Embedded Wireless Networks with 6LoWPAN

A study of Texas Instruments' low-power wireless kit using 6LoWPAN

SOFIE SJÖDAHL

KTH Information and
Communication Technology

Degree project in
Electronics and Computer Systems
First level, 15.0 HEC
Stockholm, Sweden

Abstract

This thesis is a study of the wireless embedded Internet implemented with a standardized set of protocols called 6LoWPAN (IPv6 over Low-power Wireless Personal Area Networks). 6LoWPAN enables efficient and adaptable use of IP in networks with low power and low bandwidth.

Such a network is set up consisting of an Edge Router and two nodes called CC1180DB from Texas Instruments. A program called NanoHost Example was downloaded and run on the boards making them send data packages with amongst else the current RSSI (Received Signal Strength Indication) value.

With a PC connected to the network, a program called NodeView was used to receive the data. NodeView is a product from Sensinode based on tabs that gives the user possibility to add and program a new tab.

A new tab was created in NodeView, showing the received RSSI data in a graph as function of time. The program worked well and the data was reasonable compared to two indicator LED's on the board.

The conclusion is that 6LoWPAN and the associated software and libraries provide a relatively easy way to build long-lasting low power wireless networks. The technology has an endless number of possible applications for industry and individuals to use.

Contents

1 Introduction **1**
1.1 Objective 1

2 Introduction to 6LoWPAN **3**
2.1 Routing and bootstrapping 3
2.2 The 6LoWPAN protocol stack 4
2.2.1 Transport layer 4
2.2.2 Network Layer 5
2.2.3 Data link layer 6
2.3 The Development kit 6

3 Methodology **9**
3.1 Setting up the network 9
3.2 The MSP430 application 9
3.3 Programming of the Java application 10

4 Results **15**

5 Conclusions and Discussion **17**
5.1 Applications of 6LoWPAN 18
5.2 Conclusion 18

Acronyms and Abbreviations **19**

Bibliography **21**

Chapter 1

Introduction

One of the important areas of technical development today is the phenomenon called "The Internet of things". It refers to all the small low-powered wireless networks. The nodes are called Smart Objects and they usually consist of a small MCU (Microcontroller), a network processor and some kind of sensor [1].

6LoWPAN is a set of standardized protocols defined by IETF (Internet Engineering Task Force) for enabling communication with Smart Objects. 6LoWPAN is built on IPv6 (Internet Protocol version 6) which is the newest IP and the most used networking protocol [2].

Texas Instruments (TI) has constructed a Development kit called "Sub-1GHz 6LoWPAN Development kit", from here on referred to as the Development kit. It contains five boards; four nodes of two different types and an Edge Router. The Edge Router will be connected to a PC with an Ethernet cable and the nodes will be connected wirelessly to the Edge Router. Together they form a small network [1].

In this thesis, only one of the types of boards is used. It is the CC1180DB (Development Board) which contains the network processor CC1180 and the MCU MSP4305438A [1], from here on referred to as the MSP430. There is a little sensor in the MSP430 that measures RSSI, a measurement of the radio signal strength in dBm (deci Bel per milli Watt) [3].

NodeView from Sensinode is a Java-based program for PC. NodeView is used for communicating and controlling the Edge Router and the nodes [1].

1.1 Objective

The objective with this thesis is first to use the programming environment IAR Embedded Workbench to run an application on MSP430. The application does, amongst other things, put the RSSI in the data payload and sets the transmitting interval to 40 seconds.

Then a new tab in NodeView will be programmed, where the RSSI data is received and presented in a graph.

Chapter 2

Introduction to 6LoWPAN

The source of the facts in this chapter is Shelby's 6LoWPAN [2], if nothing else is specified.

The Internet is sometimes divided into three areas; the core Internet, the fringe Internet and the wireless embedded Internet. The core Internet consists of servers and routers. The fringe Internet is what the user centric units belong to, for example PCs, cell phones and game consoles. The wireless embedded Internet is not very large-scaled yet, it refers to Smart Objects and small computers embedded in other devices [4].

The wireless embedded Internet is built up by many small stub networks (networks that lie on the edge of the Internet). A LoWPAN (Low-power Wireless Personal Area Network) is such a stub network, where the Edge Router and all the nodes in the LoWPAN share the same IPv6 address prefix.

The Edge Router of a LoWPAN can be connected the Internet and takes care of the routing of traffic to and from the nodes. The Edge Routers also handle the 6LoWPAN compression and Neighbor Discovery (both described later in this section). If the Edge Router is connected to an IPv4 network, it also takes care of Ipv4-Ipv6 interconnection. That is solved by a technique called tunneling.

A LoWPAN can also exist without connection to the Internet, a so called Ad hoc LoWPAN. This is the type used in this project.

2.1 Routing and bootstrapping

IPv6 has no routing algorithms itself, so any routing algorithm can be used for the LoWPAN. RPL (Routing Protocol for Low-power and lossy networks) is a routing algorithm developed for low power wireless networks by the IETF working group ROLL (Routing over Low power and Lossy networks) [5]. RPL has a proactive distance-vector approach, which means the routes are built up before they need to be used. And since all nodes in a LoWPAN can act as routers, they have information about their paths and where to send data [4]. Figure 2.1 from Sensinode shows how routing can look like in a LoWPAN connected to the Internet.

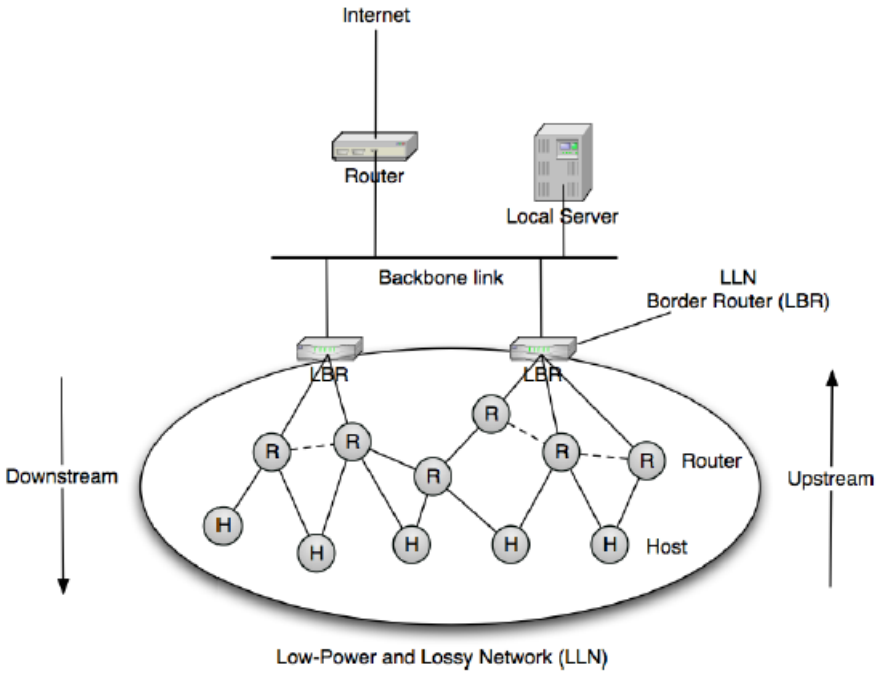

Figure 2.1. A routing example with RPL.

Bootstrapping is the process when a device, without human intervention, connects to a network and is ready to communicate. The 6LoWPAN bootstrapping is managed by the earlier mentioned 6LoWPAN Neighbor Discovery. It is a protocol that defines how the nodes in the LoWPAN communicate, how paths to routers and between the devices are built.

2.2 The 6LoWPAN protocol stack

The 6LoWPAN technique is built on the IP protocol since IP is very suitable. One advantage is that IP is world wide spread and computer engineers are already familiar with it. It means that less introduction is needed and the technology can be established faster than with a different set of protocols [6].

Figure 2.2 from Sensinode shows the 6LoWPAN Protocol stack, which is very similar to the IP stack. But between the network layer and the data link layer, an adaption layer is placed. The adaption layer is embedded in the network layer in Figure 2.2. This section will briefly describe the layers of the 6LoWPAN stack.

2.2.1 Transport layer

The transport layer protocol most preferably used for 6LoWPAN is UDP (User Datagram Protocol). It is lightweight (the full specification fits in two pages) and

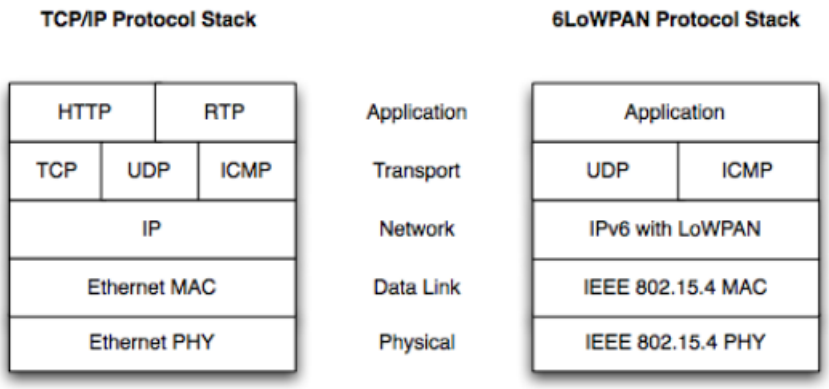

Figure 2.2. Comparison between the IP stack and the 6LoWPAN stack.

connectionless, which means that the two communicating processes do not have a standing connection established. The UDP packages are called datagrams, and since there is no established connection between the processes, the address to the destination process must be attached to every datagram. UDP leaves no guarantee that they will reach their destination, or that they will reach it in the same order as they were sent. This is the price for the lightweight; a quality that usually has a higher priority for 6LoWPAN applications then reliability [7].

The UDP header consists of 8 bytes and contains four fields of 16 bits each. For comparison, another common transport protocol from the IP stack called TCP (Transmission Control Protocol) has a 20 bytes long header. Therefore is not as suitable for 6LoWPAN as UDP is [7].

2.2.2 Network Layer

The Network layer in the 6LoWPAN stack consists of the IPv6 Protocol. IPv6 was developed decades ago when the addresses of IPv4 (Internet protocol version 4) were predicted to run out. IPv4 has 32 bit-addresses which suffices to only $4 * 10^9$ unique addresses. With the growing amount of Internet-connectable devices, it was not enough. Therefore, IPv6 was created with 128 bit addresses. That results in approximately 10^{38} addresses, which is "enough for every grain of sand on the planet" [4].

The adaption layer is what makes IPv6 manageable for 6LoWPAN. It compresses the IPv6 and UDP headers between the Edge Router and the node. This is possible since the Ipv6 prefix is no new information, since the IPv6 prefix address is shared between the Edge Router and all the nodes in a LoWPAN network. The Edge Router and the nodes only need to know each other's MAC addresses (Media Access Control address) to address each other.

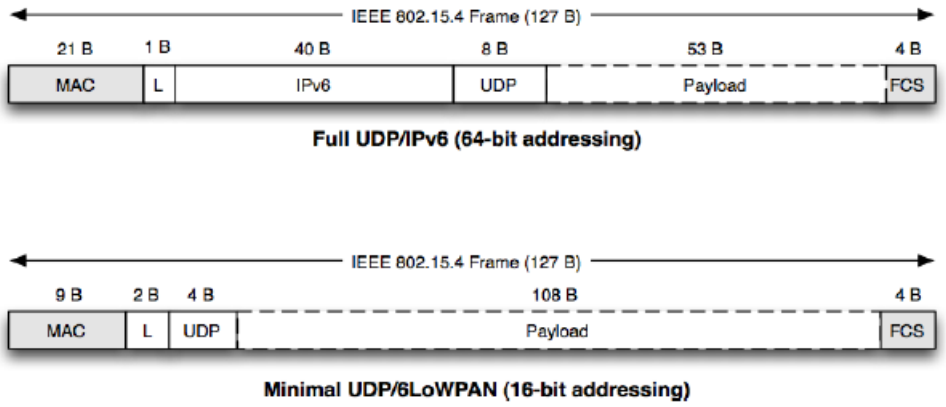

Figure 2.3. Before and after the compression of the IPv6 address.

The compressing enables a bigger payload (the part of the address that consists of the data to be transmitted). Figure 2.3 from Sensinode shows a simple model over the compression. The best case scenario is to get the IPv6 and UDP headers down to 6 bytes (the "L" and "UDP" fields in Figure 2.3), which leaves 108 bytes to payload [4].

2.2.3 Data link layer

The data link layer consists of the IEEE (Institute of Electrical and Electronics Engineers) 802.15.4, which is a link of low-power technology. 802.15.4 is used in many embedded applications, due to its small frame size and low bandwidth. It was designed to suit long lived applications on low cost nodes, such as Smart Objects [5].

802.15.4 has a maximum packet size of 127 bytes. With the 40 byte IPv6 header, this leaves less than 70 % of the packet for payload. This is the reason header compression is needed, so the IPv6 header only covers about 5 % of the packet and more can be used for payload [1].

The link layer can be replaced by almost any other link radio frequency protocol. That is thanks to the great compatibility capacity of 6LoWPAN [7].

2.3 The Development kit

The source for this section is the Development kit manual, [1].

Figure 2.4 from the Development kit manual shows a theoretical architectural overview of the Development kit. As mentioned earlier, the nodes used in this thesis are CC1180DB boards that contain a CC1180 network processor with 6LoWPAN and a MCU called MSP430. The API (Application Programming Interface)

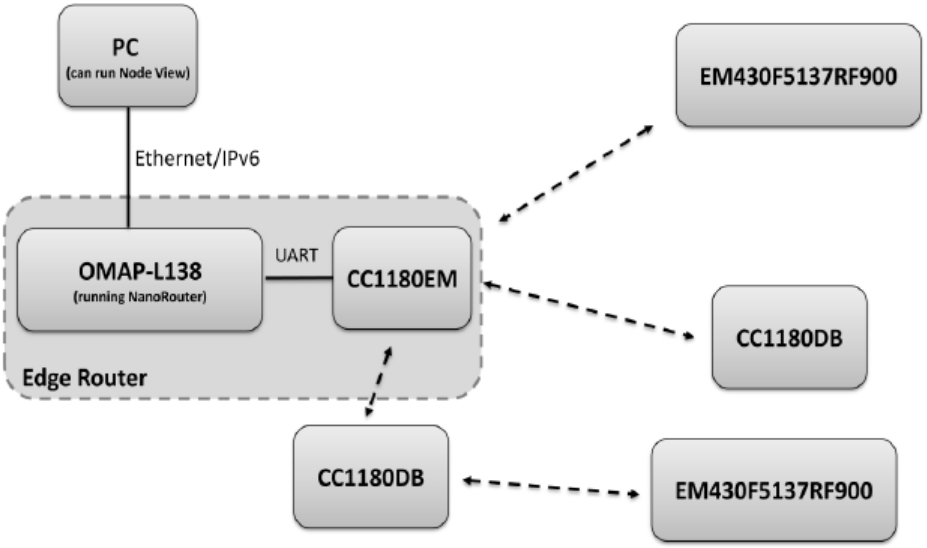

Figure 2.4. An architectural overview of the development kit.

NAPSocket library from Sensinode is the software interface for communication between the MSP430 and the CC1180. The hardware interface is UART (Universal Asynchronous Receiver/Transmitter).

The connection between the Edge Router and the PC is an Ethernet cable and the interface between the Edge Router and CC1180EM is UART. All the nodes can act as either routers or normal nodes, depending on their geographical positioning relative to each other and to the Edge Router.

The Edge Router is Linux based and contains an OMAP-L138 processor, which runs the software Sensinode Ltd NanoRouter 2.0. The Edge Router uses a board called CC1180EM (Evaluation Module) for the radio communication. An adapter board is used as communication between the networking board and the Edge Router.

The user communicates and controls the Edge Router with NodeView. NodeView has three original tabs with technical content. The "Router View" tab shows information about the connected unit. There is a list of the nodes with information about for example last connection time stamp, the number of delivered packages and the MAC-address. The "Message log" tab contains the received payloads in hexadecimal code.

IAR Embedded workbench is a free programming environment which will be used to program the boards. NanoHost Example is an application project for MSP430 creatd by Sensinode. The NanoHost is run and debugged in IAR with a debugger device called MSP-FET430UIF from Texas Instruments.

Chapter 3

Methodology

This chapter describes the process of the project. The first section tells about setting up the LoWPAN network and the second section tells about the MSP430 application. The third section describes the programming of the tab in NodeView.

3.1 Setting up the network

The connection between the PC and the Edge Router was set up by adding the Edge Router's IPv6 address with the Netsh command "2001::22" in the command prompt. The Netsh command enables displaying or modifying the network configuration of a computer. The two colons in the IPv6 address is a shorter way to write zeroes.

The Edge Router was set up by connecting the power and Ethernet cables. The adapter board and the CC1180EM board were attached to the Edge Router. Sensinode NodeView from IPSO Alliance (IP for Smart Objects Alliance) was downloaded and the Edge Router was added so the network could be viewed and analyzed.

After that, the nodes were powered on and placed with one meter's distance from each other, to avoid interference. When a node is powered on it automatically searches for a network and connects to it, with Neighbor Discovery. When it joins the network, a Network Analyzer application is launched on the node, which sends data about the node to NodeView.

The "Network Analyzer" tab contains an animated topology overview of the network. Figure 3.1 is a print screen example of the network topology in this project. A green dot indicates a normal node, and a blue indicates a routing node.

3.2 The MSP430 application

IAR Embedded Workbench Kick start Edition for MSP430, NanoHost example 1.0 and NAPSocket Library were downloaded.

The NanoHost Example is interrupt based, which means after hardware initializing, it mostly does nothing, and waits for an interrupt. The program also has a watchdog, which is a timer ticking as the program is running. It gets reset periodi-

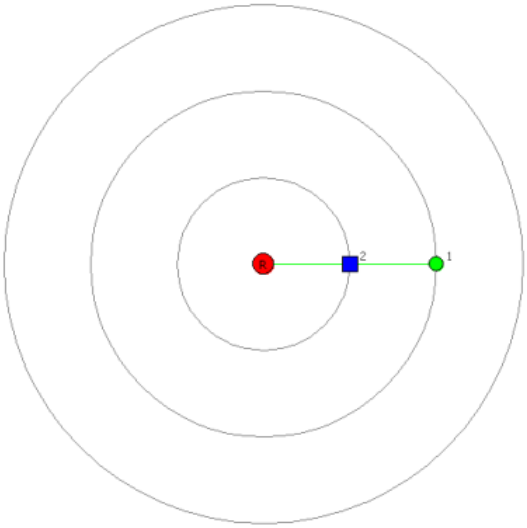

Figure 3.1. Example of the network topology in NodeView.

cally, which indicates that everything is as it should be. If it is not reset until the timer has ticked down, it means the program is stuck in an infinity loop, and then the watchdog interrupts the loop and hopefully get the program working again [3].

The NanoHost Example consists of four main code files; NetworkAnalyzer.c, pheripherals.c, uart_driver.c and timer.c. NetworkAnalyzer.c contains the Main function, which initializes hardware and then waits for an interrupt. The file also contains the event handler, with a few different possible events, mostly hardware interrupts.

The data sent from the nodes is initialized as a 31 bytes long array in Network-Analyzer.c. Figure 3.2 shows the structure of the byte array with the RSSI value lying in index 13 [3]. There is a pointer to the array that can be used to put other values in the array.

The Debugger device MSP-FET430UIF was connected to the CC1180DB and with a USB cable to the computer. The NanoHost Example project was run in debugging mode and was downloaded to the CC1180DB boards. The boards were powered up again and got connected to the network.

3.3 Programming of the Java application

The Java application for CC1180DB will, as mentioned before, be integrated Node-View as a new tab waiting for RSSI data from MSP430 and drawing graph of it. This section describes the thought behind the code of the tab.

clip=true

A new Java project and a class called GraphTab were created. A DAT-file called

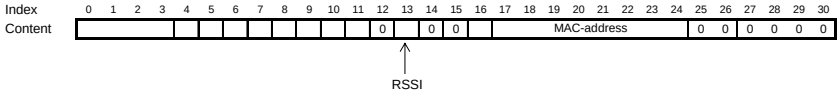

Figure 3.2. The structure of the data sent from MSP430.

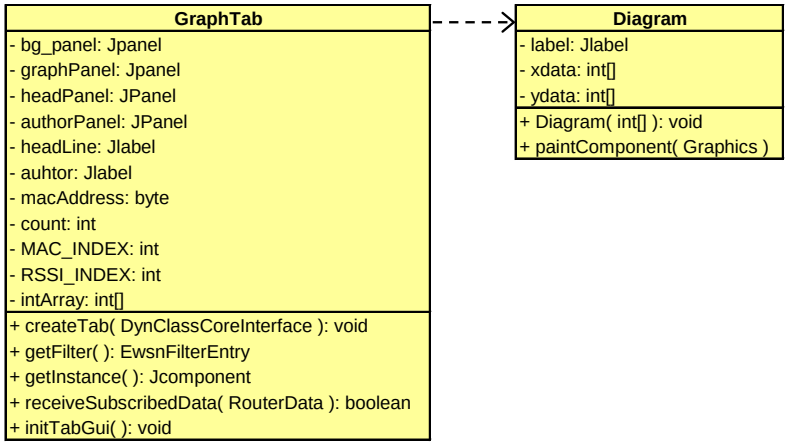

Figure 3.3. A UML diagram showing the structure of GraphTab.

license and a JAR file called NodeViewProDyn.jar (both included in the NodeView package from Sensinode) and were included in the Java project. NodeViewProDyn.jar is constructed by TI and contains the library with functions based on UDP. These are the ones used to communicate with the nodes and the Edge Router. It is necessary to list the tab's name in tabConfig.txt for it tab to appear in NodeView [1].

Figure 3.3 shows the UML (Unified Modeling Language) diagram of GraphTab where the Classes with their global variables and methods are listed. GraphTab extends **AbstractTab**, like all the tabs in NodeView do. All of the methods in the UML diagram, except **initTabGui**, are inherited from **AbstractTab** and overridden by GraphTab. These are some of the methods included in the NodeView library.

When NodeView is started, the method **getInstance** is called by the application. **getInstance** calls initTabGui which builds the GUI (Graphical User Interface) of GraphTab. The text cells with the headline and the footer are put in the GUI, and a panel is prepared to later contain the graph.

Every time data is received from one of the nodes, the method **receiveSub-**

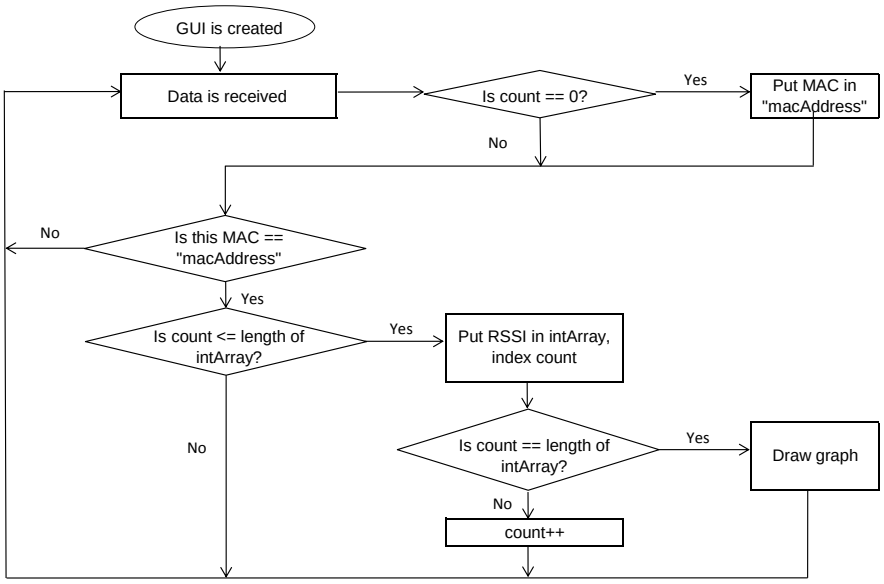

Figure 3.4. A flow chart over the code that runs every time that data is received.

scribedData is executed. It returns a Boolean; true if the data was read successfully, and false if not. This is where all the data handling needs to be done.

Figure 3.4 shows a flow chart of the code in **receiveSubscribedData** . It is a simple way to see how it works, but it is also explained in text in the following paragraphs.

When more than one node is connected, we want the values in the graph to be from the same node, otherwise the graph will lack of meaning. Therefore, there is a variable called "count" that lets us keep track of how many values are collected. The first time data is received, count is equal to zero. When count is zero, we check the MAC address of the node that sent the data. That MAC of the first connected node is saved as a variable.

Now we can check the MAC address every time data is received and compare it to the MAC address from the first node that made contact. From there on, the data packages of the other nodes are ignored by GraphTab.

The **RouterData Class** has a method called **Buffer**, which is used to get the payload from the node in form of a byte array. As seen in Figure 3.2, the RSSI lies in index 13. The RSSI is picked out of the byte array and put into an empty integer array. Every time data from the same node is received, a new RSSI value is put into the next slot in the integer array. The count variable is also increased by one. The length of the integer array is 10 (no special reason for that number) and the count variable counts how many values are received. This way, values will be

placed in the array until it is full.

When 10 values are received from the same node, an instance of the class **Diagram** is created. The **Diagram** class extends JPanel, to make it easy to integrate it in the GUI. The integer array is used as parameter to the constructor of the **Diagram** object. The constructor sets the variable "ydata" to the array parameter (the integer array). **Diagram** has a method called **paintComponent**, which is a method from the Java API. **paintComponent** draws lines and figures according to a coordinate system. In this program it draws the axes of the graph, and the curve based on ydata. The graph is finally put in graphPanel, and is shown in the GUI.

The rest of the data packages from the node are ignored by GraphTab. Next to GraphTab, NodeView original tabs are shown, and all the settings can also be made in this version.

Chapter 4

Results

When NodeView is started, the user has to add the Edge Router in the tab "Router view". In the tab "Message log", the payload and origin of each message is shown. The new tab GraphTab is so far empty, with only the headline and footer showing.

Say two nodes are powered up and connected. They start to send data packages. When 10 packages from the same node are received, a graph based on its RSSI is drawn in GraphTab. The other nodes' data are ignored.

The unit on the Y axis is dBm. The scale on the X axis is chosen by the user, under "options" in the "Network Analyzer" tab. There the user can set the time interval in seconds, default is 40.

An example of GraphTab is shown in Figure 4.1. The chosen time scale is 10 seconds per marking and the distance to the Edge Router is 1 m. As seen in the graph, the RSSI varies slightly, but is mostly around -18 dBm. Since RSSI is a measurement of signal strength, it depends on the distance between node and Edge Router. Figure 4.2 shows a measurement with 10 seconds interval and 3 m distance. As expected, the RSSI value decreases with increasing distance to the Edge Router.

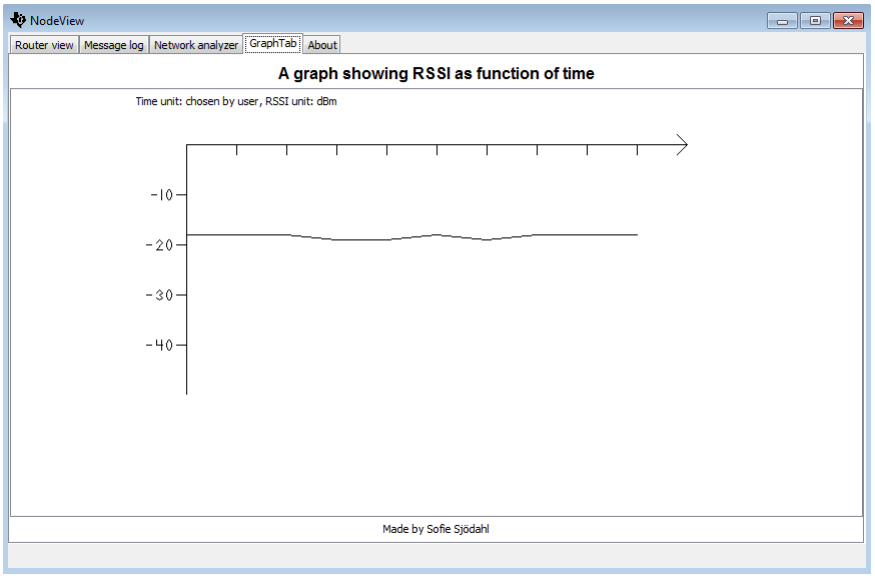

Figure 4.1. The GraphTab in NodeView, distance: 1 m.

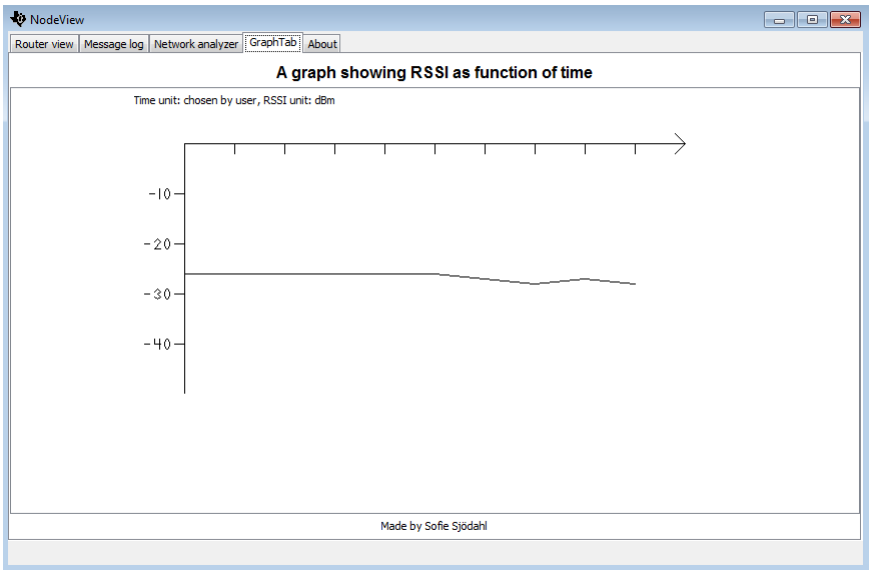

Figure 4.2. The GraphTab in NodeView, distance: 3 m.

Chapter 5

Conclusions and Discussion

The result of the project was as expected. The boards weren't always cooperative, but when they were, the data was delivered without problems. The measurements of RSSI were also reasonable, according to the manual of the Development kit [1]. In addition to getting sent by the data, the RSSI is also roughly indicated by a green and a red LED on the CC1180 board. The LEDs' behaviour was compared to the curve in GraphTab, and the results matched.

One of the difficulties with this project was handling all the documentation of the technique. But I learned to find the relevant information. Also understanding the program code of NanoHost Example was a challenge, since I am not a very experienced programmer. The program code of NodeView was a smaller problem, but it was a bit tricky.

However, the main difficulty was learning network communication on my own, with only books to help. In the meantime I have learned much about wireless network processes and hardware programming. It was also good to get an idea of how it is to work in this line of business.

Although I have reached my goal, this project is not a closed chapter; it could be much more developed. The GraphTab could be programmed to ask how many values the user wants in each graph and scale it to fit. The graphs could write over each other so that the newest values are always displayed. Or the network could be connected to the Internet and show the result on a website.

One could also draw and construct a node from scratch, by using the CC1180 network processor and a free choice of MCU. There are good pattern and schematic charts of all the boards in the manual of the Development kit.

The whole 6LoWPAN technology is very complex and it would take months to fully understand it, I have only scratched the surface. But there is no doubt that the future is bright for 6LoWPAN. The next section gives a few examples of the implementations already made and ones to come.

5.1 Applications of 6LoWPAN

6LoWPAN was created with the purpose that users will be able to modify and embed it into all kinds of devices. It is free to use and made to being easy to understand and implement for both private citizens and corporations [4].

With wireless embedded devices, many energy saving solutions are possible. One application is called "smart grid", which enables overviewing the power grid with Smart Objects. It is a smooth way to preview the energy use and adjust the electricity flow [7].

Another application area is building automation, which can be implemented by letting sensors report status of units and thus monitor the use and enable efficient improvements. Sensors can be used to track employees and vehicles. Energy reducing can be achieved by monitoring and controlling the use of light, heat and air condition [2].

Below follows a list over a few other applications that 6LoWPAN makes possible [2].

- Smart metering
- Industrial automation
- Environmental monitoring
  - E.G. measure levels of oxygen in the ground over large areas
- Transportation
  - E.G. keeping track of vehicles
- Personal fitness

5.2 Conclusion

Over all, this has been a very educating thesis. It has been quite exciting to be working with such new technology as the 6LoWPAN. The conclusion is that 6LoWPAN and the completing software provide a relatively easy way for both corporations and individuals to build long-lasting low power wireless embedded networks. And there are endless numbers of possible applications for these networks.

Personally I approve very much of the energy saving possibilities. The more we know about energy use, the more we can adjust the settings and products.

Acronyms and Abbreviations

6LoWPAN	IPv6 over Low-power Wireless Personal Area Networks; the set of standards for LoWPAN's
API	Application Programming Interface
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force; a group that creates and maintains the Internet standards
IPSO Alliance	IP for Smart Objects Alliance
IPv6	Internet Protocol version 6
LoWPAN	Low-power Wireless Personal Area Network; a physical network of low-power nodes
MAC address	Media Access Control address; the physical address of an item in a network
MCU	Microcontroller; a small computer
ROLL	Routing over Low power and Lossy networks; a group within IETF
RPL	Routing Protocol for Low-power and lossy networks
RSSI	Received Signal Strength Indication
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UML	Unified Modeling Language

Bibliography

[1] Texas Instruments, *Sub-1GHz 6LoWPAN Development kit - User's guide*, 2011 Literature number: SWRU298

[2] Zach Shelby, Carsten Borman, *6LoWPAN - The Wireless Embedded Internet*, 2009, West Sussex (UK), John Wiley & Sons Ltd, ISBN 9780470747995

[3] *NanoHost Example*, 2011, Sensinode Ltd, Sensinode Prorietary & Confidential, Version 1.0-02

[4] Zach Shelby, Seminar on 6LoWPAN, 2009

[5] *6LoWPAN: Incorporating IEEE 802.15.4 into the IP architecture*, 2009, Jonathan Hui, David Culler, Samita Chakrabarti, IPSO Alliance

[6] Adam Dunkels, JP Vasseur, *IP for Smart Objects*, 2010, IPSO Alliance, Version 1.1.

[7] JP Vasseur, Adam Dunkels, *Interconnecting Smart Objects with IP*, 2010, Burlington, USA, Morgan Kaufmann Publishers/Elsevier, ISBN 978-0-12-375165-2

TRITA-ICT-EX-2012:113

www.kth.se