresponding actions can be carried out at the expected time)

- Packet Lost – In A WSNs with controlled performance, this parameter can be as restricted as packet delay described above where packet lost must be in a certain range to consider that the networks is perform as expected. Apart from being a metric used to determine network performance, packet lost is also related to the nodes energy efficiency because when packet lost rate is high can also deplete the energy supply by packet retransmission depending on the packet retransmission policy use.

- Node Freshness – This factor is not strictly related to the network performance in general, this was added because of fact that nodes on WSNs are prone to link failures because of unreliable wireless characteristic one cannot know for sure when the node is active or inactive for certain period of time in the past. This value simply contains date and time of the last packet received from the node.

4. Experimental setup of Octopus

This chapter describes in-depth the Octopus monitoring tool and also the experimental setup of Octopus in TinyOS. The purpose of this experimentation was to familiarize with the Octopus source code, explore the octopus features and its limitation in small testbed of WSNs with two nodes (a sink node and a normal node) and exploring the possibility of using the Octopus as base tool for the monitoring tool to be developed. Octopus was chosen for the experimentation because among the monitoring tools discussed in the state of art chapter it poses some of fundamentals functionalities require to monitor WSNs and also its source code is GPL licensed which can be freely used or modified.

4.1. Global Architecture of Octopus

The image bellow shows the connection of the three components used in Octopus, the regular mote is referring to the nodes in the network that collect the data and send it to the Gateway node (sink node) via radio. The Gateway, in turn, forwards the data received to the application running on PC (GUI) using serial communication since the Gateway is connected to a serial port on the PC. The configuration commands from GUI application are sent via the serial port to the Gateway node and the Gateway either forwards the command to a specific node or broadcast the command to the all nodes in the network depending on the specified destination address by the user.

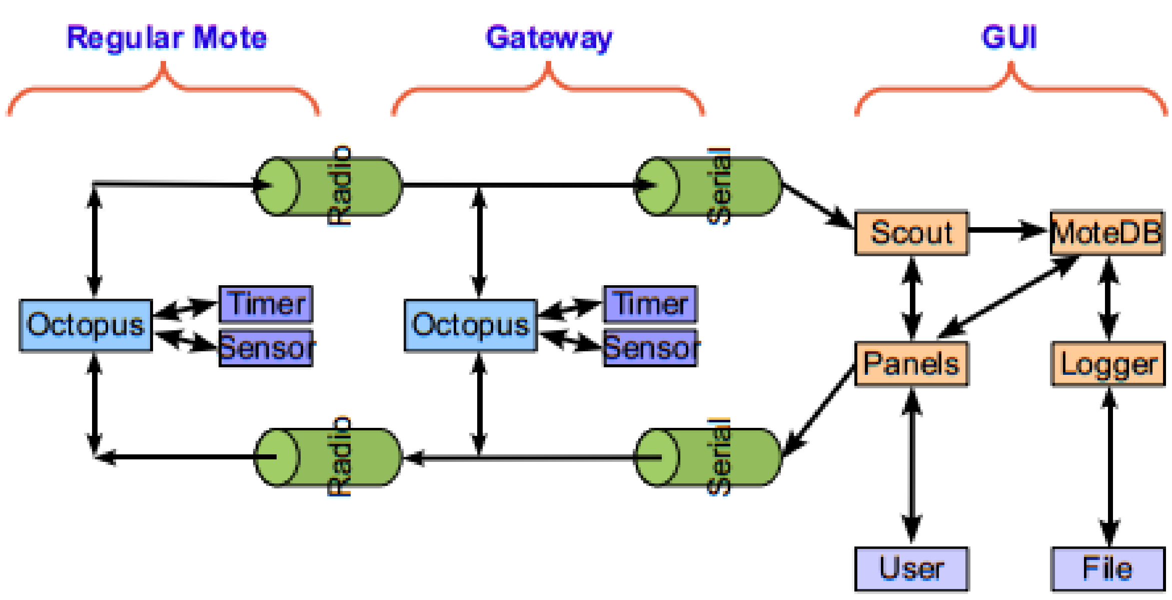

Figure 7 - Global Octopus structure [11]

As shown in the above image embedded application of Octopus relies on the Timer

The Octopus GUI comprises of four components: Scout, Mote Database, Panels and Logger. Scout is a java thread which listens to the serial port and processes the received messages, the MoteDB updates the status of corresponding node when it receive a new message. The other two components (Panels and Logger) can access the MoteDB to learn the status of a particular node, Panels access the MoteDB for displaying the node status for the user and also when user sends configuration commands to the nodes on the network. Logger component access the MoteDB to log the collected data to a file for the later analysis.

Embedded Octopus application running on the nodes, allows the node to perform sampling of the sensor values and determine how often the data is sent to the gateway. Nodes on the network can send the collected data to the gateway in three ways: (i) A timer is used to determine how often nodes send data collected data to the gateway, it is also known as time-driven mode. (ii) In an event-driven mode, when timer fires nodes check whether any of the sensor reading values exceed the defined threshold before sending it to the gateway, this mode is used to generate alerts. (iii) In a query-driven mode nodes wait for the sampling request from the gateway to sample the sensor reading and send it to the gateway.

4.2. Octopus Message Format and Protocol Independence

Octopus supports two-way communication. Data collected by the nodes is sent to the GUI application (Upstream messages) for the visualization and the configuration commands are sent from the GUI application to the nodes on the on the network (Downstream messages). Currently two TinyOS default protocols; Collection Protocol Tree [12] and the dissemination protocol [13] are used for the upstream messages and downstream messages, respectively.

Two message types were defined to support the upstream messages and the downstream messages, these messages format are shown in the picture bellow:

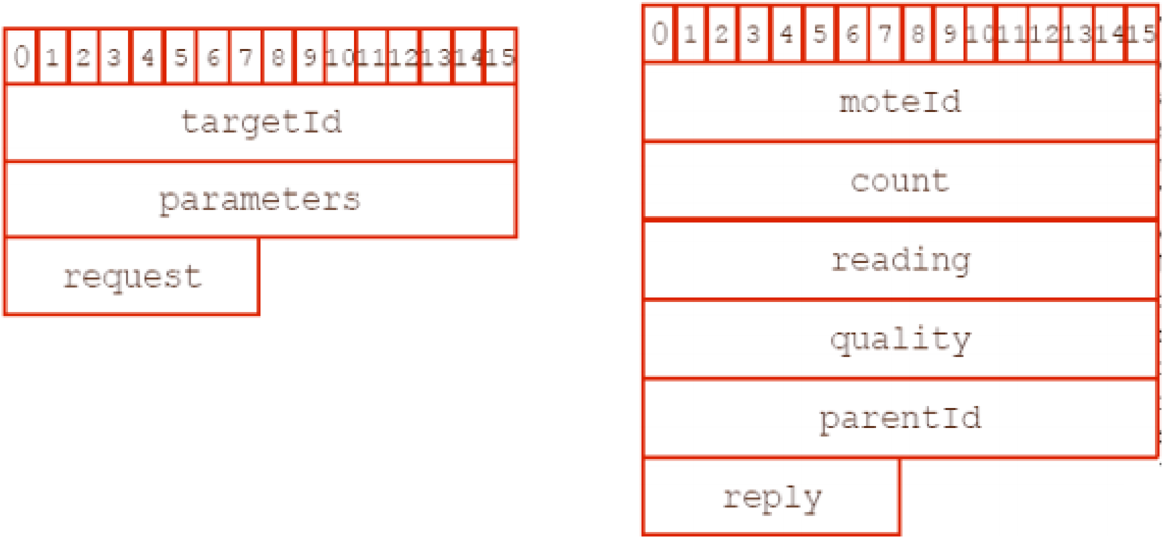

Figure 8 - Downstream message and upstream Message types [11]

Downstream message type contains the *targetId* field of (16 bits) to identify a node or a set of nodes as the message destination and the set of parameters in the *parameters* field (16 bits). The *request* field (8 bits) is used to identify the request types it also supports up to 256 requests.

The upstream message type contains: *moteId, count, reading quality parentId and replay.* These fields are used by the GUI application to display the network status such as the network topology and also log the collected data for the later analysis. To display the live network topology, GUI application aggregates the local topology information embedded in the upstream message (moteId and the parentId). MoteId refers to the source node of the message which is unchanged even when the message is forwarded through multiple hops. The *quality* field is used to determine the weight of the vector and the link quality. The following image show the vector format used to identify link quality and the aggregated vector used to display network topology in the GUI application.

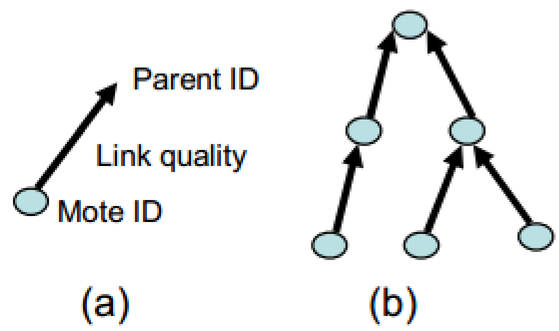

Figure 9 - (a) Octopus message vector format. (b) Aggregated vectors for forming topology [7]

When a message from a specific node arrives in the base station for the first time the *moteId* field is extracted by the java application form the message header to create a new node with this moteId, *parentId* is needed to determine the direction and the destination endpoint of the vector in the GUI. To visualize the network topology, an aggregation of the vector of all nodes in the network is carried out by the java application as shown in the figure (b) above.

The *reading* field contains the sensor reading values which can used by the java application for displaying live sensor reading, log them in a file or plot them in the network chart. The *reply* field indicates whether it's a reply to a user request or periodic message carrying the sensor reading.

All the necessary information for the visualization and the controlling the sensor nodes is included in the upstream message making the Octopus independent of the underlying routing protocol and allowing developers to experiment it with other routing protocol.

4.3. Octopus GUI Application

Octopus GUI application is developed in java that's run in computer with the sink node attached. Following are some of the main classes of Octopus GUI application:

- **Scout.java** is a thread which listens to the serial port for incoming messages from the base station to processes them, it also update the data base when receive a message from the existing node or creating new entry in the data base if for the new node.

- **Mote.java** is a class contains virtual mote with all the related characteristics to the real mote on the network and also the characteristics related to its parent.

- **MoteDataBase.java** is an implementation of linked-list to store instances of the Mote Objects it provides method for add, get and remove Mote.

- **Consolpanel.java** is use to generate text based display for events such as new mote was added to the database and when the message is received or sent.

- **ReadingrecordList.java and ChartPanel.java** these two classes are used for display the reading value of the motes.

- **RequestPanel.java and MsgSender.java** these two classes are used to create the request panel and the send the request from the user to the network.

- **LegendPanel.java** allows user to customize the displayed data such as; display nodeId, parentId, link quality, etc.

- **Logger.java** is used to log the data collected in a csv format for the later analysis.

- **Localization.java** and **Localise.java** are used to display network topology plus the floor or testbed layout.

Following is a screen shot of a latest version of Octopus User interface.

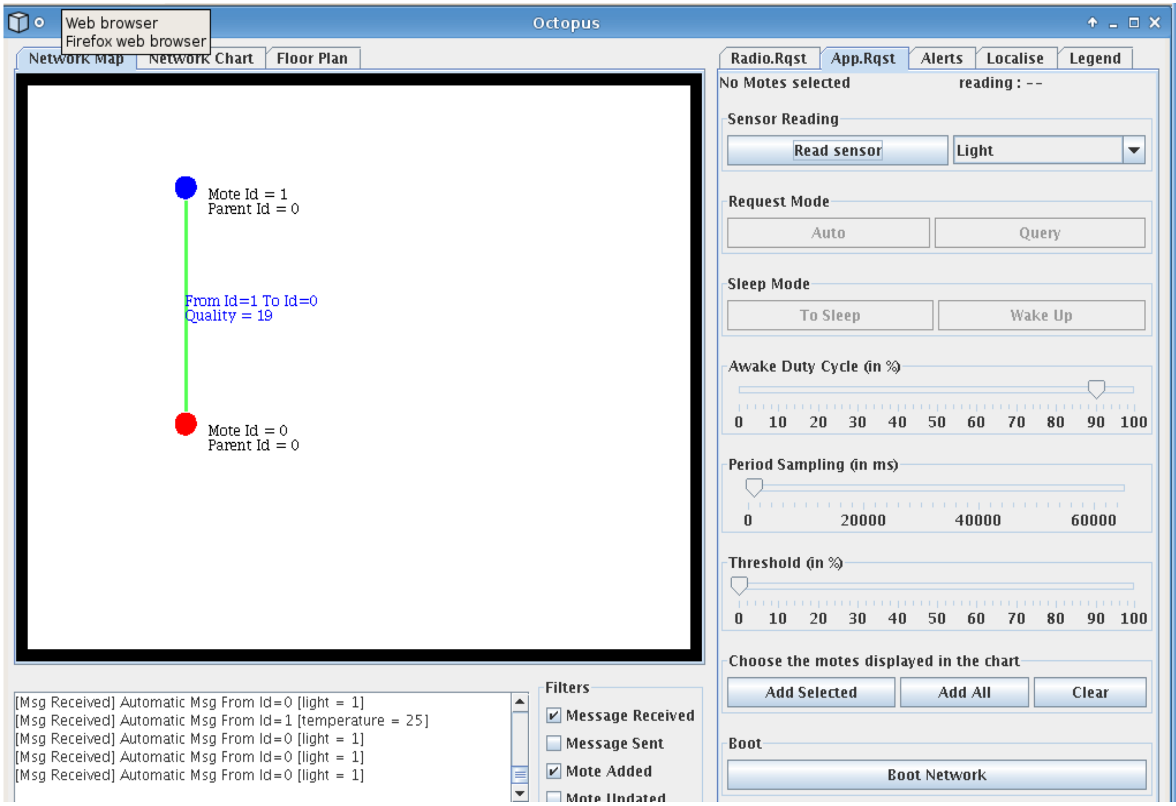

Figure 10 - Octopus GUI application

Octopus User interface can be divided in two main sections left and right table or panels. Panels on the left (*Network Map, Network Chart, Floor Plan, and Console on the bottom*) are used mainly for data visualization and the right Panels (*Radio.Rqst, App.rqst, Alerts and Legend*) are used to filter or control data displays on the left panel and also allow user to send control commands to the nodes on the network.

The console panel on the bottom is always visible to displays events i.e. according to option chose by the on the filters panel, whereas user can navigate to any panel on the left and the right panel. Above image currently chows *Network Map* on the left and the *App.Rqst* on the right. Network Map shows the logical topology of the network and some information about node and link status, also in this panel user can choose nodes to control by clicking on the nodes. After intended nodes are selected App.Rqst Panel can be used to send the control commands (choosing among three sensor reading values to be displayed, setting nodes duty cycle, setting sensor sampling period and add nodes to the chart)

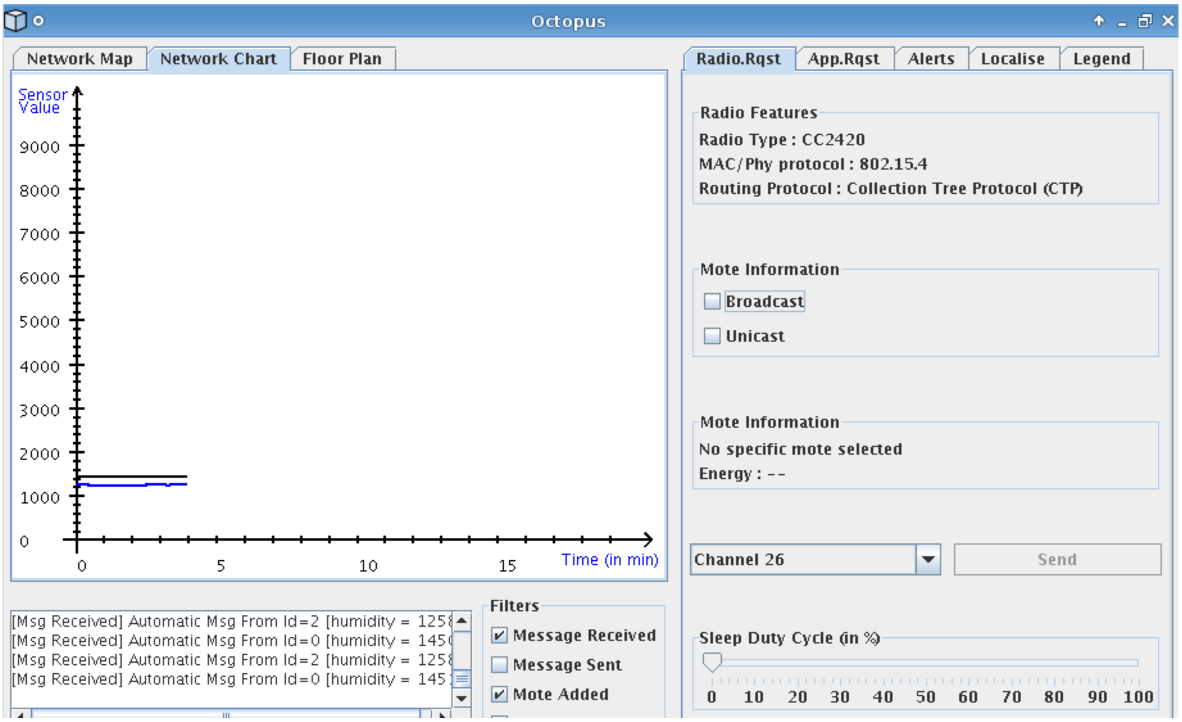

Figure 11 - Sensor reading chart

Above image shows charting feature of Octopus, after selected nodes are added to plot their sensor reading values, the chart can be visualize under *Network Chart*. The values are been plotted as they are received in a live fashion. The same values can also be displayed on the lower windows if user enables it under *Filters* panel.

Radio.Rqst Panel shows information related to Radio used for communication, allows user to choose target nodes (Broadcast and Unicast) information about selected, option to change Radio channel and configure the nodes duty cycle.

To use the *Floor Plan* panel the user must insert a floor plan layout in the *Localise* panel on the right. *Floor Plan* shows the logical network topology as shown on the *Network Map* panel plus a layout of the floor or testbed, this feature allows user to place the nodes according to their real locations.

The *Legend* Panel is used to control information to be displayed along with the nodes on the *Network Map* as shown in the Figure 10, this panel also provides the logging feature which allows user to choose fields of data to be logged.

Usage of these two features is depicted in the followings images:

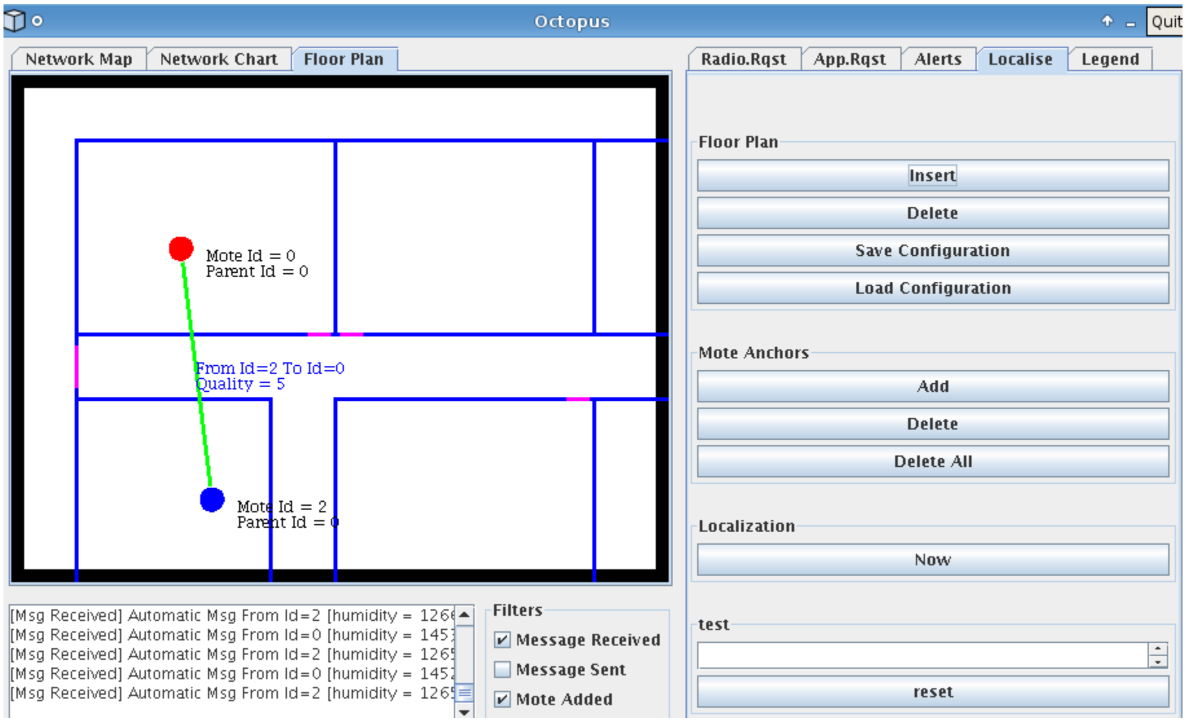

Figure 12 - Network topology plus floor layout

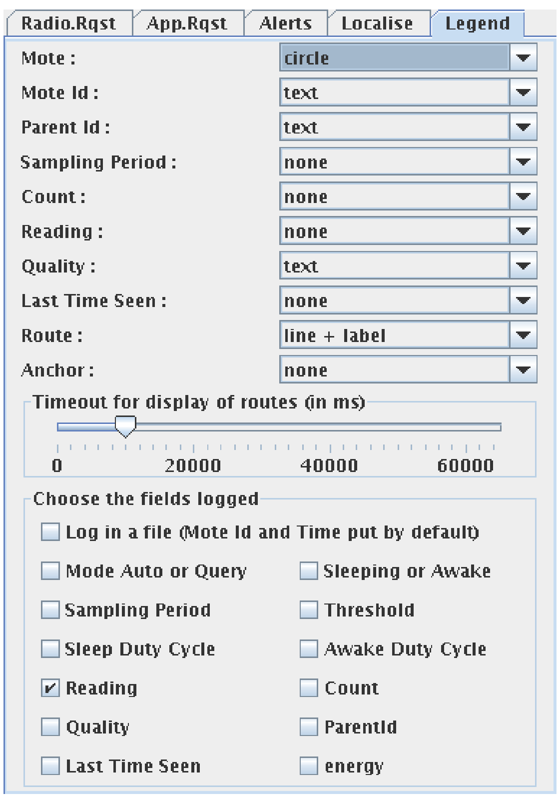

Figure 13 - Legend Panel

4.4. Experimenting Octopus and the difficulties

An experimental testbed for testing Octopus was carried out to further analyze the tool. The testbed consists of 2 nodes (1 sink node and 1 client) and the java application running on a PC for displaying the data or information gathered from the network. The sensors nodes aforementioned run an embedded version of Octopus that was implemented in TinyOS. The embedded application is responsible for sending the data gathered by the various sensors attached to the nodes to the java application running on the PC via sink node.

The purpose of this experimentation was to explore the Octopus features in action, the message format use for communication between the Octopus embedded application running on the motes and the PC application and the possibility of altering the message format used to include more fields on the packets send to the GUI application.

The changes were made only for the Octopus embedded application (The TinyOS code) to integrate a mechanism to monitor the status of the USER Button found on the nodes. To add this feature, the initial attempt was to add a new field in the massage format that contains value regarding the button status (value 1 if button is pressed and value 0 if it is not pressed). Although this is the correct and the preferred way of testing the ability of adding the new fields to the message format it would require to alter all the GUI application (the java codes) in order display this newly added field.

To avoid altering the GUI application which is not mandatory for testing the possibility of adding new fields to the message format, since message format configuration is implemented on embedded version of Octopus. The new field for the button status was added to the embedded version while on the GUI application for visualization purposes this field's values will be received and displayed in place of the light sensor reading values as shown in the following image.

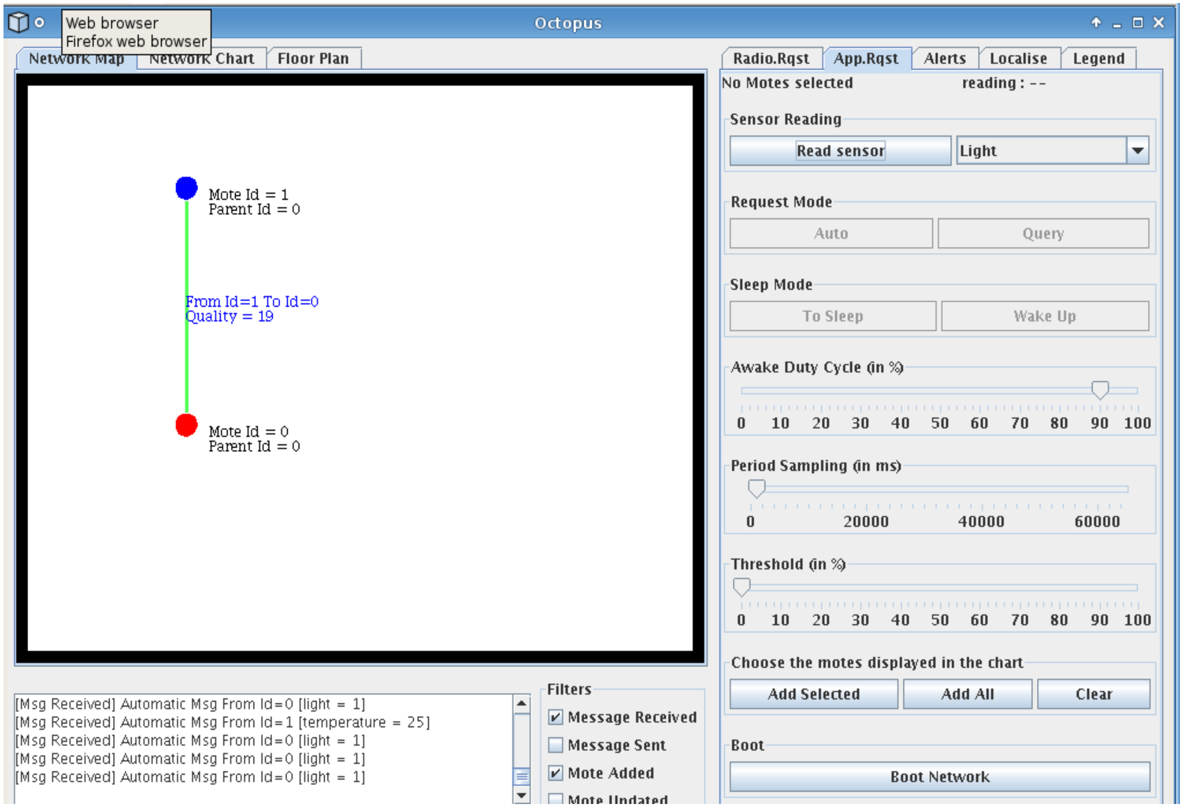

Figure 14 - Button status values

In the above image, console panel on the bottom shows the 3 sensor reading values (Temperature, Light and Humidity) in this case the Light sensor values have been replaced by the values of the button status which is either 1 or 0 depending on the button status.

The field was successfully added to the message format and also can be visualized in the place of the light sensor reading value, however, regardless of the button states the values appears on the GUI does not change (always display 1 even when the button is not pressed).

4.5. Octopus limitations

As already discussed in the state of art chapter Octopus provide data visualization (sensor reading values), nodes battery level, link quality between child and parent node these data can be visualized directly in the GUI application or logged into file for later analysis. It also provides a mechanism to perform basic control to the nodes on the network such altering nodes sampling period, query a specific data on the nodes, setting nods duty cycle and sensing thresholds.

The tools also suffer the same limitation as the other tools none of the events or packet received by the GUI application has time-stamp associated, which make it hard to determine when the last packet was received or to check the node's active time. It has no dedicated database to store all the events or packets which allows the user to replay or visualize events or packets received in a certain span of time. Octopus, as it is now visualizes the data in a live fashion in the GUI application thus by default it doesn't maintain any record.

Octopus provides possibility to log the data received to a file for later analysis, however this file only contain the information specified in the message format and the data in this file is mainly used for graphing purpose. The limitation of this approach is that the data in the file will be processed manually by the user, i.e. the data is import to a Microsoft excel program to generate graphs.

The other limitation of the saving the data to a flat file is hard to visualize event or packet received in a particular span of time, because it has to be done manually especially when the file contain hundreds or thousands. The database is necessary to store all the record and make the data processing easy by issuing queries against the database

5. New Octopus monitoring tool

After conducting the experimental setup of Octopus and analyzing its limitations in the previous chapter, this chapter focuses on the alterations necessary for Octopus to measure performance related metric of WSNs.

As discussed in the previous chapter, Octopus is using flat files to store the data received by the application. While this is a useful feature to have, it is not convenient because it is a tedious work to find specific events of interest on the file, especially when the file contains lots of records, and ultimately makes it harder for other program to analyze and used data stored on the file.

To overcome the storage problem, MySQL database is used to store all the data received by the application, this makes it easier for other program to use the collected data and retrieve interest events and data occurred in the past.

Time synchronization on WSNs is essential to keep track and time-stamping events as argued by the authors in [14] and considered paramount to perform measurement of some of the performance related parameter such as to calculate packet delay, which requires nodes involved in the communication agree upon one global time. The current version of Octopus doesn't provide any mechanism to synchronize nodes clock. For clock synchronization purposes it was necessary to integrate Flooding Time Synchronization Protocol (FTSP) [15] . FTSP will be discussed in detail in a following topic.

Packet delay information is available in a real-time fashion, where packets delay is visualized in form of a chart for each node. The delay of the last packet received from a specified node and also packets delay for a specified time range in the past, is also shown.

Each node maintains a counter for the packets it sends. This counter is included in every packet (in the counter field in message format as shown in Figure 17) and it increments by one for every new packet. Packet loss information for each node is by default shown for the last thirty minutes. Also, start time and end time can be specified to visualize packet loss information for the desired time range.

This chapter discusses in detail the development of new octopus monitoring tool and the alterations made to the previous version of Octopus and some tests for different hops.

5.1. Global architecture of new Octopus

Following image depicts the global structure of the new Octopus.

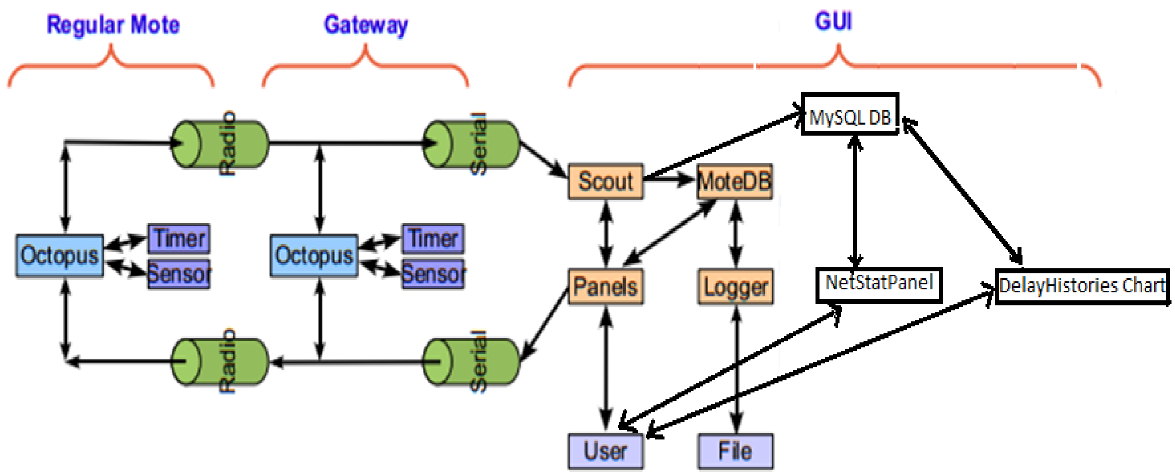

Figure 15 - Global Architecture of new Octopus

As depicted in the above new three blocks were added to the original architecture of Octopus (*MySQL DB, Net.StatPanel and DelayHistories Chart*). However it doesn't change the data flow and communication among the blocks of the previous Octopus, just added a few more interactions among "Scout" Block and some new blocks.

On the previous version of Octopus when the packets are received by the "Scout" block from the Gateway or sink node it then sends the received data for every node on the network to be stored on in the MoteDB block. Panels and Logger then can use the data on the MoteDB for Visualization or for logging purposes as shown by the image above.

The new version of Octopus follows the same procedure as described in the previous paragraph, and also send the data to be stored in MySQL database. This is necessary because the data stored in the MoteDB block is only available when the Octopus is running, when the application is closed or restarted the data is lost. By Storing the receive data also in the MySQL database make it persistent and can be accessed later in time when needed.

Another reason not to completely replace the MySQL database with the MoteDB is because some real-time data visualization such as displaying sensor reading value for each node or graphing these values as they are received is much faster using the data store in the MoteDB rather than Querying the MySQL database. For this reason all the data for visualization in a real-time fashion are obtained from the MoteDB.

MySQL database is mainly use for retrieving previous data stored in there: i.e. retrieving packet delay and packet lost histories for each node for a specified time in the past and also some statistical calculation is based on the data store in the database.

5.2. Octopus database design

MySQL database was chosen to store data collect from the network. For portability and ease of use purposes rather than installing the MySQL from the source for Linux distribution and manage it via command line, a portable version of MySQL and the GUI program for managing it (PHP MyAdmin) is made available by *Apache friends* [16] under name of XAMPP which is available both for Windows and Linux platforms.

Octopus database consists of two tables (Motes table and Packets table) "Motes table" stores information related to node status and the "Packets Table" information related to the packets received from the WSNs.

The columns in these tables are the same fields defined in the octopus message format as shown in Figure 17 on the right with new fields added to measure performance related metrics.

The columns on the both tables can be observed in the following image:

localhost » octopus_db » Motes

Browse | Structure | SQL

#	Name	Type	Collation
1	mote_id	int(11)	
2	reading	int(11)	
3	reading_long	int(11)	
4	scale	int(11)	
5	energy	int(11)	
6	quality	int(11)	
7	parent_id	int(11)	
8	is_sync	int(11)	
9	reply	int(11)	

localhost » octopus_db » Packets

Browse | Structure | SQL

#	Name	Type	Collation
1	counter	bigint(15)	
2	time	timestamp	
3	packet_id	int(11)	
4	source_mote	int(11)	
5	send_time	bigint(15)	
6	receive_time	bigint(15)	
7	delay	bigint(15)	
8	lastTime_seen	bigint(15)	

Figure 16 - Database tables (Motes and Packets) Layout

Both tables are linked by the *mote_id in* the Motes Table and the *source_mote* in the Packets table. Some of the columns in the both table are solely use as a filter or used to perform calculations. (i.e "reading_long", reply and scale in the Motes table and "counter" in the Packets table).

5.3. New Octopus message format

As discussed in the section 4.2 Octopus uses two different message formats for upstream message (from WSNs toward monitoring tool) and downstream packets (commands send from GUI to sensor node on the WSNs). The downstream message format was not changed whereas new fields were added to the upstream message format to include more information about the packets as can be observed in the following image.

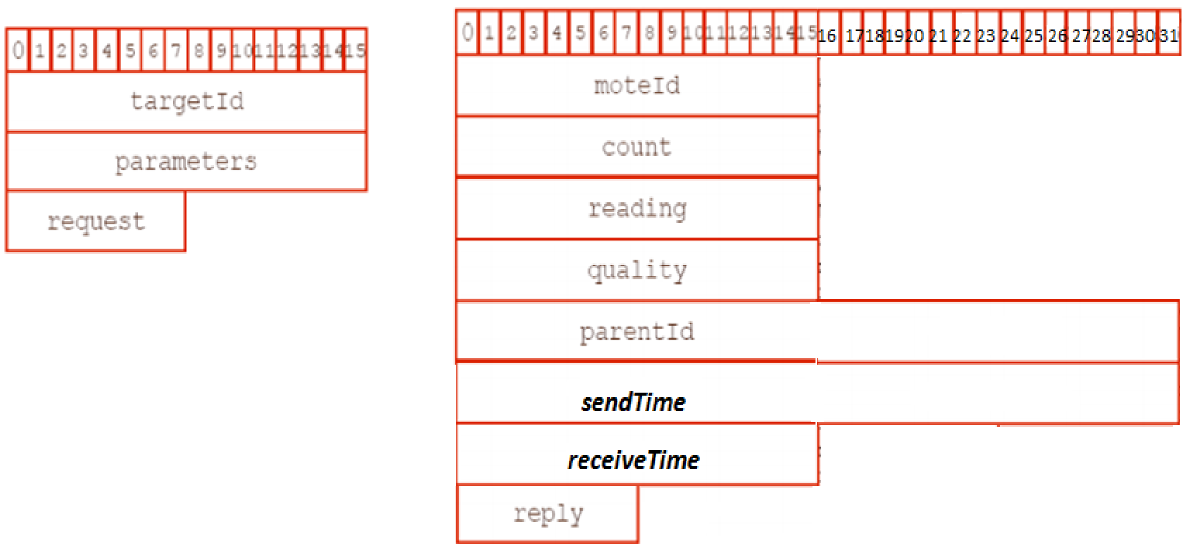

Figure 17 - Message format of the new Octopus

On the right is the up-stream message format two new fields *sendTime and receiveTime* were added to keep track the packet sending and receiving time respectively.

sendTime of a packet is obtain from the packet's source node and the *receiveTime* is the time of packet reception by the sink node, independently of the number of hops the packet travels, both values are in millisecond precision. These changes were made on the Octopus embedded application running on the motes (*Octopus.h* file)

The size of these two fields is of 32 bit long compare to others fields because the time return by the time synchronization protocol is of 32 bit long as shown in code snippet in section 5.5.

5.4. Nodes clock synchronization

The need for network level time synchronization is crucial for the monitoring purposes for time-stamping events and essential for calculating packet delay. There are several time synchronization protocols for WSNs. This section discusses briefly some of the existing time synchronization protocol for WSNs:

- Reference Broadcast Synchronizations (RBS) [17], in RBS nodes periodically broadcast reference beacons to its neighbors, this reference doesn't contain explicit timestamp instead receivers use its reception time and record their local time to exchange with their neighbors. According to [15] the protocol has not been extended for large multi-hop networks.
- Timing-sync Protocol for Sensor Networks (TPSN) [18] achieves network level time synchronization first by creating a structural topology in the network and guaranties that at least a node in a level can communicate with the upper or lower level in the topology structure, this phase is known as "discovery phase". The next phase is "synchronization phase" a node belongs to higher level synchronize to a node of level immediately below it, this process is repeated until synchronization reaches the level zero of the topology structure where the root node resides. As described in [15] by using MAC layer level radio time-stamping this protocol achieves two times better performance than RBS protocol. One of its downside is that it does not estimate the clock drift of the nodes.
- Flooding Time Synchronization Protocol (FTSP) [15] this protocol uses the radio broadcast to synchronize possibly multiple receivers to the time provided in the radio message. Receivers then record in their local time of the message reception time, this result in a global-local time pair (Time included in the received message and the time stamp taken when the message was received). This time pair is used to estimate the clock offset of the receivers.

  For managing clocks drift between a sender and receivers Linear regression is used to determine the error for different broadcast beacons, apart from determining the error liners regression is also used to determine the periodic beacons interval. In a multi-hop environment the network wide time synchronization takes a proportional time to the number of hops. Root node is determined by the lower "nodeId" and root node re-election takes place when the previous root a no longer available or disconnected from the network.

FTSP is the time synchronization protocol used with the new Octopus monitoring tool for clock synchronization purposes, because this protocol is the latest one up to date and the results obtained by this protocol is better than the two previous time synchronization protocols according to the comparisons made in [15] and among this three protocols FTSP is included as a library in TinyOS by its developer.

5.5. Alteration of Octopus embedded application

Octopus embedded application runs on the sensor nodes consists of files written in nesC language [19] and some C language header files. Beside the changes discussed in 5.3 to incorporate time synchronization protocol to the Octopus embedded application the file "OctopusAppC.nc" was added the *TimeSyncC* component which provides the clock synchronization and *LocalTimeMilliC* component to obtain node's local time and converts it to corresponding global time. The components and the wiring necessaries for the time synchronization are shown code snippet below.

```
// Components and wirings for Clock synchronization
components TimeSyncC, LocalTimeMilliC;
MainC.SoftwareInit -> TimeSyncC;
TimeSyncC.Boot -> MainC;
OctopusC.GlobalTime -> TimeSyncC;
OctopusC.PacketTimeStamp -> ActiveMessageC;
OctopusC.LocalTime ->LocalTimeMilliC;
OctopusC.TimeSyncInfo -> TimeSyncC;
```

The components used are listed in the first line after the keyword *components* and the rest of the lines are wirings which connect the user and provider of a service, the arrows indicates the user and the provider. The second and the third lines initiate the FTSP protocol when the sensor node boot, lines start with "*OctopusC.*" indicates the services use by Octopus and their providers.

The file "OctopusC.nc" was also altered to include the packet sending time from the source node and the packet receiving time in the sink node. These changes were done in their appropriate sending and receiving event.

```
//Adding the packet sending time for the regular nodes
task void collectSendTask() {
        uint32_t tmp;
        tmp = call LocalTime.get();
        localCollectedMsg.isSync = call GlobalTime.local2Global(&tmp);
        localCollectedMsg.sendTime = tmp;
        localCollectedMsg.rootAddress = call TimeSyncInfo.getRootID();

}
```

The lines above were added to the *task* that responsible for sending packets, the first line obtains the node's local time using the "LocalTime" interface then the second line converts it to the global time and finally the value is included to the appropriate field of the packet. "localCollectedMsg.isSync" and "localCollectedMsg.rootAddress" both fields are related to node synchronization information, first is used to determine whether node is synchronized and the second is to obtain the address (moteId) of which the node synchronize its clock to.

Sink node set the packet reception time, the process is similar to the one described above the only difference it's done inside the reception event. Following is the snippet code for determining packet reception time.

```
event message_t *CollectReceive.receive(message_t* msg, void* payload, uint8_t len) {
        octopus_collected_msg_t *collectedMsg = (octopus_collected_msg_t *) payload;
        //Adding packet receiving time
        Uint32_t tmp, receiveTime;
         tmp = call LocalTime.get();
        receiveTime = call GlobalTime.local2Global(&tmp);
        collectedMsg->receiveTime = tmp;
}
```

5.6. New Octopus user interface

New Octopus user interface added three new panels for calculating performance related parameters (Net.Stats), plotting packets delay in real-time fashion (Live Delay Chart) and a panel to plot packet delay for specified period of time (DelayHistory Chart).

Data uses by these new panels are obtained from the MySQL database, except data for real-time packet delay chart which plots the delay values as they are received. Real time packet information values are also sent to the database for later visualizations. New Octopus user interface is shown in the following image.

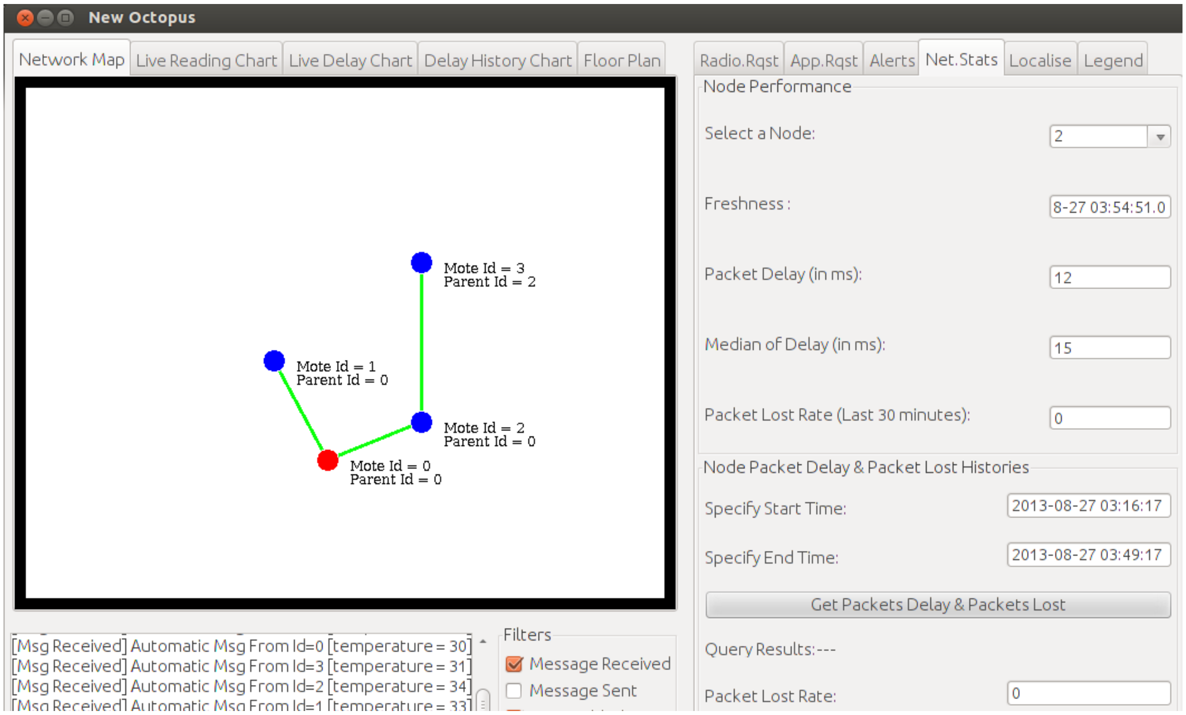

Figure 18 - New Octopus user interface

The new panels added will be discussed in details in the following subsections.

5.6.1. Net.Stat panel

Net.Stat panel shown in Figure 18 contains performance related information organized based on the selected node, and also provides access to the delay history of the node. After a node is selected from the list the subsequent fields will be populated: *Freshness, Packet delay, Median of delay and Packet Loss Rate.* The list shows all the nodes found in the database regardless of their current status (Active or Inactive), allowing users to obtain information of the desired node even if the node is currently inactive. Each of these fields will be discussed in details:

- Node Selection – This list contains the nodes stored in the database. Other fields will be updated accordingly when a node is selected. Following is the SQL query use to obtain the list of nodes "**SELECT DISTICNT source_mote FROM Packets**"

- *Node Freshness* – This field contains information of the last packet received from the node. The time is precise to seconds. This is useful in the WSNs communication because the radio links are unreliable, with nodes disconnecting and connecting again to the network at any time. "**SELECT time FROM Packets WHERE source_mote = ? ORDER BY time DESC LIMIT 1**"

- *Packet Delay* – It is a time in milliseconds denoting the delay of the last packet received from the specified node. The respective SQL query is "**SELECT delay FROM Packets WHERE source_mote =? WHERE source_mote = ? ORDER BY time DESC LIMIT 1**"

- *Median of Delay* – Median of the delay is a preferable way of measuring overall node performance because individual packet delay is not stable due to the unreliable link quality and the radio interference can cause individual packet higher or lower from the expected range. The SQL Command used was adopted for this field was adopted from [20] "**SELECT x.delay, AS median FROM Packets x, Packets y WHERE x.source_mote = ? AND y.source_mote =? GROUP BY x.delay**
  HAVING SUM (CASE WHEN y.delay <= x.delay THEN 1 ELSE 0 END) >= (COUNT (*+1)/2) AND
  SUM (CASE WHEN y.delay >= x.delay THEN 1 ELSE 0 END) >= (COUNT (*)/2) +1"

- *Packet Loss rate* – This field contains information about number of packet lost experienced by the node in the last 30 minutes. The respective SQL command "**SELECT SUM(stop – start +1) AS lost_count FROM(SELECT l.packet_id + 1 as start, MIN(fr.packet_id) -1 AS stop FROM Packets AS l**
  LEFT OUTER JOIN Packets AS r on l.packet_id = r.packet_id -1
  LEFT OUTER JOIN Packets AS fr on l.packet_id < fr.packet_id WHERE (r.packet_id IS NULL and fr.packet_id IS NOT NULL*) AND l.source_mote =? AND l.time >= DATE_ADD (*NOW(), INTERVAL -30 MINUTE*) GROUP* BY l.packet_id, r.packet_id) AS t"

- *Node Packet delay & Packet Lost history* – this feature allows user to retrieve selected node packet delay and the packet lost for a specified time range, both the start and the end time must be provided in their respective field.

- *Packet Loss Rate* – After specifying the start and the end time, clicking the "Get Packet Delay & Packet Lost button" will return the results of the search or query, the number of Packet lost for the specified time range will be shown in the "Packet Loss Rate" SQL command used is
  "SELECT SUM(stop – start +1) AS lost_count FROM(SELECT l.packet_id + 1 as start, MIN(fr.packet_id) -1 AS stop FROM Packets AS l
  LEFT OUTER JOIN Packets AS r on l.packet_id = r.packet_id -1

***LEFT OUTER JOIN Packets AS fr on l.packet_id < fr.packet_id WHERE (r.packet_id IS NULL and fr.packet_id IS NOT NULL) AND l.source_mote =? AND l.time BETWEEN ? AND ? GROUP BY l.packet_id, r.packet_id*) AS t**"

- Selected node's packets delay can be visualized in form of a chart, this feature allows user to observe the node's packets delay evolution for the specified time period. User needs to enter the start time and end time. Label on the bottom of the panel will inform whether records are found for the specified period, if no records found for the specified period user will be informed to refined the search otherwise user will be told to visualize the chart in DelayHostries Panel. Used SQL command "**SELECT delay, time FROM Packets WHERE source_mote =? AND time BETWEEN? AND?" "SELECT "**

The Question mark inside the SQL commands denotes values that are specified by the user and passed as a method parameter when the query executes.

5.6.2. Live Delay Chart panel

Delay chart panel allows users to observe the delay evolution in a real-time fashion. Packets delay is plotted as is received by the application. The data used by this panel is obtained from the MoteDB block because storing and retrieving data to MySQL database can be a slow process when database contains thousands or millions of records. The live delay chart feature is shown in the following image.

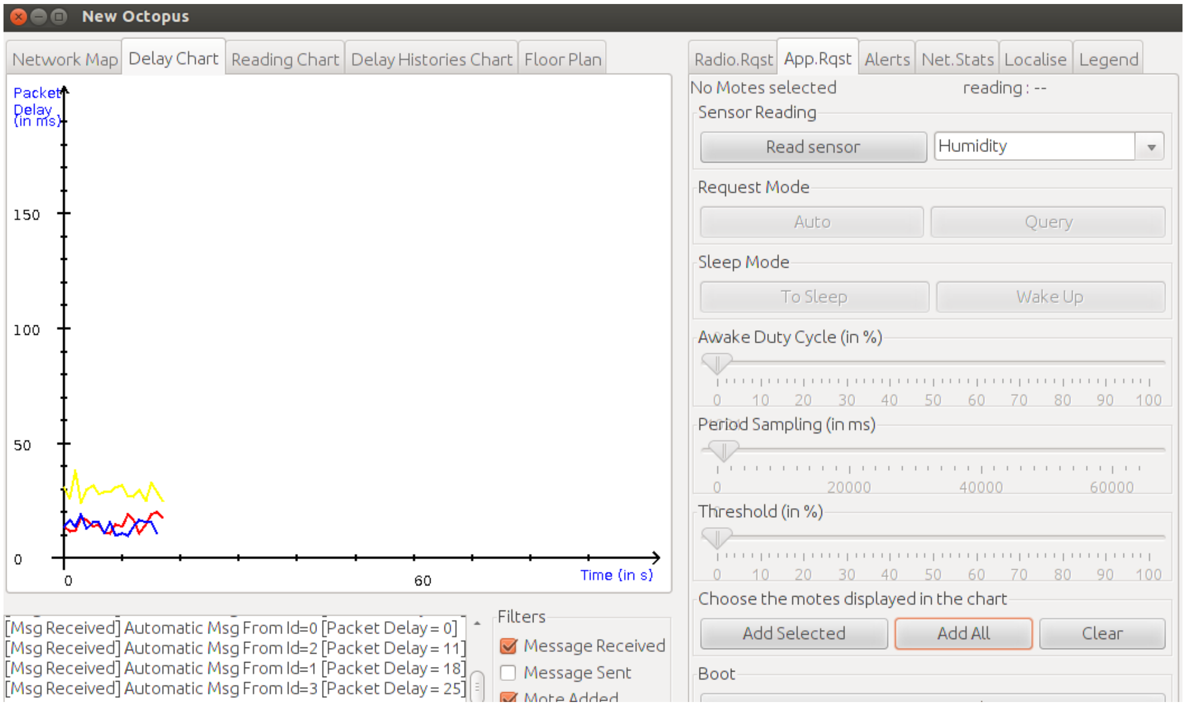

Figure 19 - live delay chart

Nodes can be added to the chart at any time, to visualize their packets delay evolution or to add specific nodes. On the *Network Map* Panel, as shown in Figure 18, an individual node can be selected by clicking on the desired nodes. Afterwards, on the *App.Rqst* choose the “Add Selected” under the “Choose the motes displayed in the chart”. If all the nodes will be included on the chart, “Add All” button can be used to avoid selecting the node one by one. The delay values are plotted on the Y-Axis. X-Axis is used to compress the chart if needed - the time will be shown in seconds for the first 100 seconds and then will change to minute. The live chart can be used to visualize the delay evolution for two hours. The “Clear” button can be used to clear the panel and plot a new chart. The corresponding delay values and nodes are shown on the panel in the bottom.

Above image shows a live delay chart of three nodes. Nodes with id 1 and 2 are connected directly to the sink node whereas node with id 3 is two hops away from the sink node which has a packet delay average twice higher than those connected directly to the sink node, as shown in Figure 18 - New Octopus user interfaceFigure 18.

5.7. Delay History Chart Panel

As described in section 5.6 after specified the time range and execute the query if results exist for the period, then, the corresponding chart can be visualized by opening the Delay History Chart, as shown in the following image.

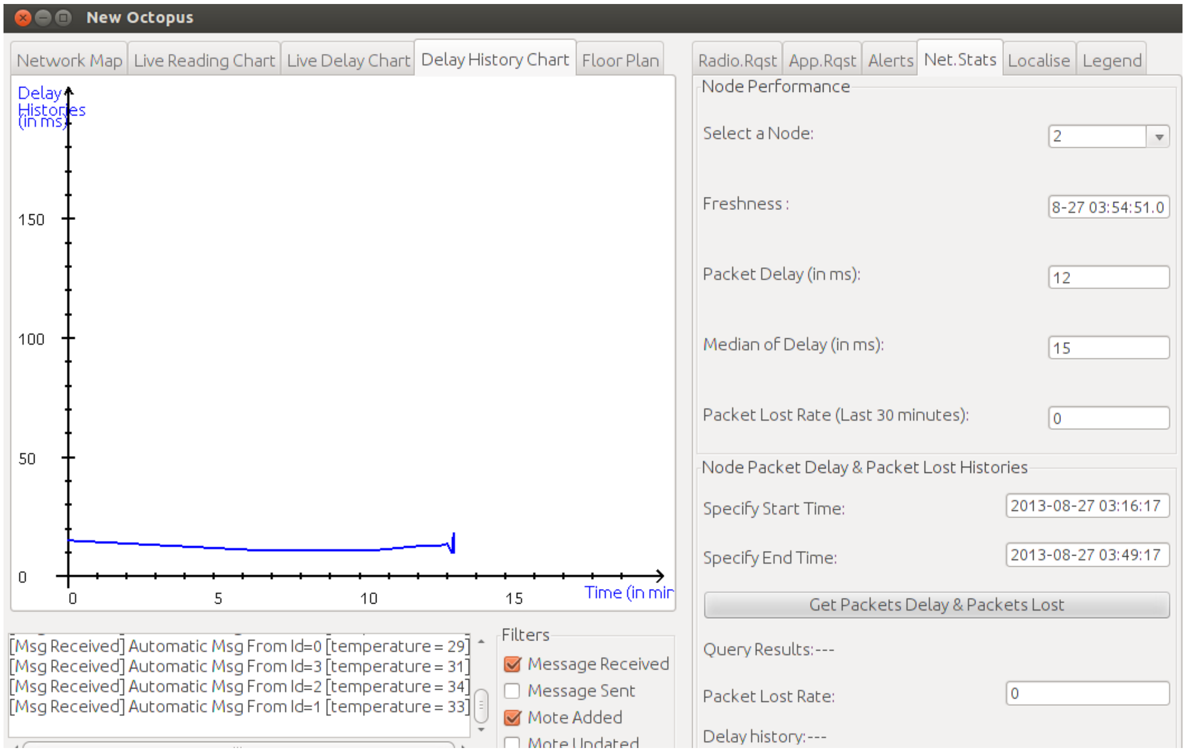

Figure 20 - Packets delay history chart

If no result found for the specified period, the “Delay history:---” label will inform that no results found for the specified period and no chart will be provided.

5.8. Evaluating delay and Packet lost in multi-hop WSNs scenarios

Octopus as described in the previous chapter uses Collection Tree Protocol (CTP) routing protocol to route packets from regular nodes to one or more specified root nodes. CTP form a tree structure where data from the leaf nodes are sent to the one or more root nodes, as described in its implementation for TinyOS TEP [21] , it is addressed-free which means nodes do not send packets to specific nodes, rather they implicitly choose next hop as root.

Expected Transmissions (ETX) is used to determine the next hop to send the packet to. Root node has ETX of 0 while regular nodes ETX are determined by the ETX of its parent plus the ETX of the link to its parent. The next hops for a node is not always the same every time the application start sand in case of link failure the protocol is also agile in finding a new next hop.

The purpose of the evaluation of packets delay and packets lost in a different number of hop of communication is to find out how the number of hops involved in a communication affects the packets delay and packets lost, individual packet delay as already discussed in section 5.6.1 is not stable that is why the media of the delay will be used. In case of packets delay the median delay value for each hop must be proportional for the number of hops that the node has or in.

In a topology as shown in Figure 18 which comprises of four nodes, node with ID 1 and 2 communicate directly with the sink node while, node with ID 3 is two hops away from the sink node in this case node 2 relays the packet from node 3 to sink node. Also in the same figure shows that the median packet delay for node 2 is 15 milliseconds (for uni-hop communication) even in a median representation the median of packets delay can vary from 8 to 17 milliseconds for different environment, packet delay when the test is done in an environment where other wireless technology also use the ISM frequency as described in [22].

The packets delay for node in a multi-hop scenario is dependent proportionally to the packets delay of its parent node. Following image show the difference packets delay carried out in a different environment Figure 18 is a screen shot of test was in an environment which WSNs coexist with other wireless technology such WLAN and Bluetooth. The same test was done in an environment where there is no other wireless technology the results can be observed in the following image.

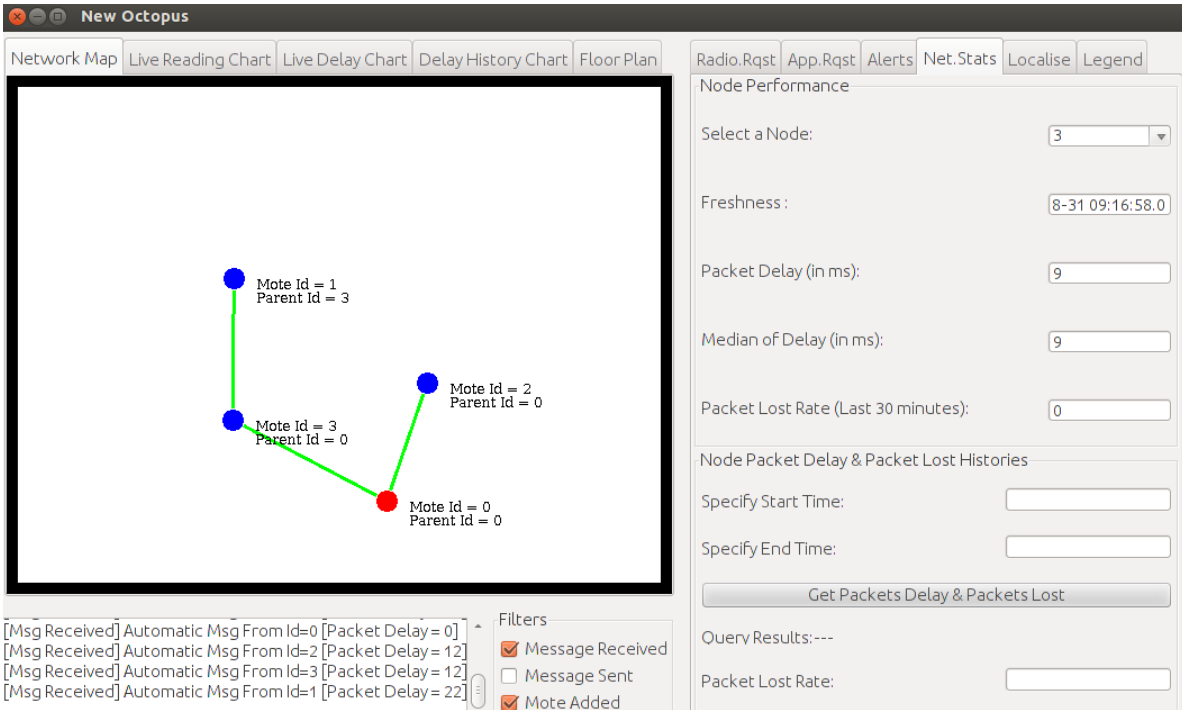

Figure 21 - Packets delay in a testbed with no interference

This is the same topology structure with less or no interference from the environment. The median delay in this case for node with ID 3 is 9 milliseconds and the median packet delay for the node 1 is 22 milliseconds, which is a little bit over twice as the median delay for the parent node (node 3). The fact that node 3 relays messages from node 1 do not affect (increase) its packet delay

On the other hand the packet loss rate is loosely related to the number of the node from these two tests. There is almost no difference in the evaluation of the packet loss rate for each node in different hops for the last 30 minutes.

5.9. Overhead incurred by the adding new features

The packets structure also known as *message_t* in TinyOS consists of header, data and footer as described in [23], size of each section is platform specific (depends on the radio chips used). For telosB platform which used during this dissertation uses CC2420 radio chip which has the following configuration: packet header size of 11 bytes, data size of 28 bytes (This is reconfigurable depending on the amount of data to be included in the packet) and the packet metadata of 7 bytes.

By adding new fields to Octopus message format as described in section 5.3 requires to extend the data section of the packet which by default is 28 bytes. It was necessary to alter this section to 48 bytes, this was done using the ncc (nesC compiler) option by setting "CFLAGS += **-DTOSH_DATA_LENGTH=48"**, this change was made in the Makefile. Previous version of Octopus used the default size.

Following is the content of the Makefile, the line referred above is found in the second line.

```
#Content of Makefile
COMPONENT=OctopusAppC

CFLAGS += -DTOSH_DATA_LENGTH=48

CFLAGS += -I$(TOSDIR)/lib/net/ -I$(TOSDIR)/lib/net/ctp -I$(TOSDIR)/lib/net/le -
I$(TOSDIR)/lib/net/drip/ -I$(TOSDIR)/lib/ftsp -I.
BUILD_EXTRA_DEPS = OctopusSentMsg.class OctopusCollectedMsg.class
MIGFILES = OctopusCollectedMsg.java OctopusCollectedMsg.class OctopusSentMsg.java
OctopusSentMsg.class

PFLAGS += -DENERGY_MONITORING=0
OctopusCollectedMsg.class: OctopusCollectedMsg.java
	javac OctopusCollectedMsg.java
OctopusCollectedMsg.java: Octopus.h
	mig java -target=$(PLATFORM) $(CFLAGS) -java-classname=OctopusCollectedMsg
Octopus.h octopus_collected_msg -o $@

OctopusSentMsg.class: OctopusSentMsg.java
	javac OctopusSentMsg.java
OctopusSentMsg.java: Octopus.h
	mig java -target=$(PLATFORM) $(CFLAGS) -java-classname=OctopusSentMsg
Octopus.h octopus_sent_msg -o $@

include $(MAKERULES)
migclean:
	rm -rf $(MIGFILES)
```

5.10. Possible improvements and future work

This section discusses the possible improvements, future work and some remaining problems regarding the implementation of the new Octopus. Specifically, this section focuses on time required by the FTSP to synchronize all the nodes on the network and other features that can be added to the new octopus as a feature work.

The current implementation of FTSP uses a periodic broadcast beacons to time synchronized nodes on the network, however, the time requires to synchronize all nodes is non-deterministic, especially when the networks consist of multiple hops. While the clock synchronization in process packet sending time and the packet reception time can be in a different value range, which can affect the packets delay calculation. This happens when the nodes boot for the first time and also when there is a sudden topology change in the networks, when this happens the packet delay can be a huge positive number or even a negative number. This phenomenon sometimes can be observed on the live packets delay chart when it happens.

Another problem with the FTSP is that happens sometimes even when all nodes synchronized, especially for packets delay, the value spike to a huge positive or negative value. Individual packet delay is unstable as already discussed in section 5.6.1 thus, median of the packets delay is used instead. In order to get a correct median packets delay for each node, the negative value of packet delay will be set to zero before the median packets delay calculation is perform.

The node duty cycle setting is not included in the new Octopus because it can provoke an increase of non-deterministic time of FTSP to synchronize nodes in the network and because the FTSP requires the radio to be on every time to keep nodes synchronized. The current FTSP broadcast beacons is every 5 millisecond, even though it still struggle to synchronize multiple hops with this time interval. More work for FTSP is required to overcome above mentioned problems.

Regarding features improvements of new Octopus, one would be adding more sophisticated statistic data presentation i.e. *moving average* of packets delay to smooth out the packets delay fluctuations base on time series.

6. Work plan

This chapter discusses the works done in the first semester and the changes made to the initial work plan for the second semester.

6.1. Work plan for the first semester

Beside of Octopus experimentation in TinyOS, a number of studies about the WSNs monitoring tools such as: study of literature and the requirements analyses of WSNs monitoring tool for measuring QoS related parameters in WSNs with controlled performance

ID	Task Name	Start	Finish	Duration
1	Reading literature of WSNs monitoring tools	17/09/2012	28-09-2012	10d
2	Study existing WSNs monitoring tools	01/10/2012	31-10-2012	23d
3	Experimental setup of Octopus	01/11/2012	03-12-2012	23d
4	Requirements analysis of the monitoring tool	05/12/2012	08-01-2013	25d
5	Elaborating intermediate report	12/12/2012	25-01-2013	33d

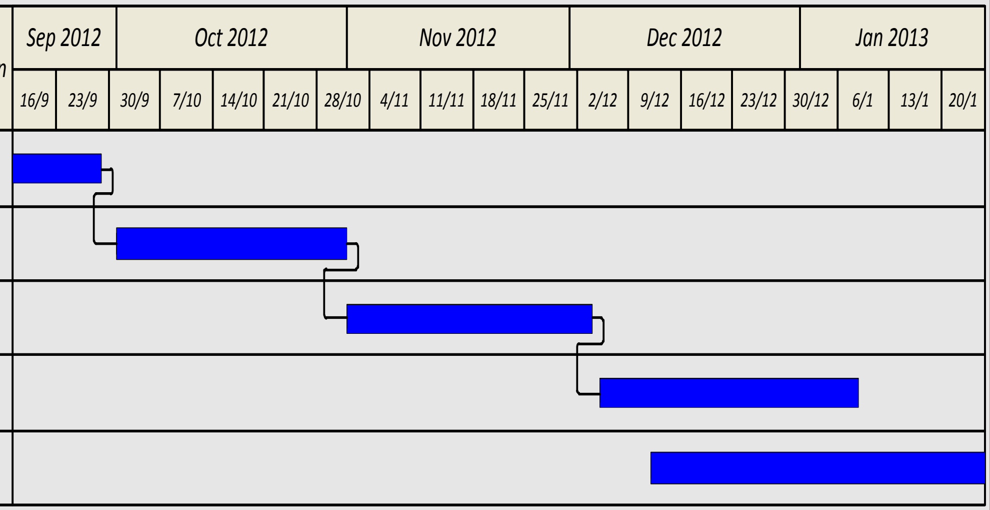

Figure 22 - Works done in the first Semester with their respective durations

Main tasks of the first semester are the following:

6.2. Work plan for the second semester

An initial work plan for the second semester consists of five main tasks; (i) Adapting Octopus monitoring tool, (ii) Implementing Database and measuring performance related metrics, (iii) Developing the GUI and the node's application , (iv) Testing the new monitoring tool, (v) Elaborating final report.

A new task was added to the prior proposed tasks. This task is essential to perform the Packet delay calculation and also time-stamping events, this task was inserted as the second task prior to database implementation and measuring some QoS (mainly Packets delay) related parameters.

Following is the description of the tasks and their durations are available in a Gantt chart.

1. **Adapting Octopus monitoring tool.**
   The monitoring tool will be based on Octopus monitoring tool, after an experimental setup made in the first semester. Octopus was chosen because of interesting Octopus features will be retained on the new monitoring tool, such as the ability to send control messages to nodes on the network. Also in this activity octopus GUI components and classes are studied in-depth.

2. **Studying and incorporating FTSP with Octopus.**
   Three time synchronization protocol for WSNs discussed in 5.4 (RBS, TPSN and FTSP) were revised to acquire knowledge of time synchronization protocol for WSNs and modify Octopus embedded application to incorporate FTSP to perform network wide time synchronization.

3. **Implementing Database and measuring performance related metrics.**
   Database is necessary to store data received by sink node (i.e. sensor reading values) and store time stamp for every event to perform calculation of performance related parameters and allows access of the history values.

4. **Developing the GUI and the node's application**
   The Graphical User Interface of the monitoring tool will be developed in java programming language. The GUI allows user to display network topology, visualize sensor reading values and send control commands to nodes on the network. Node application is an embedded application that will be developed in TinyOS. This application is responsible for collecting data from the monitored phenomenon and

sends these data to the sink node and also process the command send by the user to the node.

5. **Testing the new monitoring tool.**
   The purpose of this work is to test the new WSNs monitoring tool in a small testbed with several nodes for by varying number of hops travel by the packets.

6. **Adjusting intermediate and elaborating Final Report.**
   This task refers to the adjustments necessary for the intermediate report according to the recommendations from the jury in intermediate evaluation and elaborates the final dissertation report.

Following Gantt chart depicts the activities of the second semester and their respective durations

ID	Task Name	Start	Finish	Duration
1	Adapting Octopus monitoring tool	11/02/2013	28/02/2013	14d
2	Studying and incorporating FTSP with Octopus	01/03/2013	15/04/2013	32d
3	Implementing database and measuring performance related metrics	16/04/2013	03/06/2013	35d
4	Developing the GUI and node's application	04/06/2013	08/08/2013	48d
5	Testing new Octopus tool.	09/08/2013	23/08/2013	11d
6	Adjusting Intermediate report and elaborating Final Report	09/08/2013	30/08/2013	16d

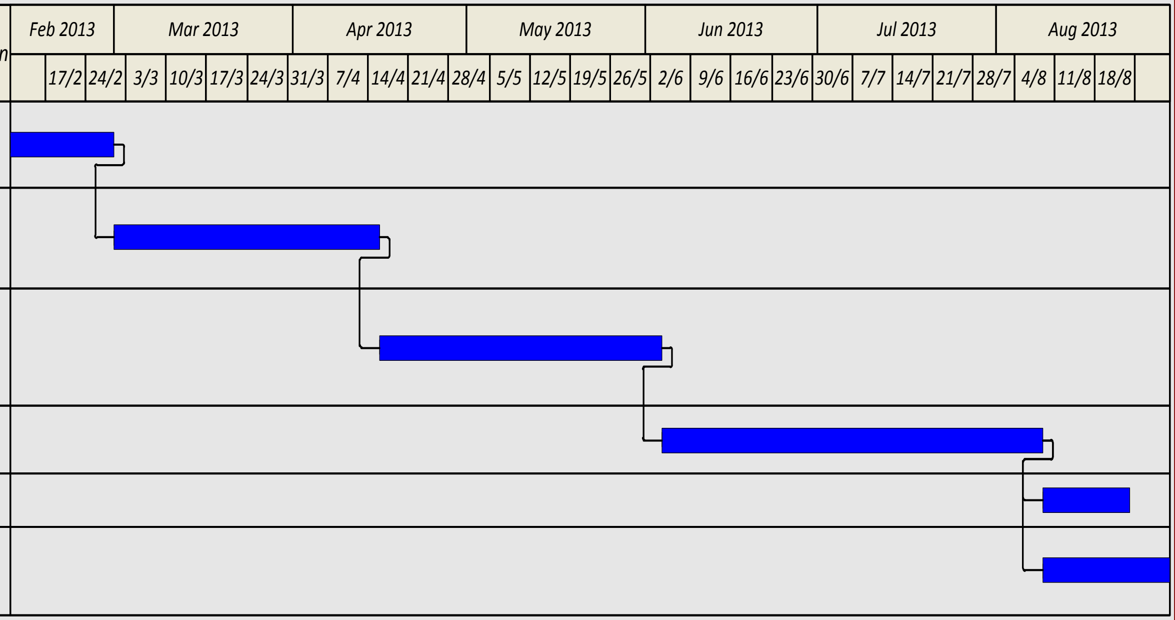

Figure 23 – works done in the second semester and their respective durations

7. Conclusion

WSNs are a network of small electronic devices, each with a micro-controller and a radio transceiver that gather data from physical environmental conditions. Each small electronic device, also known as sensor node, is severely limited in terms of computation and power resources so the information gathered by these nodes is sent to the sink node for further analysis.

Most of these small sensor nodes are lack of display which is hard for the user to analyze the node condition, or to verify if the nodes are functioning as intended. Furthermore WSNs usage in critical areas such as plant automation requires some of the performance related parameter such as message delay and message reliability must be within tolerable range.

The above mentioned issues can be solved by using a WSNs monitoring tool which allows user to monitor nodes condition, display data gathered by the nodes and perform basic control of the network. In WSNs with controlled performance the tool must also provide mechanism to measure performance related metrics.

Chapter 2 of this report discusses some of existing WSNs monitoring tools by listing their functionalities and presents a brief comparison of them, Most of the tools only provide a network monitoring data visualization functionality, which is not suitable for WSNs with controlled performance where it is needed to determine the performance of the network in real-time.

An experimental setup of Octopus was carried out to explore the possibility of using it as the base of the new tool to be developed after the tests made, it was decided that Octopus would be a good starting point to the new monitoring tool. To accomplish that, a new module with a mechanism to perform measurement of the network performance and a database to enable data history, will be added.

Chapter entitled "New Octopus monitoring tool" describes the changes made to the Octopus to make a new WSNs monitoring tool, which is able to measure performance related parameters. Two of the most used QoS related parameters are measured by this tool packets delay and the packet lost the information about these two parameters are available in live and also allows user to retrieve information for a period of time by specifying start and end time.

8. References

[1] J. Yick*, et al.*, "Wireless sensor network survey," *Computer networks,* vol. 52, pp. 2292-2330, 2008.

[2] P. Levis*, et al.*, "Tinyos: An operating system for sensor networks," *Ambient intelligence,* vol. 35, 2005.

[3] W. L. Lee*, et al.*, "Network management in wireless sensor networks," *Handbook of Mobile Ad Hoc and Pervasive Communications: American Scientific Publishers,* 2006.

[4] M. Turon, "MOTE-VIEW: a sensor network monitoring and management tool," 2005, pp. 11-17.

[5] Crossbow. (2006, August 10). *Mote-view users manual*. Available: http://www.willow.co.uk/MOTE-VIEW_Users_Manual_.pdf

[6] C. Buschmann*, et al.*, "SpyGlass: a wireless sensor network visualizer," *Acm Sigbed Review,* vol. 2, pp. 1-6, 2005.

[7] R. Jurdak*, et al.*, "Octopus: monitoring, visualization, and control of sensor networks," *Wireless Communications and Mobile Computing,* vol. 11, pp. 1073-1091, 2009.

[8] J. Pinto*, et al.*, "MonSense–Application for Deployment, Monitoring and Control of Wireless Sensor Networks," *ACM Real WSN,* vol. 2006, 2006.

[9] M. Yu*, et al.*, "NanoMon: An Adaptable Sensor Network Monitoring Software," 2007, pp. 1-6.

[10] S. Park*, et al.*, "A nano operating system for wireless sensor networks," 2006, pp. 4 pp.-348.

[11] A. G. R. Raja Jurdak, Alessio Barbirato, and Samuel Boivineau. (2009, 10 August). *Octopus developer documentation*. Available: http://www.csi.ucd.ie/files/Developer%20Documentation.pdf

[12] O. Gnawali*, et al.*, "Collection tree protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, 2009, pp. 1-14.

[13] P. Levis*, et al.*, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," 2004, pp. 2-2.

[14] J. Wang*, et al.*, "On the Delay Performance Analysis in A Large-Scale Wireless Sensor Network," in *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, 2012, pp. 305-314.

[15] M. Maróti*, et al.*, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004, pp. 39-49.

[16] A. Friends. (10). Available: http://www.apachefriends.org/en/index.html

[17] J. Elson*, et al.*, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review,* vol. 36, pp. 147-163, 2002.

[18] S. Ganeriwal*, et al.*, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003, pp. 138-149.

[19] D. Gay*, et al.*, "The nesC language: A holistic approach to networked embedded systems," in *Acm Sigplan Notices*, 2003, pp. 1-11.

[20] J. G. Ales Spetic. (2002, 01 of August). *Transact-SQL Cookbook*. Available: http://oreilly.com/catalog/transqlcook/chapter/ch08.html

[21] O. G. Rodrigo Fonseca, Kyle Jamieson, Sukun Kim, Philip Levis, and Alec Woo. (2006 10 of August). *CTP TinyOS TEP*. Available: http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html

[22] Y. Wei, "Coexistence of IEEE 802.11 b/g WLANs and IEEE 802.15. 4 WSNs: Modeling and Protocol Enhancements," 2011.

[23] P. Levis. (20 of August). *message_t TinyOS TEP*. Available: http://www.tinyos.net/tinyos-2.x/doc/html/tep111.html

APPENDIX - How to install and run New Octopus monitoring tool

To install and run new octopus monitoring tool require a working TinyOS 2.1 environment on the PC. Following files need to be place in their respective locations or directories:

- Copy the "NewOctopus" directory which contains mote and java directories to TinyOS App folder, default location is */opt/tinyos-2.1.x/apps/;*
- Copy the "lamp" which contains the Octopus database to */opt/* directory
- Copy CC2420TransmitC.nc and CC2420TransmitP.nc from energy directory into the folder*: /opt/tinyos-2.x/tos/chips/cc2420/transmit/;*
- Copy LedsC.nc and LedsP.nc from energy directory into the folder*: /opt/tinyos-2.x/tos/system/;*
- Copy PowerCycleC.nc, PowerCycleP.nc from energy directory into the folder: */opt/tinyos-2.x/tos/chips/cc2420/lpl/;*
- Copy Energy.nc from energy directory into the folder: */opt/tinyos-2.x/tos/interfaces/*

These .nc file will be replacing the same files existing in the destination folders, these .nc files are basically the same file with the one existed in the destination with new code added which requires by Octopus to work (in case you do not want to replace the file, just make a backup to restore it to their original when needed but, it will not causing any problem if you decided to replace the original files).

To install Octopus embedded application on the motes do the followings

- Go to the "motes" directory, which is located in *cd /opt/tinyos-2.x/apps/NewOctopus/motes*
- And compile the code as you would for any TinyOS application by typing *make telosb* (this was the platform used for development) but it should work also with Micaz Platform.
- *make <PLATFORM> reinstall,<NODE_ID> <your usual installation options>* for telosb mote check first USB port used by using *motelist* command to refer it in "your installation option"

- Make sure to specify the NOTE_ID = 0 for the sink node, other nodes can have other consecutive distinct IDs
- Connect sink node to the PC serial port and switch on the other nodes
- Export the MOTECOM variable in your environment: *export MOTECOM=serial@<SERIALPORT>:<SERIALPORT BAUD RATE>*
- Go to "java" and compile the java code using the Make program in case there are no class files.
- There is two java application to receive and display the data received from the network: "OctopusOutput.java" which is a console application which displays the following data to the console moteId, parentId, isSync, sendTime , receiveTime packetDelay, packetId, and rootAddress from left to right column respectively. This application is used mainly for debugging purpose and does not save received data to the database, thus no need to start XAMPP server to run this application. OctopuGUI is the GUI application of Octopus which give you more option and controls, to run this application first it required to start the XAMPP server by typing sudo /opt/lamp/lamp start and provide your password when ask to. Therein OctopusGUI can be started.
- OctopusDB can be accessed via a web browser (PHPMyAdmin) after the XAMPP is started by typing localhost in browser address bar.