REAL TIME MONITORING AND CONTROL OF WIRELESS NETWORKS

Mr. Girish Revadigar[1] and Mrs. Chitra Javali[2]
[1]Allgo Embedded Systems, Bangalore, India
girishrevadigar@gmail.com
[2]PES Institute of Technology, Bangalore, India
chitra.javali@gmail.com

ABSTRACT

Adhoc sensor networks consists of dense wireless network having tiny, low-cost sensor nodes which collect the environmental data and send it to the base station. Using such networks helps in monitoring and control of physical environments remotely with ease and accuracy. Examples include military applications, and acquiring sensing information from inhospitable locations like thick forests, active volcano regions etc. Since the devices will be battery powered and are scattered in a complex network, It is very difficult to locate each node, know about its status and how the devices are connected in the wireless network. Because of their increased importance and applications, wireless networks are becoming part and parcel of our day to day life, and hence the need of developing a system to visualise , control and monitor such networks arises. This paper presents implementation issues and author's contribution to design and implement a generic framework of the 'Network Visualization tool' to monitor adhoc wireless networks. The paper also elaborates system architecture, hardware and software organizations, and integration details of the proposed system with an exemplary wireless network based on standard IEEE 802.15.4 MAC protocol[1] to monitor temperature of different rooms in a house.

KEYWORDS

Adhoc Wireless Networks, IEEE 802.15.4 MAC, Network Visualization

1. INTRODUCTION

Wireless adhoc networks have enormous applications in today's world, to name a few – Military applications, networks to detect chemical/radio active element/explosives, to study and monitor environmental changes in forests/oceans/active volcano regions, surveillance applications, traffic control systems, and vehicle parking systems, home and industrial automations etc. Such adhoc wireless networks consists of large number of cheap, micro-controller based low power battery operated wireless devices which are very small in size, less capable, and may or may not have a sensor based on application.

Recent advancements in wireless networking technology has enabled us to use wireless connectivity in almost all our applications. The ease of integration, support from multiple platforms, interoperability and co-existence with other technologies have made these more popular. At the same time, as the complexity of such networked systems increases, the effort needed to monitor and control such systems also increases. Since the topology of adhoc wireless network is very complex with numerous devices which are scattered in the large geographical area, It will be very difficult to analyse how the wireless nodes are connected in the network, what is the status of each and every node, how is the network behaviour at any given time. If the

DOI : 10.5121/ijesa.2012.2103

application is critical and somehow if a node has gone bad, then the loss of such node data may cause serious complications.

In this paper, we describe our design, implementation and integration details of a 'Generic Network Visualization tool' for monitoring adhoc wireless networks taking an example of a wireless personal area network based on IEEE 802.15.4 MAC protocol using AllGo's wireless nodes based on Freescale's MC1321X MCU[11]. This paper also explains about hardware and software platforms used, issues involved, and our solution to address the problem stated.

Section 2 explains the system architecture of Generic Network Visualization tool, section 3 describes target subsystem representing a wireless network. In section 4, the Host subsystem part featuring a PC based application details are present. Section 5 describes the integration and testing of our Network Visualization tool with an example of wireless network based on IEEE 802.15.4 MAC protocol. Section 6 illustrates a real time application of temperature monitoring network and Section 7 concludes the paper.

2. NETWORK VISUALIZATION TOOL ARCHITECTURE

The framework of Network Visualization tool is designed in a generic way such that it can be easily integrated with any type of existing wireless network protocols like IEEE 802.15.4 MAC, SMAC, ZigBee[4]/ZigBee Pro[5], Z-Wave etc. Each component of the tool is designed taking care of integration with the upcoming wireless technology protocols also. Figure 1 shown below describes the system architecture.

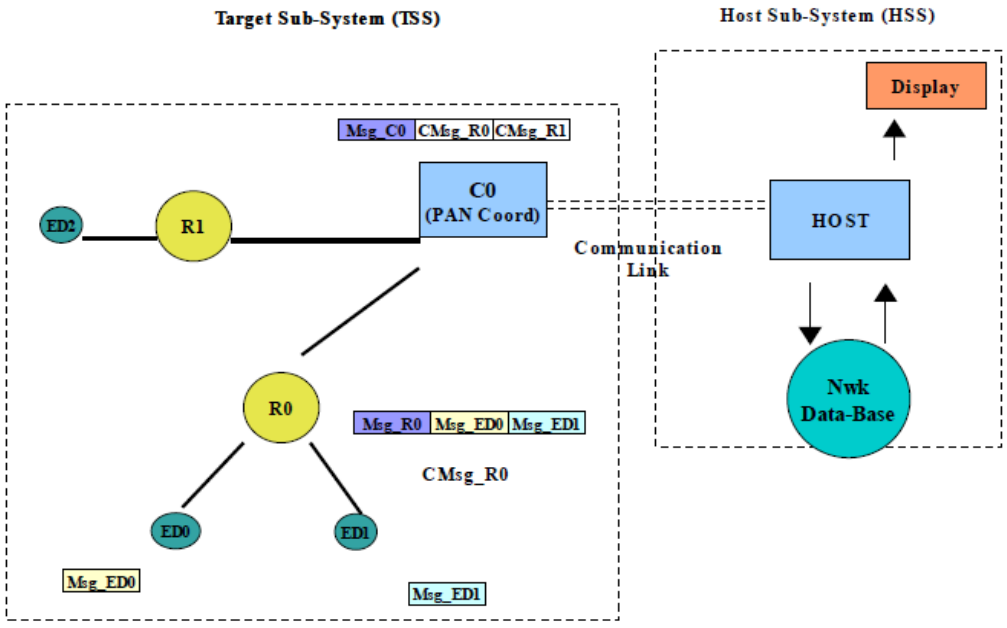

Figure 1. Generic Network Visualization tool architecture

The tool consists of two mutually communicating components, a Target subsystem and Host subsystem.

Target subsystem resides on all the wireless nodes of the network viz. Coordinator, routers and end devices and it is a distributed subsystem. From a high level, it performs the task of sending network related information to the host. Host subsystem resides on the host PC/Laptop and is responsible for processing and displaying the network related information. Since the information received from target subsystem will be in a distributed form, host subsystem performs the function of analysing and interpreting the distributed information and display it in a meaningful way. Framework is independent of the communication mechanism between the two subsystems, ex. The network coordinator device can communicate with host via serial port, but the design is portable to any other future communication links also.

Since it is a visualization tool, most of the traffic will be from target subsystem to host subsystem. However a low bandwidth reverse flow is also required to configure target subsystem behaviour based on user inputs provided to the host subsystem.

3. TARGET SUBSYSTEM (TSS)

Target subsystem is the most basic component of our visualization tool, since all the messages and information related to network are all constructed and sent to the host subsystem for interpreting the same. The different design constraints of target subsystem are explained in detail in the following sub sections.

A target subsystem resides in the actual network that we are going to visualize. Hence the target subsystem should be least intrusive to the actual network functioning, ex. In a multi priority system, message logging by the target subsystem will be the lowest priority task. In a single priority system, logging by the target subsystem will be the task executing after the devices have executed all the pending items.

3.1. Bridge Node (TSS-BN)

This is the node in the wireless network that communicates with the host subsystem(HSS) and all the other nodes in the TSS send (over the air) the data to be logged to this particular node. Let's call it the TSS-BN (TSS Bridge node). It can be any node but the main network coordinator device is a preferred choice for TSS-BN. By the way of network functioning (in most of the applications) network coordinator should have lot of network information that has to be logged. In such a case the tool bandwidth requirements will be greatly reduced by having coordinator perform as a TSS-BN also.

3.2. TSS message types

The messages exchanged between TSS and HSS are classified as below

1. Information Logs (Tx): These are the logs for network visualization. Two sub-types of logging are possible - event based logging and periodic logging,
2. Debug Logs (Tx): Software debug logs
3. Configuration Msgs (Rx): Used by host to configure logging parameters (log frequency, debug logs ON/OFF, Control signals etc.)

To display the network configuration the HSS will require information about neighbours/child devices for each device in the network. This will be done once during initial set up of the network. Subsequently, only 'differential configuration information' will be sent to HSS. It means that each device will send configuration information only if the configuration changes. This can

be viewed as a event based logging. It means that a device keep checking for certain events to happen (add/removal of a node, start_log_signal, refresh_signal, low_instantaneous_LQI etc.) and send data logging message only if the event occurs. This would also mean that the TSS logging tasks will have some logic embedded and will lead to increase in memory and MIPS requirement. However it will reduce the number and content of logging messages. Event based logging is made possible by the use of TSS-Call Back (TSS-CB) functions provided by the TSS to the network application.

To display certain network attributes (average LQI, network statistics, average_ON_Time) a periodic logging is required. This implies a timer based logging of the required attribute.

To allow a limited debugging each device can log data independently that will not be interpreted by HSS. It will be displayed as it is, in a separate window and can be turned ON/OFF dynamically on user requests. (It can be used to track the progress of any individual device as if the device itself is connected to serial link).

3.3. TSS Message structures

To facilitate all the requirements at the target side, generic message structures are designed which can be used for specific purposes, for example, the Coordinator Device can use a stucture called "DevInitMsg_t" as shown below to log its initial data tot the host when it enters the TSS state machine's STRT_LOG state for the first time.

```
typedef struct DevInitMsg_tag
{
uint8_t devid[2];          // variable to hold the device id (short address)
uint8_t devtype;           // variable to hold the device type
uint8_t numchilddev;       // variable to hold the number of child devices.
} DevInitMsg_t;
```

3.4. Classification of TSS Messages and Functions

The messages used in TSS are classified into two types as 'Control Messages' and 'Log Messages'. Also the there are two types of functions used with these message structures, viz 'APIs' and 'Callback functions'.

Control messages are those which are used by the devices in the network for the purpose of keeping track of the Host Subsystem. These can be as follows,

1. Control messages Request type - used by the end devices most of the time for querying the coordinator about the Host subsystem status
2. Control Messages Response type – used by the coordinator device to send the response back to the end devices in the network which send the query requests.

Log messages are the types of messages used by almost all the types of the devices in the network. There can be different types of log messages like the ones listed below,

1. The log message used by the coordinator and other devices in the network for logging their initial data,

```
typedef struct DevInitMsg_tag
{
        uint8_t devid[2];
```

```
        uint8_t devtype;
        uint8_t numchilddev;
} DevInitMsg_t;
```

2. End device can log its Link quality using a specific log structure as shown below.

```
typedef struct LinkQualityMsg_tag
{
        uint8_t len;
        uint8_t lqi;
} LinkQualityMsg_t;
```

3. Similar message structures can be used for different application purposes also, for example, for logging the temperature value received from a sensor etc.

```
typedef struct AppTemperatureMsg_tag
{
        uint8_t len;
        uint8_t temperature;
} AppTemperatureMsg_t;
```

3.5. TSS operations

TSS consists of all the devices in the network including the Pan coordinator, and End devices. (routers also form the part of it if the network is multi hop). Each device in the TSS has a basic state machine for their functionality but the individual functionality of the devices varies in different states based on the device type, viz Coordinator/End device or router. The basic state machine and also different functionalities of the devices in these states are explained in the subsections of this chapter.

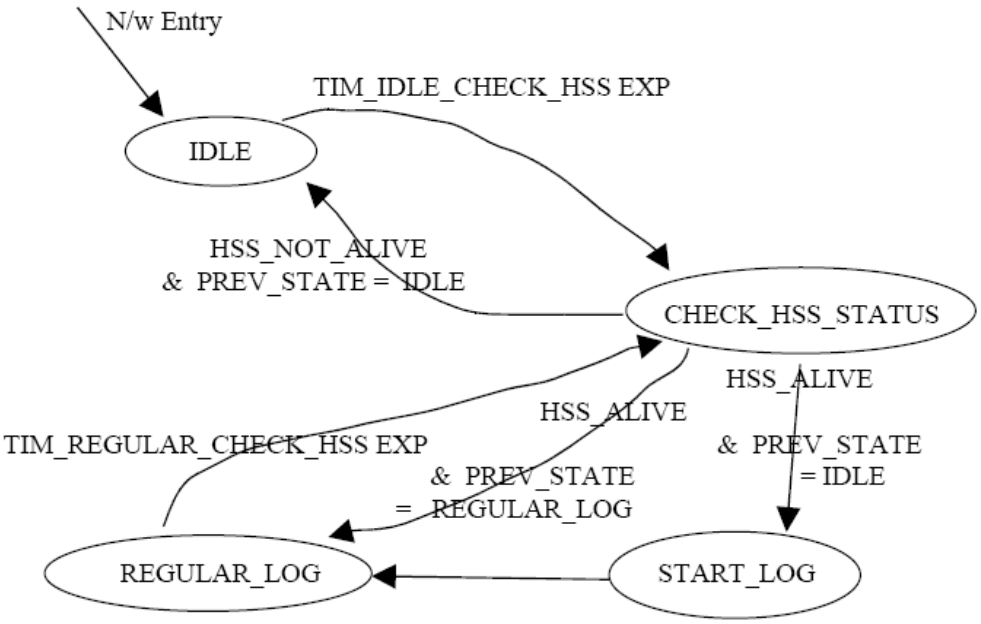

Figure 2. Device state machine

The basic state machine is same for all the devices, but the individual functionality of each device varies. This is clearly explained in later sections. The device will be in the state called "INVALID_STATE" until it joins the network, after it joins the network, the first state it enters is IDLE. The subsequent state transitions are explained as follows.

3.5.1.State IDLE

In this state the device will not do anything except waiting for the HSS to become active. A timer set to expire at every 10 seconds is used to change the device tool state to a state called 'CHECK_HSS_STATUS' where the actual polling for HSS status happens. The device stays in this state until it receives 'HSS alive' response for the query of HSS status.

3.5.2.State CHECK_HSS_STATUS

In this state the device sends a query for HSS status and according to the present status receives one of the three possible responses as 'HSS alive', 'HSS not alive', or 'Refresh log'. According to the type of response obtained for the query of HSS status the device then decides its next state to jump. If the response is 'HSS not alive' then the device jumps back to 'IDLE' state. If the response is found to be 'HSS alive', then the device chooses its next state as 'START_LOG' if the initial log is not complete(i.e., if the previous state from which it entered the current state was 'IDLE'.), or the device may directly jump to another state called 'REGULAR_LOG' if the initial log is already completed.(i.e., if the previous state from which it entered the current state was 'REGULAR_LOG'.)

3.5.3.State START_LOG

In this state the device just does its initial log where it logs all its data related to network configuration and other useful information maintained by the device.

3.5.4.State REGULAR_LOG

In this state the device keeps waiting for any change event to occur in the network and if it finds the one, it just logs that data. Also the periodic logging is provided by means of a timer set for a particular time (say every 5seconds). To keep track of HSS status, a timer is set to expire at every 20 seconds, after each time the timer expires, the device state changes to 'CHECK_HSS_STATUS' where the actual polling for HSS status happens. If the response for the HSS status query is found to be 'HSS alive' and the 'Initial log complete bit' in the device tool status register is set, (i.e., the previous state from which it entered the current state was 'REGULAR_LOG') the device then enters 'REGULAR_LOG' state again.

3.6. Data logging mechanism

To reduce the number of air accesses and the logging latency, a buffer based logging mechanism is employed by the TSS. The information to be logged will be stored in a 'log/dump-buffer' in an appropriate message structure:DEV-SINGLE-MSG. Multiple such messages will be collected in the 'log-buf'. Then either timer-based or total-message length based (or both) logic will be employed to actually send this information over the air to the parent/coordinator. This message will be sent via another message structure:DEV-OTA-MSG. A few benefits of such a mechanism are:

1. Better control for over-the-air message lengths and frequency.
2. Better control over the logging latency.

A priority logging can be easily introduced by having a higher priority buffer that has to be flushed out first.

3.7. Timer based activities in TSS

All the time based activities in TSS are specified in terms of 'TimeTicks' of a specified duration. The 'TimeTick' duration can be quite flexible if a regular processor timer is chosen to provide the basic 'TimeTick'. But since an application will require the end-devices to sleep it might not be appropriate to use the regular processor timer. In such a case a basic 'TimeTick' will be equal to 'sleep duration' of the device. Thus, to be able to efficiently handle both these cases the period based events will be tracked using the 'TimeTicks' instead of the amount of time (s or ms).

3.8. TSS interactions with applications

TSS is a part of wireless device will be at the same level as the 'APP' layer in the application domain. Figure 5.2 shows the TSS interactions with the application. As mentioned above, TSS needs to be easily integrating onto existing application and hence the interfaces between TSS domain and Application domain needs to well defined and as little as possible. TSS will need NWK APIs to send (**Tx**) log messages over the air, to query some typical parameters from the network data-base (Neighbour tables, routing tables etc.) or to receive (**Rx**) some configuration parameters/control signals over the air from coordinator.

On the other hand the application might also need to interact with TSS. For ex. to inform TSS of a network configuration change (association successful, addition/removal of node), register a data structure to log (actual data transfer takes place through shared memory) etc. Interactions between tool domain and application domain will be through 3 mechanisms: NWK APIs, Event Call Backs and Shared Memory.

1. **Event-CB Functions:** Call-back functions for application to inform TSS task of different events. Application will need to use these call-back functions whenever any event takes place that needs to be communicated to TSS.

2. **Network-Usage APIs:** APIs for using network/applications services. These APIs will be using the network service provided by the application and hence will need to be adapted based on the network we are integrating the TSS with.

3. **Shared Memory:** To share application data structures to be logged. To share data received over the air meant for TSS.

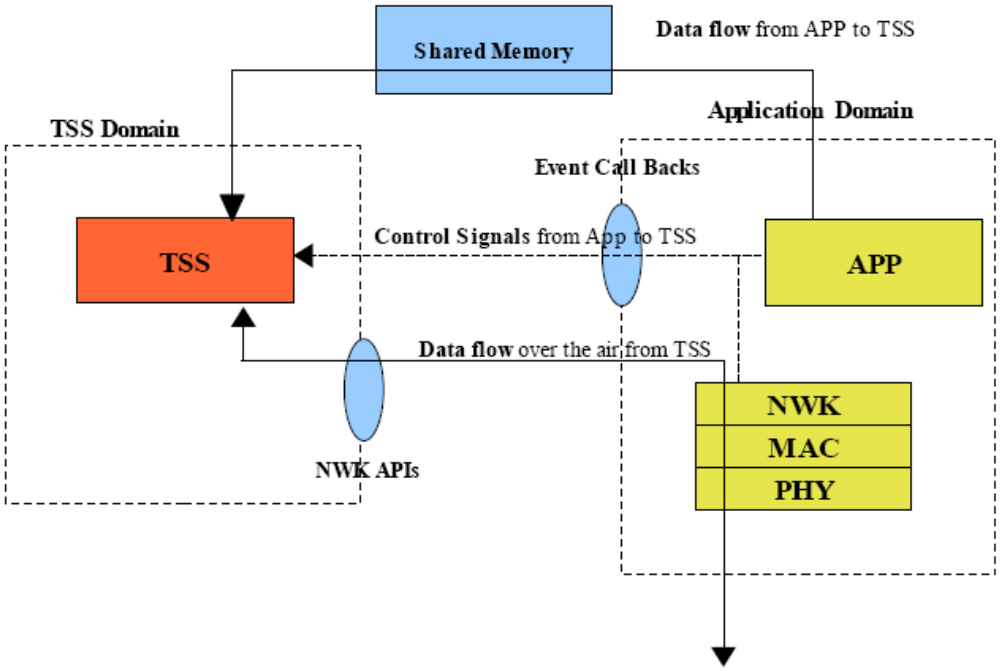

FIGURE 3. TSS AND APPLICATION INTERACTIONS

4. HOST SUBSYSTEM

The host subsystem essentially consists of the following parts,

1. A host Pc/Laptop on which the network visualization tool runs
2. A serial port configuration module and Interpreter written in Java,
3. A database for storing the data
4. Prefuse tool kit – A Java based GUI tool (Graph visualization tool),

5. INTEGRATION AND TESTING

The tool developed has been tested by integrating with a simple star network based on standard IEEE 802.15.4 MAC protocol. Following are the hardware and software used for the integration and testing:

1. A PC for development and using as a host system,
2. AllGo's wireless modules based on Freescale's MC1321x MCU, an 8 bit microcontroller of HCS08 family,
3. P&E USB multilink debugger,
4. IEEE802.15.4 MAC code base for HCS08 generated from Freescale's Beekit,
5. JDK 1.5,
6. Prefuse toolkit for Java.

A simple star network set-up as shown in Figure 5. was used for integration and testing. The network coordinator is connected to host Pc via serial port. All other devices in the network are connected to the coordinator through wireless network.

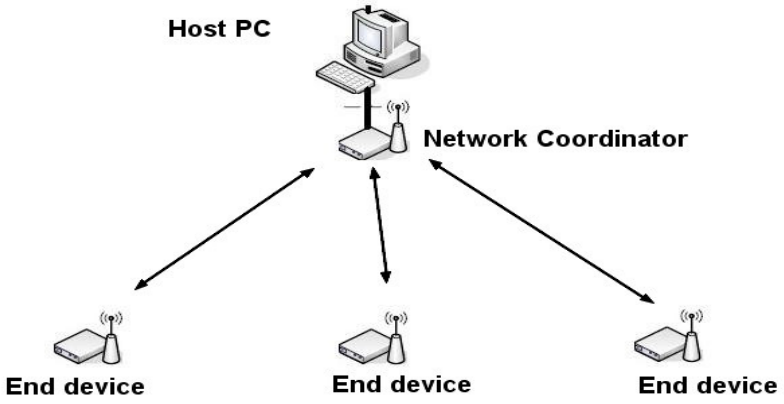

Figure 5. Example network : Star network based on IEEE802.15.4 MAC

Each phase of development of this project have been undergone various types of testing from the beginning of high level design to till the the end of interfacing the the Target Sub System part with the Host Sub System for its proper functionality. The types of tests can be classified as

1. Module tests
2. Integration tests.

5.1. Module tests

These are the tests that are carried as and when the individual modules of the project were developed. For example, following are the module tests carried out during the development process of the project.

1. Testing the basic state machine of Target subsystem: The first part of the module tests is to test the state machine of the target subsystem. This is tested by printing the debug strings on the hyper terminal from individual states when the device enters these states. The snapshots of test results are shown below. The state transitions are tested by manually setting the required conditions on the Coordinator device, so that when the end device requests, it sends the response according to the condition set manually. And at the End device we can clearly notice these state changes. Following output is obtained when we captured the data printed on hyper terminal while testing the end device's state machine (Figure 5):
   The Myapp demo application is initialized and ready.
   Start scanning for a PAN coordinator
   Sending the MLME-Scan Request message to the MAC...Done
   Found a coordinator with the following properties:

   Address.....0xCAFE
   PAN ID.....0xBEEF
   Logical Channel...0x0B

Beacon Spec.....0xCFFF
Link Quality.....0xD3
Associating to PAN coordinator on channel 0x0B
Sending the MLME-Associate Request message to the MAC...Done
Successfully associated with the coordinator.
We were assigned the short address 0x0002
Ready to send and receive data over the UART.
Entered IDLE :
Entered CHECK_HSS_STATUS :
QUERY_HSS_STATUS request sent.. :
waiting for cmd ack. :
sent poll req :
HSS not alive ...
Entered IDLE :
Entered CHECK_HSS_STATUS :
QUERY_HSS_STATUS request sent.. :
waiting for cmd ack. :
sent poll req :
HSS alive ...
Entered STRT_LOG :
Initial Log Complete ...
Entered REGULAR_LOG:
Entered CHECK_HSS_STATUS :
QUERY_HSS_STATUS request sent.. :
waiting for cmd ack. :
sent poll req :
HSS alive initial log already done.
Entered REGULAR_LOG:
Entered CHECK_HSS_STATUS :
QUERY_HSS_STATUS request sent.. :
Refresh Log Found..
Entered STRT_LOG :
Initial Log Complete ...

2. Testing the data Logging mechanism of the Coordinator device : The following is the content of log buffer of the Coordinator when the Coordinator initial data is logged according to the structures shown below. The structures are filled using the Functions and API's designed.

3. Smsg Structure used for filling the Coordinator Device initial data to the log buffer.

```
typedef struct DevSMsg_tag
{
uint8_t msgid; // 09 for devinitMessage type,
uint8_t len; // 04 for devinitMessage type,
union
{
DebugMsg_t debugMessage;
NWKparamsMsg_t nwkparamsMessage;
Child_OTAMsg_t childOTAMessage;
EndDevInitMsg_t enddevinitMessage;
DevInitMsg_t devinitMessage;
AppTemperatureMsg_t apptemperatureMessage;
```

```
LinkQualityMsg_t lqiMessage;
} DevSMsgData;
} DevSMsg_t;
typedef struct DevInitMsg_tag
{
uint8_t devid[2]; // 0xFECA in little endian mode,
uint8_t devtype; // 0xAA - coordinator device type
uint8_t numchilddev; // present no. of child devices , 01
} DevInitMsg_t;
```

4. Structure used by the Coordinator device for logging the data to the host through serial link, the below output shows the content of the message structure "DevHSSLogMsg" when filled using the buffer contents.(Smsgs). This clearly shows the log buffer filled with proper values using the functions implemented. This also verifies the correctness of the API's designed for logging mechanism.

```
typedef struct DevHSSLogMsg_tag
{
uint16_t hssmsgid;
uint8_t logSmsgdata[MAX_DEV_HSS_MSG_SIZE];
} DevHSSLogMsg_t;
output :
in Send_HSS_Msg
tail // tail buffer index
00
head // head buffer index
06
count
08 // difference count
33 // Hss msg id[0]
44 // Hss msg id[1]
09 // Smsg id - 09 for devinitMessage type,
04 // Smsg len - 04 for devinitMessage type,
FE // devid[0] – 0xFE coordinator short address lsb
CA // devid[1] – 0xCA coordinator short address msb
AA // coordinator device type
01 // no. of child devices – 01
```

5.2. Integration tests

These are the tests which are carried out at the time of integration of Target Sub System and the Host Sub System. The various phases of network formation test results are shown below.

1. Starting the Pan Coordinator : Initially the Host is started and the visualization graph is empty since there is no network to show. Now connect the device in which Coordinator software is installed, to the Host PC through Serial cable, and switch on the device. (Coordinator is mains powered always.) When the coordinator starts and enters its TSS State machine, it logs its initial data to the Host regarding its 2 byte Short Address(id), device type, and the number of child devices. The graph now shows the Pan coordinator as in Figure 6.

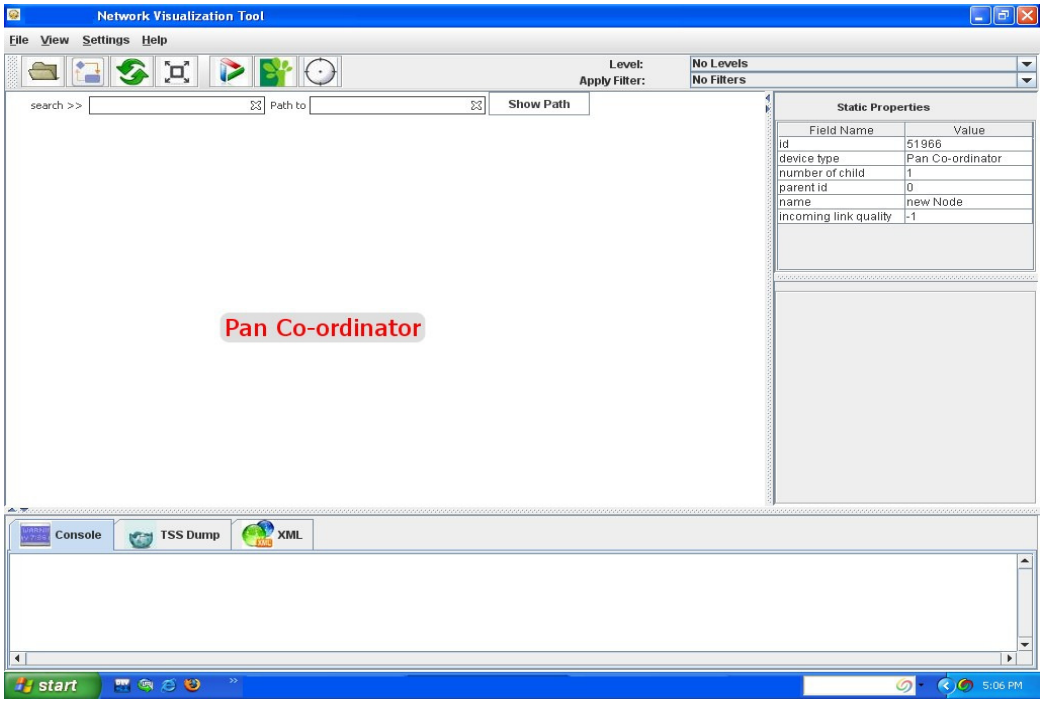

Figure 6. Snapshot of Visualization tool window showing the Coordinator device

2. Starting the End devices : Switch on the Kumbha2 Board in which the End device software is installed. End device is battery powered. The end device when joins the network and enters its state machine, it sends the initial data to Coordinator regarding its 2 byte Short address (id), and the device type. The end device also starts logging its actual link quality to its log buffer. This data is sent by over the air messages to the coordinator from the REGULR_LOG state of the device. The coordinator in turn collects these messages and logs these data to Host. The host now interprets these messages and updates its GUI graph by adding the end device to the Pan coordinator. This is shown in the next screen shot Figure 7 of the GUI tool. We can also see the attributes of each devices in the right side of the window. Figure 8 shows dump messages from the end device.

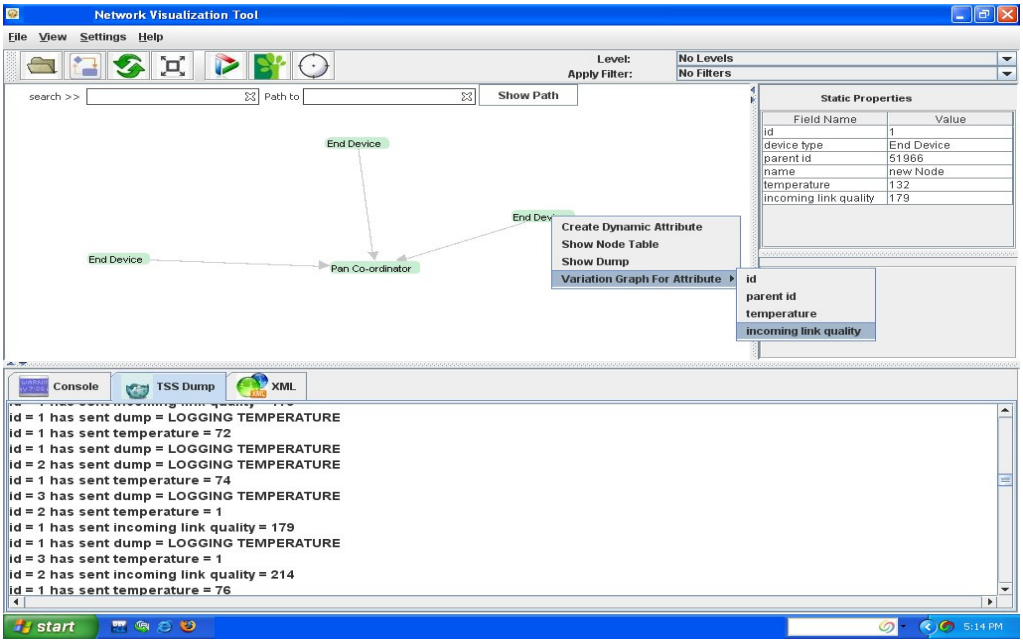

Figure 7. Showing 3 end devices connected to coordinator and sending status messages.

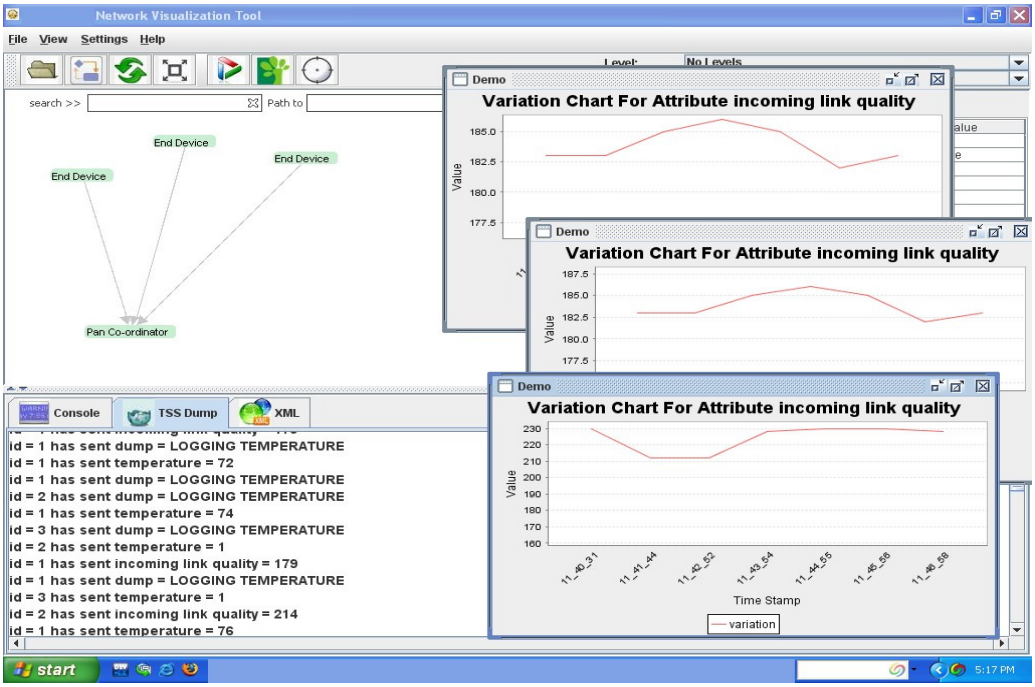

Figure 8. Showing the link quality variations for each connected device

6. REAL TIME APPLICATION OF TEMPERATURE MONITORING NETWORK

The block diagram of a real time application of temperature monitoring network is as shown in Fig.9.

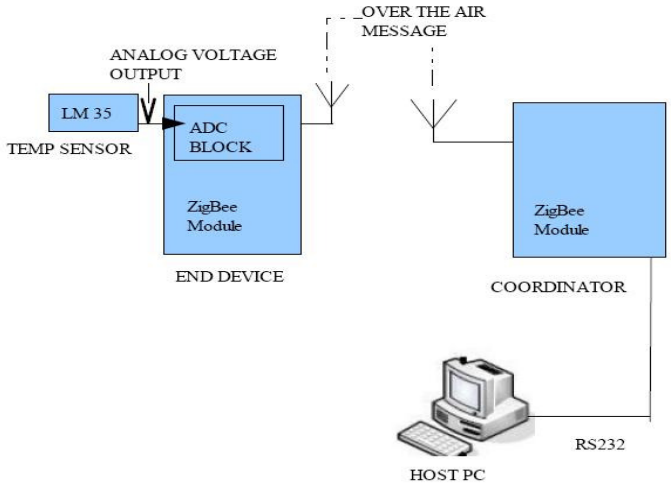

Figure 9.Block diagram of real time application of temperature monitoring network

The wireless network is a star network consisting of one network coordinator and many end devices connected to it. The coordinator is connected to the PC to log any incoming messages from the end devices. The end devices are interfaced with temperature sensor circuit. The LM35 is a precision-integrated temperature sensor, whose output voltage is linearly proportional to the Celsius(Centigrade) temperature. The output voltage is 10mV per degree centigrade. LM35 temperature sensor integrated with the wireless module is as shown in Fig.10

Figure.10 Wireless module with temperature sensor

The visualisation tool shows the temperature monitored by the sensor nodes from each of the rooms as shown in Fig.11.If the temperature exceeds the normal temperature then it shows an alert message as shown in Fig.12 or switches on a cooler depending on the way the application is programmed. The temperature variation can also be viewed in a graph as shown in Fig.13

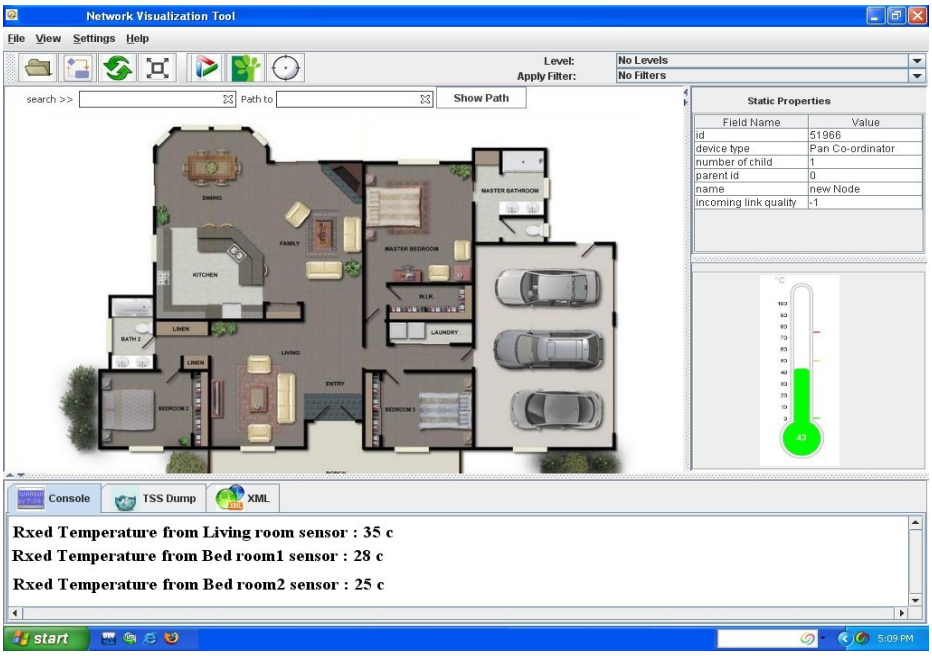

Figure 11.Network visualisation tool at the host monitoring the temperature

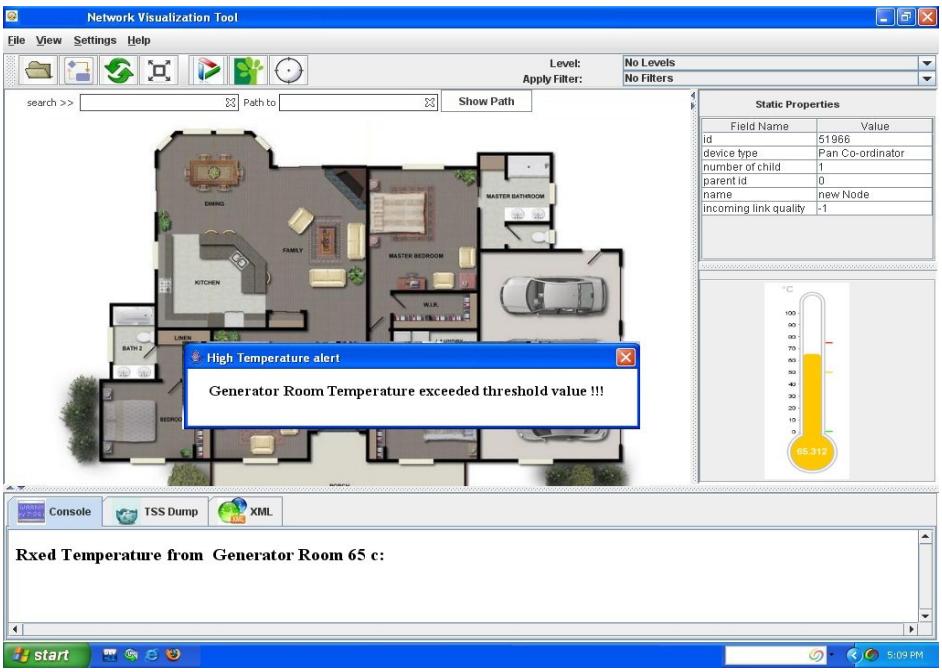

Figure 12.Network visualisation tool at the host showing the temperature exceeded

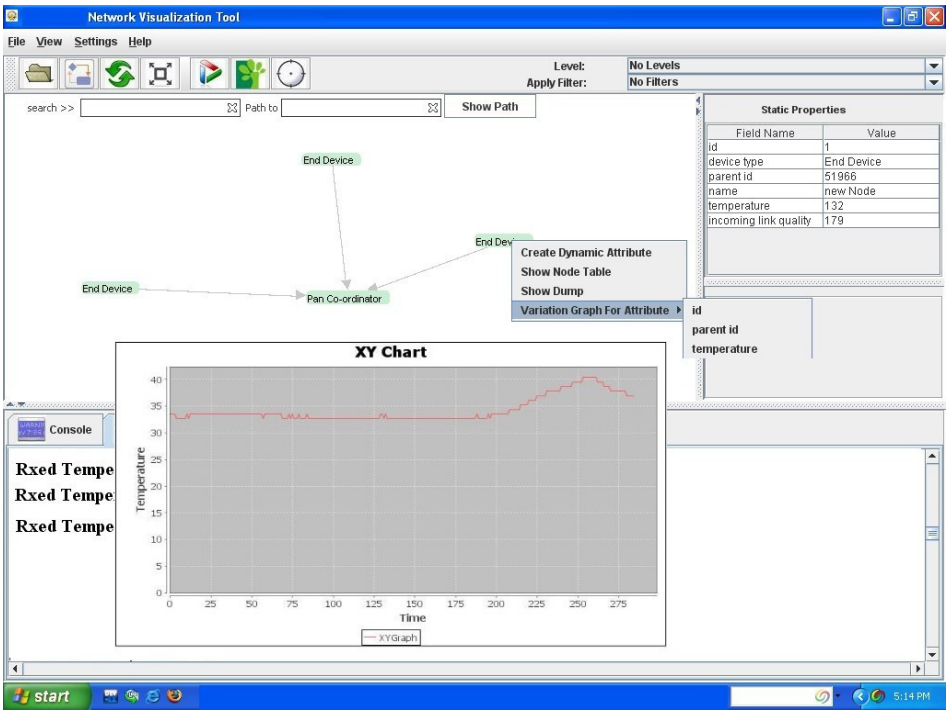

Figure 13.Network visualisation tool at the host showing the temperature in a graph

7. CONCLUSIONS

Thus the Tool can easily communicate with the application also for its different purposes. Like obtaining the short address of the device, or obtaining the link quality at the data indication, or obtaining the application specific data such as temperature or other parameter to be logged etc. The tool takes the essential basic network functionality and builds its structure on top of that. Since there is only change in the network specific API's and Callback functions needed for the integration, this tool developed finds great importance with the focus on future types of sensor networks also. The tool developed is presently tested by integrating with an 802.15.4 MAC[1] build based star network configuration. The following features of tool are verified systematically by the test results:

1. The basic state machine of each device is tested for proper functionality, which is the basic requirement for the tool to operate for any type of network since this is network platform independent.

2. The generic message structures developed are functioning as desired for the specific build used (802.15.4 MAC[1]). This proves the approach of designing platform specific API's and Callbacks.

Thus, this network visualization tool can become a standard framework for developing the network visualization and monitoring tool for any of the Future network types also.

REFERENCES

[1] Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), Institute of Electrical and Electronics Engineers, IEEE 802.15.4, 2003

[2] ISO/IEC 10039:1991, Information technology—Open systems interconnection —Local area networks— Medium Access Control (MAC) service definition.

[3] ISO/IEC 15802-1:1995, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common specifications—Part 1: Medium Access Control (MAC) service definition.

[4] ZigBee Specification, ZigBee Standards Organization., 053474r06 Version 1.0, Jun 2005.

[5] ZigBee Network layer specifications, revision 10, version 1.00, ZigBee document, 2004.

[6] N. Bulusu, J. Heidemann and D. Estrin, GPS-less Low Cost Outdoor Localization For Very Small Devices, *IEEE Personal Communications*, Special Issue on Smart Spaces and Environments, Vol. 7, No. 5, October 2000, pp 28-34.

[7] ISO/IEC 8802-2:1998 (IEEE Std 802.2™, 1998 Edition), Information technology—Telecommunications and information exchange between systems —Local and metropolitan area networks—Specific requirements—Part 2: Logical link control.

[8] ISO/IEC 9646-1:1994, Information technology—Open systems interconnection—Conformance testing methodology and framework— Part 1: General concepts.

[9] ISO/IEC 9646-7:1995 (ITU-T Rec. X.296 (1994)), Information technology— Open systems interconnection—Conformance testing methodology and framework—Part 7: Implementation conformance statements.

[10] D. Estrin, R. Govindan, J. Heidemann and S. Kumar, Next Century Challenges: Scalable Coordination in Sensor Networks, *In Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99)*, Seattle, Washington. August 1999.

[11] Girish Revadigar, Chitra Javali, Generic Network Visualization Tool for Monitoring Adhoc Wireless Networks, *In Proceedings of the Third International Conference on Wireless Mobile Networks, (WiMone 2012)*, Bangalore, India. January 2012.

Authors

Mr. Girish Revadigar

Mr. Girish holds a BE degree in Electronics and communication from STJIT Ranebennure, Karnataka India, and M.Tech. Degree in Industrial electronics from SJCE Mysore, Karnataka India. He is currently working as a senior software engineer for AllGo embedded systems, Bangalore. His areas of interest include Low rate wireless personal area networks, Embedded systems, and Device drivers.

Mrs. Chitra Javali

Mrs. Chitra Javali has obtained BE degree in Electronics and communication from KLE college of Engineering Belgaum, Karanataka India. She holds a M.Tech Degree In Networking and Internet Engineering from SJCE Mysore, Karnataka India. She is currently working as an Assistant Professor in Computer science department of PES Institute of technology, Bangalore. Her areas of interest include, Networking, Body sensor networks, Computer networks and Embedded systems.