Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Kommunikationsnetze
Prof. Dr.-Ing. Wolfgang Kellerer

Home Security & Surveillance
Final Report Group 02

Berke Aydinli,
Ermin Sakic

WSN Final Report

Semester:	SoSe 2013
Group:	02
Authors:	Berke Aydinli
	Ermin Sakic
Advisors:	Prof. Dr.-Ing. Wolfgang Kellerer
	Dipl.-Ing. Luis Roalter
Date:	July 15, 2013

Contents

1 Introduction

1.1 Motivation

The SS13 WSN-Lab project "Home Security & Surveillance" is a demonstration of a possible use of sensor networks in an indoor monitoring and intruder detection scenario, aiming to offer a reliable and robust control & safety set of tools for companies and private home owners. Physical security is nowadays a big concern and one of the areas where subtle wireless sensor nodes can be efficiently deployed, thus resulting in a reasonable alternative to the state-of-the-art techniques and setups. Instead of using expensive security cameras, base stations and alarms, which usually are highly dependant on wired connections, proprietary protocols, external power supplies and complicated setup procedures, we offer a flexible, highly-scalable and energy-concerned solution, easily manageable directly or remotely from cloud.

The nodes used in this scenario were equipped with IR and accelerometer sensors, and can be positioned on windows, doors and various other "fragile" parts of a home, in order to detect motion and intelligently inform the home owners of suspicious activities. In addition to this purpose, the nodes can also provide some other useful information, like temperature and illumination readings of different spots in the house. The house monitoring data could help in attaining an energy-efficient management of indoor areas by regulating heating and lightning as well as offering fire protection. The proposed solution could easily be integrated in more complex smart house systems, which usually involve some kind of actuators as well, but also complement the arising Internet of Things concepts, focusing on deeply connected and autonomous network organization and management.

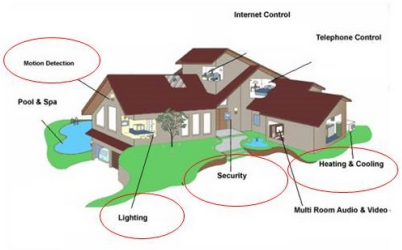

Figure 1.1: The basic concept of Home Security & Surveillance

1.2 Scenario

As depicted in Figure 1.2, our "all-in-one" solution proposes a distributed system of mutually-dependant components, interacting with and supporting each other. These components are a ZigBee compilant wireless sensor network for data accumulation and a typical server-client infrastructure for packet parsing, remote communication and graphical interfacing. Multiple nodes, positioned in the target environment, make use of simple routing algorithms in order to support an event-based communication and a command & response management system. The nodes, equipped with four different types of sensors, transmit responses to queries received from the main node we call "commander". As the name says, the commander device needs to be able to parse neighborhood information and build routing tables, implicitily creating a picture of the network topology and create and send out the commands to nodes in its vincinity. Thus, our system utiliizes bidirectional 1:N / N:1 communication. In addition to the above purpose in context of the wireless networking, it also needs to communicate with a Raspberry device using serial connection, parse the incoming commands and prepare response strings for the GUI. The Raspberry device serves as an interface to Client software communicating over Internet. It is able to make use of the UART interface in order to send and receive data to and from the wireless sensor network, manage sockets and, using TCP/IP stack based-communication, send the data to a GUI client running on a remote machine. All parts of the system were self-implemented, totalling in over 2500 lines of C and Python code. The nodes participating in the wireless network are running the Nano-RK real-time operating system and make use of Berkeley MAC contention protocol for shared medium access. Server and GUI applications were written for the purpose of demonstrating the remote management and surveillance capabilities. These make heavy use of Socket, TkInter, PySerial, PythonMegaWidgets and various other open-source modules.

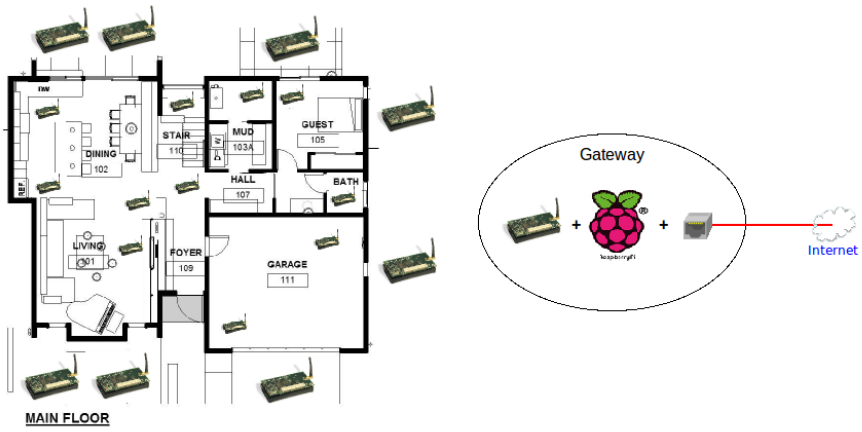

Figure 1.2: Sample Scenario

2 Implementation

2.1 Overview

As previously mentioned, we differ between two types of nodes, the master node (commander) and the slave nodes, which take care of data measurements, packet forwarding, event creation and query responses. We distinguish between temperature and brightness nodes, which should ideally be located on various locations inside the house, and outer nodes with equipped IR and acceleration sensor which can sense intruders and subsequently alert the commander node.
The middleware server application makes use of multi-threading in order to receive asynchronous input data streams on both TCP socket and the UART interface, but also to reformat and forward command queries to commander node and prepare the response and event data for simple parsing on the client side.
The graphical user interface (GUI), written in Python, can be used to easily send commands, manage the nodes individually by means of turning the event messaging system on and off when it is not needed. It provides the user with an easy-to-use interface and can handle events and update the main window correspondingly. Some additional features, i.e logging were implemented in order to ease the debugging step of implementation.

2.2 Infrared-Acceleration Sensor Workflow

Writing multi-tasking applications with hard real-time constrains was made easier by making use of existing Nano-Rk C Libraries. The MicaZ nodes used in our project are based on typical 8-Bit ATMega Microcontroller Units (model 128L), which are fully supported by the Nano-Rk and allow for easy configuration of task-specific settings like task stack size, the maximum number of tasks allowed to run at the same time, task-context switching, preempting etc. In this section, the basic workflow of the data-collection nodes, equipped with infrared, temperature, brightness and acceleration sensors will be described. In principle, all data collection nodes have the following four tasks running:

- Initialisation Task: Associates each node with their corresponding neighbors in network by broadcasting the "hello" packets. Establishes first contact to newly added nodes.

- Receive Task: Allows for packet receiving in fixed time intervals, distinguishes between multiple package classes and command types. Neighbor differentiation, the sense of existing topology and the population of routing tables are also achieved inside this task.
- Transmit & Forward Task: Enables transmitting of various kinds of packages (info, routing, event-based payload) in any direction. The target connection is extracted from the routing tables and set beforehand in the RX task.
- Warning Task: Reacts to spikes in temperature and IR readings and sets the flags for event package transmission to commander.

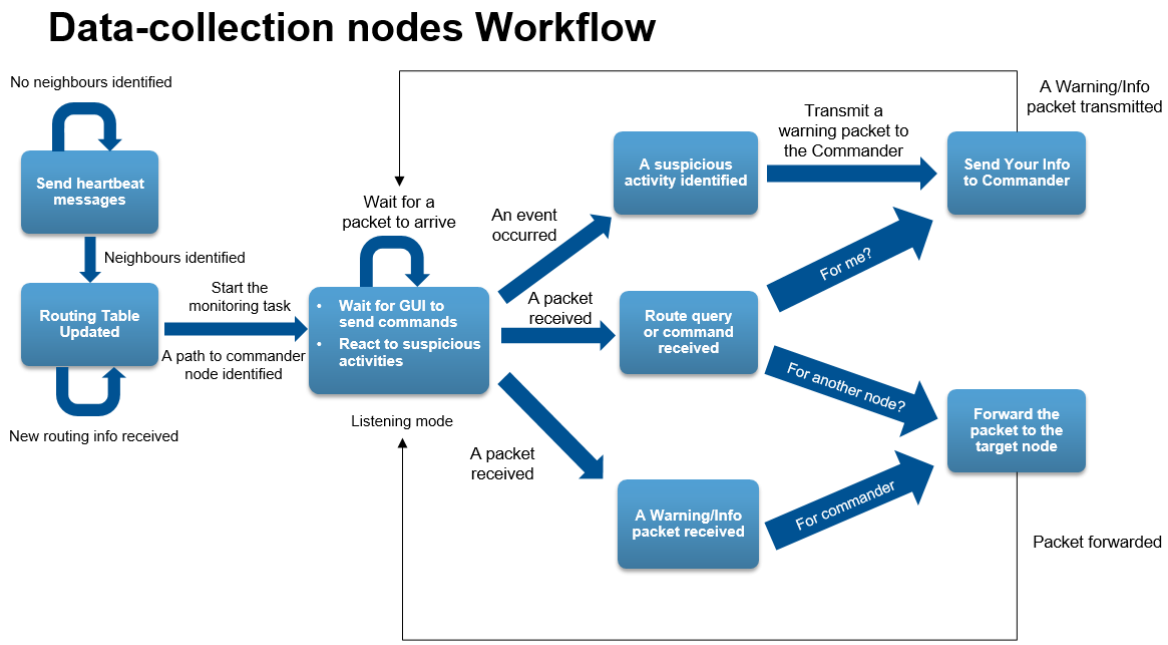

Figure 2.1: Infrared Acceleration Sensor Workflow

2.2.1 Initialization

Every fifty milliseconds, the initialization task broadcasts a "hello" package, consisting of only the one byte long package identifier (255) and the unique identifier of the source node. The nodes in the live demonstration were numbered from one to ten, but the code itself is scalable and not node-specific. The dynamic self-organization allows for automatized distinguishing of purposes and positions of the nodes in the network. In order to differentiate between different nodes, only the MAC byte identifier macro needs to be adjusted, using the preprocessor define directive. Any node in vincinity, also running the initialization task, will accept the initialization package

and sent one last "hello" package before going into sleep mode. In sleep mode, no initialization packages are sent out anymore, except on the occassions where a "hello" packet from a newly added unregistered node is received, after which one targeted initialization packet to this node will be transmitted. This is done in order to ensure that all sides are informed of each other's participation in the network. The received data about the neighbors in vincinity is saved into a local routing table. This allows for targeted package transmitting to neighbors and defined broadcasting to multiple nodes in later phases of communication.

2.2.2 Routing and package transmission

After the initial deployment phase, in which the nodes were assigned fixed positions and have finished determining their neighbors, the commander node, with the MAC byte identifier 1, can be added into the network and register itself against the slave nodes within reach of its signal. Similiar to above, after turning on, the commander node broadcasts one initialisation packet on the medium, gets recognized by the slave nodes in vincinity and receives responses from these nodes. This way, the commander knows who his direct neighbors are, and is able to demand for routing information from the nodes which are more than one hop away in the network. Since the laboratory room, in which the project was planned and executed, is of a rather small size, different transmit power flags were used when transmitting packets, in order to simulate the varying distances between the nodes participating in the network. The remaining nodes then switch to the listening mode (main loop), in which they wait for both incoming commands sent out from commander and external spikes (IR sensor readings) to prepare and send out the corresponding event packets if the commander node is to be notified. Different kinds of commands need to be parsed in the receiving (RX) task, and are distinguishable from each other by examining the first byte in the received package. The structure of these messages varies with different kinds of packages:

- Initialization packet: Byte identifier 255 and a payload consisting only of the source node's MAC ID.
- Command packet: Byte identifier 254, and a payload consisting of source node's and sink node's MAC identifiers and one command identifier (ranges between 1 and 6). Depending on the command identifier received, the sink node executes the action associated with it and sends the corresponding response back to the source node.
- Response (ACK) package: Byte identifier 253 together with a payload consisting of source node's and sink node's MAC identifiers and an answer string to previously sent queries.
- Event (warning) packages: Byte identifier 252, sent out whenever suspicious activities are detected. Payload consists of source node's and sink node's MAC identifiers and data obtained from current sensor measurements.

Depending on the command identifier, the info response (answer to command 1) may be sent back, which contains data about battery voltage state, current temperature, accelerometer readings and infra-red sensor state. Also, information on if the eventing system is down or up and running is sent as the last byte in this package. This byte serves as an acknowledgement for when the event system is shut down manually over remote connection. It can be toggled on and off by sending the commands with command identifiers 2 and 3 to the target node. The command identifier 4 is used to ordering the nodes in vincinity to switch their target when sending packages like initial routing tables to this packet's source node. Command 5 is usually sent only once and is a packet which should arrive at the closest neighbors of commander, requesting from it to send/resend their routing tables, together with the routing tables of their neighbors to the commander node. Similiar to the command 5, command 6 exists, but with individual targets. Using the command identifier 6 only the routing table from the target node can be requested. When actually looking for a pathway to send a message through, the commander looks through its own global routing table and looks for the closest connection over which the packet should be sent. The rest of the routing is done on the nodes in between and involves the same procedure of looking for the next connection in order to get to the target node, but on these nodes, the local routing tables are used.

	Nodes	Neighbors →				
*	1 (Comm)	5	2			
	2	1	5	3		
	3	2	6	5		
	4	5				
	5	3	2	1	4	
	6	3				

(*): The IDsIdentified[i] array

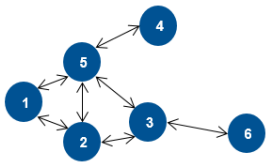

Figure 2.2: A global routing table example

Local routing tables are realized as arrays describing the neighborhood of the nodes which have a connection to this node. Eventually, all routing queries are answered by all nodes, and this data is collected and parsed in the commander's global routing table, but in between, data is stored on the hops over which the routing table data was sent through, as well. Since the target node's MAC identifier is always available as a part of the payload, the forwarding of packages is

rather simple. In case the target node is already available as a direct neighbor, the package is sent directly. Otherwise, the local routing table is searched through for the next connection in the corresponding pathway.

2.2.3 Event-based communication

Periodically, the port pin on which the the pyroelectic infrared sensor is attached, is checked for its current state. In case a movement was sensed, the sensor writes a 1 on its digital output, resulting with a raised warning flag in the node workflow. This makes the Transmit task send out a warning message (with the byte identifier 252) to the commander node. The message is then understood as an event by the commander and asynchronously transmitted to the server over UART and, finally, over a connected socket to the GUI application, where the user gets informed of suspicious activity at the mentioned node. The warning is shown as a pop-up window on the screen, independently of the rest of the application. This eventing system can be disabled using a toggle function directly from GUI, for occassions where the user is assured no events should be classified as warning events (during the daytime, for instance). In addition to the function of reporting the out-of-place IR readings, the nodes can notify the user of low battery power available and high temperatures readings (fire alarm).

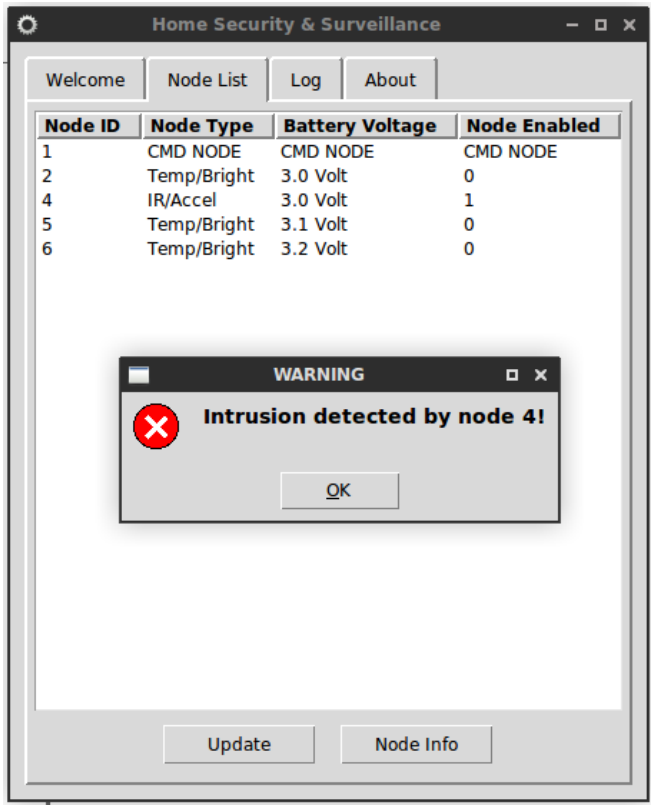

Figure 2.3: Event received and shown on the GUI

2.3 Commander & GUI Workflow

2.3.1 Initialisation

After the commander node enters the network, the initialisation packets are broadcast to the closest nodes, and the corresponding responses received. By sending a command package (254) with the command identifier 5, the remaining nodes in the network are requested to share their routing tables with the commander. For each of the nodes, the commander dedicates a row in the routing table, containing their direct neighbors. Whenever a targeted transmission needs to be executed, the commander will search the routing table through for the node in question. The two-step search algorithm is then executed:

1. If the target node is found in the row containing the direct neighbors, the packet will be sent directly to this neighbor.

2. If not, the whole routing table is searched through and the target node found in a row containing the neighbors of a node over which the packet could be forwarded. The "forwarder" node is then taken as the new input value of the search algorithm and the two-step process repeated until the corresponding next hop is returned as the value.

After the correct next hop or a direct connection gets defined, the commander sends out the packet containing the end node identifier byte. Later on, for forwarding purposes, the nodes in the pathway use this flag and the two-step search process in order to determine their next connection.

2.3.2 Data stream to the GUI over a dedicated interface

In addition to managing routing tables, sending out commands and receiving responses and events, the commander needs to bidirectly communicate with the server interface as well. The byte-wise transfer is accomplished over the UART interface in a dedicated task. The incoming data on UART is parsed and the command identifier and sink node determined. The server interface is a multi-threaded Python application, making use of PySerial library in order to write and read from the USB port the commander is attached on. The server application was implemented for and demonstrated on a Raspberry Pi device, a credit-card-sized single-board computer running Debian Linux. The application serves as a package parser and formatting tool, since it also prepares the strings before sending them out to the GUI application. Using the Socket library, a TCP socket for the communication with the outside world is created, an IP address and listening port bound to it and the interface application put in a waiting state. For the purpose of the project, a GUI application had to be developed as well, and currently allows for a connection to the above mentioned server interface over the TCP/IP socket, sending of commands, asynchronous receiving,

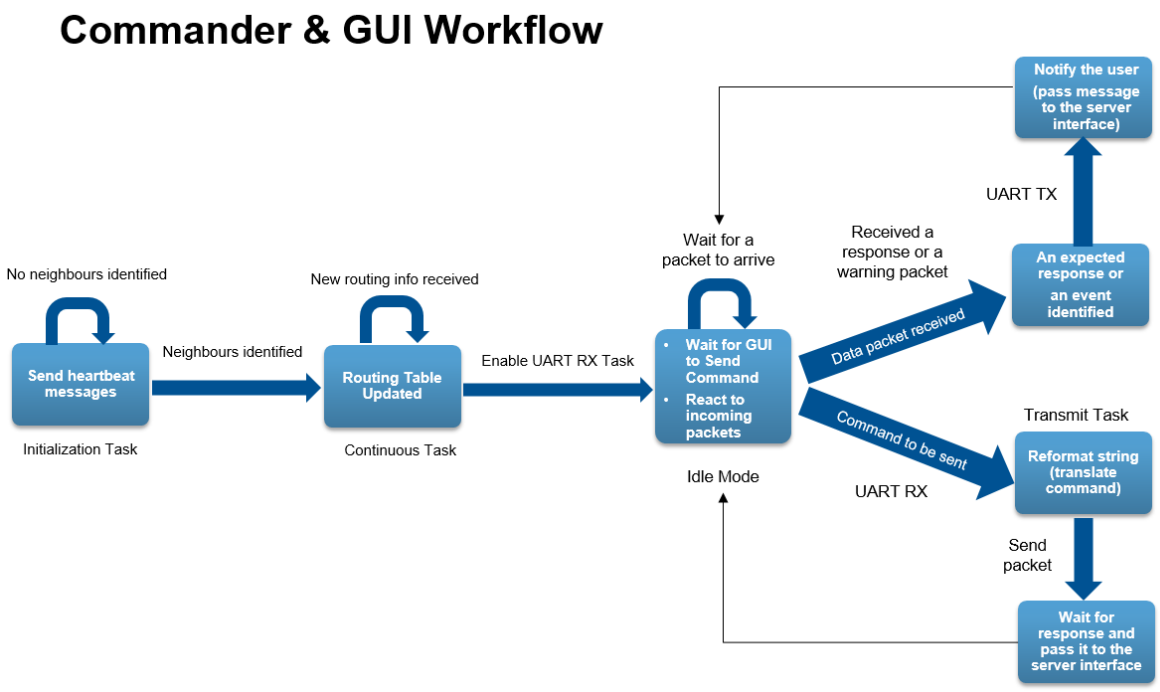

Figure 2.4: Commander & GUI Workflow

parsing and display of responses. It is able to react correspondingly to incoming event messages, via pop-up windows. In addition to being able to request node information and measurement data individually, the GUI also allows for enabling and disabling of eventing system on each of the nodes. Another feature is a possibility of manual refreshes of the routing tables, which allows for network reorganization if the commander is to be moved around inside the network. In the GUI, the events are shown as pop-up windows, warning the user of suspicious activities. The measurement data for each node presented in a clean multilist interface, and is dynamically updated depending on the return values of the requests for current network readings (info command). The GUI also supports simple logging, disconnecting and uses a Tcl/Tk framework (TkInter) as the base window library.

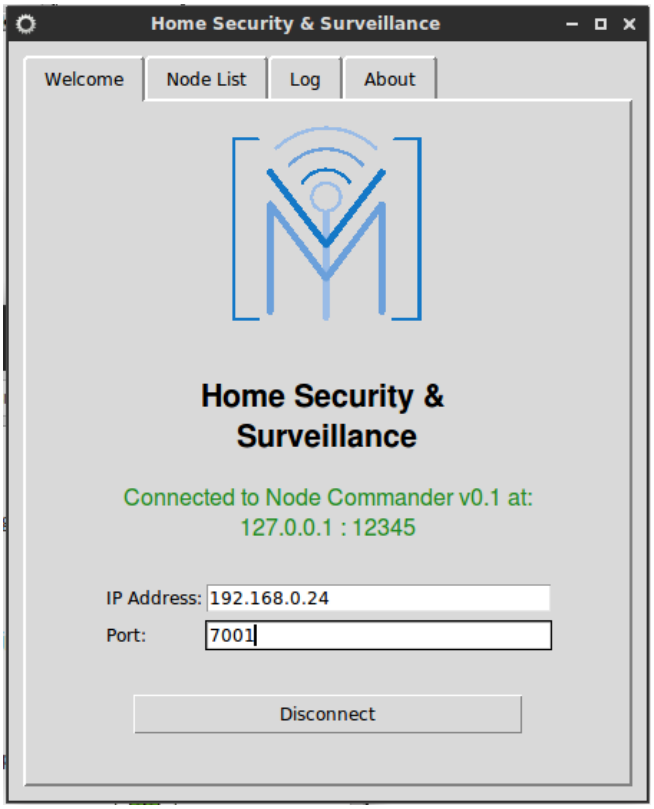

Figure 2.5: Connecting to the server interface

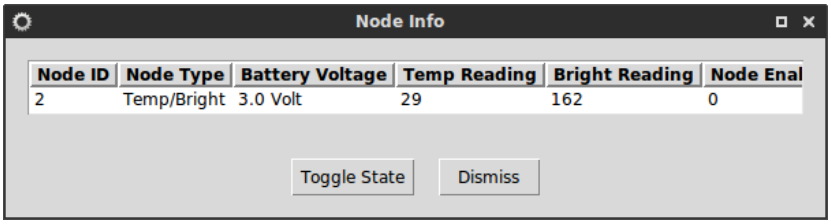

Figure 2.6: Node info window

3 Conclusion & Prospect

During the project, we had a chance to implement and work with a fully-scaled network of wireless nodes and develop a practical application for it. A distributed system was written, taking bits of different kinds of networks and putting them together in order to achieve functionality of fully-fledged wireless networks without having an actual TCP/IP stack and with a focus on low-overhead, resource-constrained communication. The network communicated with a generic infrastructure, making use of self-developed command & response based system, combining event packets for asynchronous communication and is adaptable to future wireless sensor projects. Also included was a user interface to monitor the status of the entire system, send commands, receive responses and intelligently notify user of suspicious activities. Still, we want to mention a few details that might have been implemented, had a bigger timeframe been given to us. Although we can route messages using different pathways (two-step algorithm for the search of multiple next neighbors), we didn't actually take statistical measurements or had any constraints about which concrete path should be favored at any time during the transmission. Multiple possibilities exist here. For instance, favoring the path with the highest overall battery voltage statistics or going for the shortest possible path are only two of these. For both of these, the generic architecture was coded, has been shown functional and could be appropriately adapted. Another point that was missed out, is the audio motion sensing which would involve some deeper signal processing knowledge and statistics about the frequencies used in case a window was broken, for example. Nevertheless, the goal of building the wireless network capable of monitoring and offering an additional mean of home security was developed, and is already fully functional and usable.

A List of Figures

B Bibliography