UNIVERSITY OF HERTFORDSHIRE
Faculty of Engineering and Information Sciences

MCOM0177 Computer Science MSc Project (Online)

Final Report
January 2013

Design and Evaluation of a Home Automation Monitoring System based on Smart Network Nodes, Open-Source Software and Standardised Protocols

J.W. de Hoog
09213408
1. List of Contents

1. List of Contents .......................................................... 1
2. Abstract ............................................................................. 3
3. Introduction ....................................................................... 5
  3.1 Vision statement ....................................................... 5
  3.2 Research questions ................................................... 5
  3.3 Objectives .................................................................... 6
  3.4 Project background .................................................... 6
  3.5 Forward references to parts of this document ................ 7
4. Smart Network Nodes ....................................................... 8
  4.1 Types of smart nodes ................................................. 8
  4.2 Constraints in using smart nodes ............................... 9
5. Comparison of Communication Protocols ......................... 10
  5.1 Overview of smart network protocols ....................... 10
  5.2 Z-Wave .................................................................. 10
  5.3 ZigBee .................................................................. 11
  5.4 IPv4 .................................................................. 12
  5.5 IPv6 .................................................................. 14
  5.6 Preferred choice of protocol ................................. 15
6. Design of an Optimal Home Automation System ............... 16
  6.1 Choice of hardware ................................................. 16
  6.2 Choice of firmware ................................................ 16
  6.3 Simulation of a medium-sized network ..................... 18
  6.4 Results of simulations ............................................ 19
  6.5 Physical implementation of a small-size network ........ 26
  6.6 Home automation profile ....................................... 26
  6.7 Using web services ................................................ 26
7. Software Design and Implementation ............................... 31
  7.1 System design ......................................................... 31
  7.2 Database design ..................................................... 32
  7.3 Data collection design ............................................. 33
  7.4 Web-based monitoring design ................................ 38
8. Project Evaluation and Conclusions .................................. 43
9. Future Work ..................................................................... 44
10. Bibliography .................................................................... 45
11. Appendices ...................................................................... 51
    A1. List of software used ............................................. 51
    A2. List of abbreviations used with smart node networks ...... 51
    A3. Project plan and timeline ....................................... 52
    A4. Database design ................................................... 52
    A5.1 Data Collection application requirements specifications . 55
    A5.2 Data collection application process and product quality measures 56
    A5.3 eHome website requirements specifications ............... 57
    A5.4 eHome website process and product quality measures .... 58
    A6. Attached files ....................................................... 59
    A7. Simulator result data listings .................................. 60
    A8. Listings of modified Contiki files .............................. 63
2. Abstract

As a low-end implementation of a smart node network, a Home Automation Network will in many respects be subject to the same design considerations as larger networks such as those used for power-grid management and urban traffic control networks. One prime design consideration, besides the selection of hardware, is the choice of networking and routing protocols that must offer a versatile, scalable and management-free solution.

A further important choice must be the selection of a set of parameters particularly suited for the purpose and size of the smart node network. As a Home Automation network is much smaller and simpler than its above-mentioned counterparts, it is more than likely that a set of networking and routing parameters can be optimised for such a situation. Answers to a number of research questions should lead to such a home automation profile. As a physical network is hard to evaluate for network and routing parameters, a software simulator/emulator will be used to obtain the required data.

The use of web services on the network nodes has been tested and found to be feasible providing hardware with the necessary memory capacity is available. Web services provide a flexible and reliable way to remotely obtain sensor and network data and are easy to program once the underlying system firmware is available.

Finally, artifacts consisting of a Windows application to periodically read out and filter the smart network data, then store the data sets into a normalised database as well as a secure website that obtains the sensor and network data from the database and displays the results in an easily interpretable graphical format together with some key statistical values to be shown by any remote browser are presented. The development of these artifacts took place using Extreme Programming (XP) techniques.
3. Introduction

3.1 Vision statement

The eHome project wants to provide a low-cost but versatile and flexible Home Automation Monitoring System for use in private homes. This Home Automation System will consist of a number of smart network nodes incorporating several environmental sensors such as light, temperature and humidity sensors. These network nodes will be wireless and therefore cable-free and easy to install and maintain. The system will be easily usable by non-technical clients.

An easy-to-use secure website will be built to read node and sensor data from the internal database and display these in an easily interpreted graphical fashion. The website can be accessed through the internet by any suitable browser but is optimized for FireFox.

The eHome project will act as a test bench for optimisation of the communication protocols used, in particular the routing protocol, and might result in a Home Automation profile that can be used for similar installations.

3.2 Research questions

- Is it possible to design a wireless Home Automation System based on smart network nodes and standardised network protocols that can be made sufficiently reliable to exchange messages between nodes even under adverse conditions such as radio interference and unreliable low-power links?

- Can such a system be made to automatically set up, maintain, and repair a routing topology?

- Can such a system be easily connected to the internet and thus be controlled and monitored from anywhere?

- Can system parameters be set up such that messaging and routing are optimised for a typical Home Automation System rather than just defaulted for a smart node network of any size and complexity?

- Can parameters for a small network of wireless nodes functioning on battery power be optimised so that the nodes function reliably and with a long lifetime?

- Can these optimal parameter settings be pulled together and result in a profile for general Home Automation System use?
3.3 Objectives

- Compare legacy proprietary protocols against modern standardised protocols for smart object networks (core).
- List the most important design considerations in terms of a Home Automation System (core).
- Design such a system (in demonstration size) using COTS hardware and open-source firmware (ACM SenSys, 2007) (core).
- Using a smart object network simulator, demonstrate that a system with a reasonable number of nodes (25-40) can be used to collect sensor values and network parameters (IARIA SensorComme, 2011) (core).
- Build the artifacts as detailed above and set up a physical network with a limited number of nodes (core).
- Show that the data collection process functions properly, even when some of the nodes are out of direct wireless reach of the collecting node resulting in multi-hop routing (core).
- Change the node firmware in such a way that each node will contain a web service rather than just a simple set of data collection routines (ACM BuildSys, 2009) (advanced).
- Replace the current data collection through a serial port with a process that uses an Ethernet port, or SLIP (RFC-1055) and in this way connects more directly to the internet (advanced).

3.4 Project background

During the last decade, the internet has grown beyond being a network of connected routers, servers and PCs and has increasingly included networked sensing devices such as those being used for smart metering, control of the electric power grid or urban traffic control as well as sensors associated with industrial and building automation (Vasseur, Dunkels, 2010 pp. 325, 361). For these devices to be useful, they must be able to communicate, wirelessly or otherwise, so that sensor values can be read, combined or aggregated and eventually result in complex actions to, say, route power through the electric grid in an optimal fashion (Vasseur, Dunkels, 2010 p. 305).

This “Internet of Things” (Shelby, Bormann, 2009 p. 4) (Kortuem et al, 2010) is expected to grow far beyond the number of nodes presently consisting of PCs, laptops, or smart phones. Low cost intelligent devices used for urban or power grid control could easily number in the tens or hundreds of millions. Such devices will, by necessity, be very limited in power, wireless range and other resources such as processor speed and the amount of available memory (RFC-4919). Especially the limited radio range will have a large impact on reliability in communication and routing of such a network. These devices are
therefore often described as forming Low-power and Lossy Networks (LLNs) (Brandt et al, 2010).

At the low end of the spectrum of such a smart object network, in terms of number of nodes and network complexity, a Home Automation System will integrate many aspects of sensing and remote control: energy monitoring and control such as for heating and electrical usage, light control, security and access control, health control, air quality, etc. (Brandt et al, 2010) (IEEE ISISE, 2008). Embedded intelligence in home commercial and multi-media equipment also has an increasing need to communicate so that actions can be coordinated (ZigBee Alliance, 2009).

During many years, solutions, especially in the home automation field, have been presented in a proprietary fashion but lately, an increasing awareness of their shortcomings, especially in scalability, has led to the use of standardised and open-source protocols resulting in rapid integration with the existing internet (Ko et al 2011a).

3.5 Forward references to parts of this document

The types and constraints of smart network nodes will be discussed in section 4 followed by a survey of suitable communication protocols in section 5. Section 6 will concentrate on the design of a Home Automation System, its choice of hardware and firmware as well as describe a series of simulations conducted to determine an optimal solution. The necessary artifacts for gathering and storing data and display of current and historical data in graphical format are described in section 7. Section 8 will summarise the conclusions that can be drawn from this study and evaluate its achievements. Section 9, finally, will present an outlook at future work.

An appendix section is available to list the software used with this project (A1), a list of abbreviations used with smart node technology (A2), as well as a project plan and timeline (A3). Appendix A4 shows details of the database design. The various parts of appendix A5 list the requirement specifications and process/product quality measures of the software artifacts. Appendix A6 has a list of files provided with this project and A7 shows the detailed simulator data listings. Finally, appendix A8 provides the modified Contiki files.
4. **Smart Network Nodes**

4.1 Types of smart nodes

Smart nodes for a Low-power and Lossy Network (LLN) have several key parameters that allow them to be used in great quantity, possibly at hard-to-reach places, use very little energy and are able to communicate amongst one-another through means of a wireless routable network. Often, key nodes functioning as routers, are directly connected to the mains while the majority of sensor nodes are battery-powered.

Energy scavenging (obtaining energy from the environment) may sometimes be a solution. In particular small solar panels could be used to power outdoor nodes. Wall switches can often glean sufficient energy from the mechanical switching action (EnOcean, 2012).

Smart nodes must be low-cost to allow them to be used in large quantities while keeping the network affordable. As the term “low-cost” is relative, I would like to pose that £50 per sensor node would constitute a sensible maximum for home automation use.

Large node networks do not normally rely on cabling as placement flexibility would be lost and cost may be prohibitive especially when being installed in existing buildings. In general, wireless, battery-powered nodes are therefore preferable. These function by means of a low-power radio that operates with a low duty-cycle and is mostly switched off (this use of a sleep mode will be explained later). Less common are nodes that function by means of Power Line Communication (PLC).

Through their low-power radio, nodes will be able to communicate with their immediate neighbours. Interference with these radio links is common and often packets will be lost (hence the name “Low-power and Lossy Network”). Mostly, radios are used that conform to the IEEE 802.15.4 specifications (IEEE 802.15.4-2006, 2006) (Karapistoli, 2010).

Transceiver hardware can be divided into two main types: those with a sub-GHz radio and those with a radio transceiver operating in the well known 2.4 GHz band which also accommodates WIFI (802.11), portable phones and even microwave ovens.

Although the heavy use of the 2.4 GHz band causes more interference, the use of Direct-Sequence Spread Spectrum (DSSS) modulation allows distinguishing the needed signal from other ones, simultaneously present. An important advantage of a 2.4 GHz radio over the sub-GHz types is the much higher data rate that can be accommodated: 250 kbps against 40 kbps.
4.2 Constraints in using smart nodes

Smart nodes must use extremely little energy so that lithium-ion batteries, which are the preferred source of energy, can last for years without renewal. This allows the nodes to require no maintenance and to be used in places that could be hard to get at.

Computing power and memory size that may be available in abundance in desk-top computers is an extremely constrained commodity in smart nodes. Not only must the physical size of the node be limited but its energy consumption must also be extremely low. This leads to constraints in the size of the firmware that can be accommodated as well as the energy that is consumed by the node in general and the on-board radio in particular, which is often the largest energy consumer (Vasseur, Dunkels, 2010, p.121).

As seen later in this report, some commercially available hardware will be so limited in memory space that the implementation of web services on the nodes becomes a severe problem.
5. **Comparison of Communication Protocols**

5.1 **Overview of smart network protocols**

Over the last decade, several attempts have been made at the design of a versatile, scalable and easy to use set of communication protocols for a wireless Low-Power and Lossy Network (LLN), some quite successful, others less so. Some of these protocol stacks have been written for a specific node operating system; others to function stand-alone or as a library of functions to be called by an embedded application (Dwivedi et al, 2009).

Several protocol stacks are aimed more at an industrial automation and process control environment such as ISA100a (ISA100, 2008) and WirelessHART (Jianping et al, 2008). These protocols target high interference and high security environments and use TDMA time-slicing, channel-hopping and end-to-end AES-128 encryption. They are generally unsuitable for Home Automation Systems that look for low-cost and possibly open-source.

I will present a number of communication protocols that may be more suitable for wireless home automation LLNs and choose one as a preferred protocol stack for further use by this project.

5.2 **Z-Wave**

Z-Wave is an early, simple wireless protocol for home automation. It cannot be used for larger or more professional networks and is limited to 232 nodes. The protocol consists of four layers over the RF-Media layer which is, just like the protocol layers, proprietary. It is not based on IEEE 802.15.4. Recently, the International Telecommunications Union (ITU) included the Z-Wave RF-Media and MAC layers as an option in its new G.9959 standard, which defines a set of guidelines for sub-GHz narrowband wireless devices (ITU, 2012). Table 1 shows the layered architecture.

<table>
<thead>
<tr>
<th>Application Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing Layer</td>
</tr>
<tr>
<td>Transfer Layer</td>
</tr>
<tr>
<td>MAC Layer</td>
</tr>
<tr>
<td>RF-Media</td>
</tr>
</tbody>
</table>

Table 1. Z-Wave protocol stack.

The Z-Wave network consists of several basic kinds of devices: controlling nodes and slave nodes. Controlling nodes are those nodes in a network that initiate control commands to other nodes. Slave nodes are the nodes that reply to and execute the commands. Slave nodes can also forward commands to other nodes, which makes it possible for the controller to communicate with nodes out of direct radio reach (Johansen, 2006).
The Z-Wave radios work at low bit rate (less than 40 kbps) sub-GHz frequencies which vary depending on country regulations. Nodes, arranged in a mesh network, can communicate when in direct radio range of each other. When this is not the case, a node must find another, intermediate node that is within reach of both source and destination nodes.

Being a proprietary protocol with integrated circuits that implement Z-Wave only available from a single manufacturer, Z-Wave does not have the versatility necessary for today’s complex smart object networks.

5.3 ZigBee

The ZigBee Alliance is a non-profit association of members seeking to promote and advance commercial applications in the “smart energy” and home/building automation fields. “Solutions adopting the ZigBee standard can be embedded in consumer electronics, home and building automation, industrial controls, PC peripherals, medical sensor applications, toys and games” (ZigBee Alliance, 2006 p. 1-1). ZigBee is a proprietary specification for wireless communication between smart nodes in a network based on the IEEE 802.15.4 radio link layer.

The ZigBee communication stack is a set of library routines that function without use of an operating system. This stack is built on top of the physical and Medium Access Control (MAC) layers provided by the IEEE 802.15.4 standard. Unlike standardised protocols such as IP, where each layer is independent from the others, the ZigBee stack is interwoven with the IEEE 802.15.4 layers and only functions on top of those. It does not provide any alternative for these radio layers. Table 2 shows the four layers of the ZigBee communication stack, with their sub-layers, where the top two layers belong to the ZigBee stack proper and the lower two layer belong to the IEEE 802.15.4 protocol (Cunha et al, 2007).

<table>
<thead>
<tr>
<th>Application Layer (APL)</th>
<th>Application Framework</th>
<th>ZigBee Device Object</th>
<th>App. Support Sub-layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Layer (NWK)</td>
<td>Security Manager</td>
<td>Message Broker</td>
<td>Routing Manager</td>
</tr>
<tr>
<td></td>
<td>Medium Access Control (MAC)</td>
<td>Beacon-enabled Mode</td>
<td>CSMA/CA</td>
</tr>
<tr>
<td></td>
<td>Physical Layer (PHY)</td>
<td>2.4 GHz/250k bps/16 ch.</td>
<td>915 MHz/40 kbps/10 ch.</td>
</tr>
</tbody>
</table>

Table 2. The IEEE 802.15.4 / ZigBee protocol stack architecture

The ZigBee communication stack does not provide its own node addressing and routing but depends for these functions on the underlying IEEE 802.15.4 protocol’s short addressing mode and “mesh-under” routing (Shelby & Bormann, 2009).
Not all ZigBee nodes can be battery-powered devices as the nodes functioning as either coordinator or router need more energy. These nodes, because of their specialised function, must have their radios continuously on and must have sufficient memory to buffer messages to their client-nodes until these turn on their radio to read messages destined for them. This power scheme causes a limitation in the flexibility of the network installation as some wiring may be needed.

The use of designated routers leads to a multiple-star network topology where each router needs to provide its services to a number of (up to 14) end devices. This tends to limit the total network size. Routing takes place “on-demand” when a packet needs to be sent to a destination node. The state information is then stored in the node’s routing table. For larger networks this method may lead to overloading of the node’s constrained memory (Vasseur & Dunkels, 2010).

As the ZigBee stack was designed for node networks used with varying applications in building automation, remote control (ZigBee Alliance, 2009), smart energy, health care, home automation, etc that all use the underlying protocol layers but differ in their application layer formats, a “profile” was established for each of these fields that describes the parameters and variables to be used with each particular application field. This idea is appealing as it seems to be the only attempt to allow adaptation to a more specialised area in particular with regard to network size.

The proprietary and non-standardised nature of the ZigBee protocol stack has proven to be an obstacle for more general use in the Home Automation market although several attempts have been undertaken to overcome this problem: an open-source implementation of the ZigBee stack has been created for the TinyOS operating system (Cunha et al, 2007) as well as the creation of a draft RFC from the IETF Network Working Group to provide an UDP/IP adaptation of the ZigBee application protocol (Tolle, 2008). Finally, the ZigBee Alliance announced in mid-2010 that it will, in cooperation with the IPSO Alliance (IPSO, 2012), adapt its communication stack to be IP-based, at least for its smart-energy and home automation profiles (ZigBee Alliance, 2010).

5.4 IPv4

IP has been around since 1981 and is derived from the DARPA Internet Protocol. It allows a best-effort, unacknowledged, connectionless datagram delivery service. It is so wide-spread and so well known that a detailed explanation of its functioning would be misplaced in the context of this paper; it will be sufficient to recall the advantages and disadvantages for this protocol to be used in a Low-Power and Lossy Network (LLN). These listings have been largely gleaned from (Davies & Lee, 2003).
The advantages to the use of IPv4 in an LLN are the following:

- **IPv4 is a routable protocol.**

  The IP header contains both source and destination IP addresses as well as a link (hop) count. This allows the protocol to be routable through a limited number of links and to be discarded when the link count is exceeded. Each datagram, when needed, can take its own path through the network and may arrive in a different order. This is known as packet-switching.

- **IPv4 can carry multiple upper layer protocols.**

  In the same layer, IP can make use of ICMP and IGMP. In the next layer it can carry transport protocols such as TCP and UDP.

- **IPv4 is independent from the lower level Data Link and physical layers.**

  IP’s 32-bit addressing scheme allows it to be independent from lower level attributes such as MAC address, frame size and bit rates.

- **IPv4 allows fragmentation.**

  IP can adapt its packet size to accommodate for different sizes of lower level payloads through fragmentation and re-assembly of packets.

Unfortunately, IPv4 also has a number of disadvantages that make it difficult to use in resource-constrained LLNs:

- **IPv4 is demanding in terms of needed resources.**

  IPv4 is an extensive and large set of protocols that needs both sufficient computing power and memory space.

- **IPv4 has a limited address space.**

  The available public addresses are rapidly depleting. This has enforced the use of Network Address Translation (NAT) which enables a single public address to be shared by several private addresses. However, NAT defies what should be an end-to-end addressable and configurable network. Furthermore, it blocks security traffic such as that from IPSEC.

- **IPv4 has a hierarchical routing infrastructure.**

  The originally flat routing structure of the internet has given way to a hierarchical subnet structure in order to minimize the number of entries in routing tables. This is often wasteful of precious address space.
• IPv4 needs configuration.

Configuration of an IPv4 network must occur manually or be arranged automatically by the use of DHCP. However, DHCP servers need to be managed themselves.

5.5 IPv6

IPv6 is the successor to IP version 4 and tries to alleviate its disadvantages; in particular its address space limitations. However, it also offers many other improvements. Like IPv4, IPv6 allows a best-effort, unacknowledged, connectionless datagram delivery service. Like IPv4, the IPv6 protocol is well known and described in the literature. Here, I will concentrate on its importance within a LLN environment. If any disadvantages are retained from IPv4, it would be that IPv6 is also a large and demanding protocol. The following list is based on (Hagen, 2006).

The advantages to the use of IPv6 in an LLN can be listed as follows:

• It has been developed based on IPv4 and retains its good points.

  The versatility, reliability, scalability and independence of upper and lower stack layers have been fully retained and even extended. Scalability for example has been greatly extended by the larger address range of IPv6.

• The range of possible addresses is now $2^{128}$.

  Compared to version 4 which had an address range of $2^{32}$, the extended range is truly enormous. “This can be compared to providing multiple IP addresses to every grain of sand on the planet” (Hagen, 2006 p.36). For LLN devices which are expected to grow into many millions and by far surpass the number of PCs on the present internet, this is an essential feature.

  It also means that there will no longer be a need for Network Address Translation (NAT) and will restore the original end-to-end model which is important e.g. for implementation of IPSec and to simplify routing.

• Stateless auto-configuration.

  Centralised and managed host configuration of IP addresses by DHCP is no longer necessary. Host nodes can configure themselves with one or more valid IP addresses based on their MAC address (established as a unique number during manufacturing of the device) or as a random number. This Host ID will be combined with a prefix advertised by a router. If so desired, a form of DHCP can still be used to provide network options such as DNS server addresses.
• Simpler header format.

Instead of the variable length IPv4 header format (20-60 bytes), IPv6 has a fixed-length header of 40 bytes. Options are taken care of by extension headers that can follow the IP header if and when required.

• Broadcasts are no longer used.

Instead of broadcasts, multicasts are now extensively used. This avoids many problems with routing of messages (routers cannot forward broadcast messages).

5.6 Preferred choice of protocol

As can be seen from the above survey, a wide variation of communication protocols for LLNs is available. I would like to argue that two features are of overriding importance in deciding which protocol to use for resource-constrained networks:

• The by far most importance issue is standardisation of the protocol by an independent standards organisation such as ISO or IETF.

• The second importance issue is the availability of the protocol stack as open-source software.

Based on these overriding requirements I will use IPv6 as the communication protocol of choice for my project. Given the increasing market penetration of IPv6 for use with LLN devices I believe this to be a reasonable choice. Furthermore, the application of IP technology will provide the following benefits for LLN devices (Kushalnagar, 2007 p.4):

• IP-based technologies already exist, are well-known, and proven.
• IP is specified in open and freely available form and can thus be understood by a wider audience than proprietary solutions.
• Tools for diagnostics, management, and commissioning of IP networks already exist.
• IP-based devices can be readily connected to other IP networks such as the internet, without the need for translation gateways or proxies.

Further sections will specify more detailed requirements and will discuss upper and lower layer protocols as well as a suitable routing protocol to be integrated with IPv6.
6. Design of an Optimal Home Automation System

6.1 Choice of hardware

The hardware chosen for this project is a readily available commercial type derived from the original TelosB platform design (also known as the Tmote Sky) which was developed by the University of California at Berkeley. Its low-cost hardware design, shown in figure 1, is open-source and it has been used for many academic and scientific projects. The design is based on a 2.4 GHz IEEE 802.15.4 compatible transceiver. Table 3 gives a summary of its general characteristics.

![Figure 1 CM5000 Sensor Node](image)

<table>
<thead>
<tr>
<th>AdvantecSys CM5000 Sensor Node</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
<td>TI MSP430F1611</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>Program Flash</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>Data RAM</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>External Flash</td>
</tr>
<tr>
<td><strong>Interface</strong></td>
<td>Serial over USB</td>
</tr>
<tr>
<td><strong>RF Transceiver</strong></td>
<td>TI CC2420</td>
</tr>
<tr>
<td><strong>Sensor Light 1</strong></td>
<td>Hamamatsu S1087</td>
</tr>
<tr>
<td><strong>Sensor Light 2</strong></td>
<td>Hamamatsu S1087-01</td>
</tr>
<tr>
<td><strong>Temperature/Humidity Sensor</strong></td>
<td>Sensirion SHT11</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3 CM5000 Main Characteristics

6.2 Choice of firmware

As shown in section 5.6, the IPv6 protocol stack is an excellent choice to provide standardised and reliable communications in a smart node network providing its implementation can be made sufficiently small to accommodate the constrained environment.
A popular open-source software package for smart nodes is Contiki. This package has been designed and implemented on various platforms by the Swedish Institute for Computer Science (SICS). It is constantly maintained and extended and now forms a very complete solution. The software is written in ‘C’ and comprises a small but capable operating system, an IPv6 communication stack, routing software, implementing the IETF ROLL WG recommendations, named RPL (IETF ROLL, 2011) as well as a physical and data link layer implementation of the IEEE 802.15.4 specification. Furthermore, a 6LoWPAN adaptation layer is implemented (IETF 6LoWPAN, 2005). Table 4 shows the layered software approach of the Contiki architecture (Shelby & Bormann 2010). Further details are presented hereunder.

<table>
<thead>
<tr>
<th>User Apps</th>
<th>Built-in Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket API</td>
<td>Contiki OS</td>
</tr>
<tr>
<td>UDP</td>
<td>IPv6</td>
</tr>
<tr>
<td>TCP</td>
<td>RPL</td>
</tr>
<tr>
<td>ICMPv6</td>
<td>6LoWPAN Adaptation</td>
</tr>
<tr>
<td>Platform</td>
<td>Hardware Drivers</td>
</tr>
<tr>
<td>Hardware Drivers</td>
<td>CPU</td>
</tr>
</tbody>
</table>

### Table 4 The Contiki architecture

**Contiki operating system**

The Contiki operating system is a lightweight system designed for use with resource-constrained smart nodes. Its kernel is event-driven rather than multi-threaded in order to limit memory usage (multi-threading needs a stack maintained in memory for each thread). However, a multi-threaded programming abstraction called ‘protothreads’ is provided to make it possible to code in a multi-threading style while only using an overhead of two bytes per protothread (ACM Sensys 2006).

A file system named ‘Coffee’ is part of the OS. It allows accessing the node’s flash memory in a style similar to the normal ‘C’-language file access constructs.

**Data link layer**

The OSI data link layer (sometimes labeled ‘Layer 2’) enables medium access control functionality based on Carrier Sense Multiple Access (CSMA). This means that packets will be retransmitted for a limited number of times when sensing packet collisions or in the presence of radio interference (Shelby & Bormann 2010 p.129). The RIME library of communication functions provides low-level communication functionality that can be used for fast and efficient data collection from the nodes. The RIME functions are also used by the higher level IPv6 stack. Although this technique breaks the isolation
between layers, it does make for an efficient structure (Shelby & Bormann 2010 p.153).

The data link layer must conform to the IEEE 802.15.4 specifications. The link messages are by default secured by a built-in 128-bit AES encryption feature. As this would be a very processor-intensive operation, encryption takes place in the hardware of the CC2420 transceiver.

6LoWPAN adaptation layer

The IEEE 802.15.4 frame Maximum Transmission Unit (MTU) is 127 bytes minus the MAC frame overhead of 25 bytes and a security header of up to 21 bytes. This leaves only 81 bytes for the IPv6 payload. After removal of the IPv6 header, which has a fixed size of 40 bytes, and the upper layer protocol header, 8 bytes for UDP and 20 bytes for TCP, very little available space is left for the final application payload. Furthermore, IPv6 specifies a minimum MTU size of 1280 bytes which cannot be fulfilled by the IEEE 802.15.4 frames.

The 6LoWPAN adaptation layer can be seen as an additional layer between the IPv6 stack and the data link layer. It provides functionality to enable relatively large IPv6 packets to be adapted to the constraints of the IEEE 802.15.4 interface.

To bring the incompatible message sizes to match, the 6LoWPAN adaptation layer provides for packet fragmentation and reassembly as well as compression of the IPv6 main and extension headers (Shelby & Bormann 2010 p.41).

RPL

RPL is an open routing protocol for IPv6 smart node networks. It is subject to an ongoing specification process by an IETF working group: Routing Over Low-power and Lossy networks (ROLL). The RPL routing protocol functions at the IP layer and communicates with the network protocol using ICMPv6 (IETF ROLL, 2011). It will be discussed in more detail in a later section.

PowerTrace

For instrumentation purposes, a built-in application for measurement of energy usage of the different hardware and transceiver components has been added named PowerTrace (Dunkels et al, 2011a).

6.3 Simulation of a medium-sized network

Differences of scale are obvious when one compares a large smart objects network such as an urban traffic monitoring system perhaps consisting of thousands of nodes to a small home automation system constructed around a few dozen smart objects. Yet, these very diverse networks can use identical node operating systems, IPv6 stacks and RPL routing protocols. Are there
differences in configuration and protocol parameters between these applications? If so, is it possible to optimize a small home automation system for its particular environment? These are the questions that I would like to find answers to and use these results in my home automation network.

In order to find valid answers, a network needs to consist of a reasonable number of nodes. As a fully populated home automation project may well consist of several hundred nodes (Brandt et al, 2010), a test setup of, say, at least 40 nodes will be essential. This is much larger than my physical hardware setup consisting of 5 nodes. I will therefore use an open-source smart-object network software simulator/smart-node-emulator to perform the required tests. This will also allow good access to networking and routing data so that responses to parameter changes can quickly be evaluated.

A cross-level simulator/emulator, named Cooja, is available as open-source for the Contiki operating system and the RPL routing protocol. Cross-level in this sense means that simulation can take place at network level, operating system level, and machine-code instruction set level (Österlind et al, 2006).

6.4 Results of simulations

As home automation systems, more than larger smart object networks, may use battery powered devices that are, at least for the average HAS user, often placed in a relatively inaccessible position, it is of utmost importance that node energy consumption is minimised so that lithium cell batteries can typically last for several years (Vasseur, Dunkels, 2010). As most parts of a smart object device, such as the microprocessor, the memory and IO-circuitry have been designed for minimal energy usage and can, if needed, be brought into sleep mode for electrical current use in the µA region. However, the radio used by wireless devices does use an appreciable amount of energy, while transmitting. Also, surprisingly, listening for transmissions by other nodes uses as much power as the transmission mode. It is therefore essential that the radio is set up such that it is “duty-cycled” and only spends a small portion of time in its active state (typically less than 1% of the total time). Several special-purpose protocols, such as the ContikiMAC Radio Duty Cycling Protocol (RDC) (Dunkels, 2011b) have been designed to provide a power efficient wake-up mechanism and a set of timing constraints to smart object transceivers (Dunkels, 2011b, p. 1).

![ContikiMAC transmission timing sequence](image)

Figure 2  ContikiMAC transmission timing sequence (Dunkels, 2011b p.2)
Figure 2 shows a typical RDC timing sequence. The transmitter sends a series of data packets until an acknowledgement packet is received (or the sender times out). The receiver probes for activity on the radio channel (the referenced document, somewhat unfortunately, names this action ‘radio activity’). If the receiver does not detect a signal, it returns to sleep as seen by the first two receiver-on pulses. When it does see activity, it stays awake long enough to receive the package and acknowledge it. Should the sending packets belong to a broadcast, as opposed to a unicast transmission, an ack-packet is not expected. Instead, the packets will be sent for a full receiver wake-up interval.

The period of time within which the receiver repeats its listening probes (its wake-up interval) is an important parameter. Its default value is 8 Hz (125 mSecs) but can be overridden. This value should have a pronounced effect on the energy usage of the radio. To test this effect, I have set up a simulation of a smart object data collection network with one ‘sink’ (the collecting object) and 40 sensor nodes. The network uses UDP over IPv6 to periodically send a set of sensor values and node parameters to the sink. The simulation is run during 10 minutes of simulation time (which is different to real time as it depends on other processes run by the computer). In any case, this simulation time should be sufficiently long to allow startup of the node, setting up RPL routing and gathering enough node information to allow a trustworthy averaging of data.

The first test provides the data obtained from the network with the default RDC frequency of 8 Hz. Figure 3 shows a graph of the various power values per node. A second test was performed with an RDC frequency of 16 Hz followed by a third test with a frequency of 4 Hz. The related charts are shown by figures 4 and 5, respectively. Table 5 gives a summary of average power values at the different frequencies. The full data collection screen-shots are provided in appendix A7. Values and charts were obtained using the Contiki Data Collection application (Dunkels, 2012).

It can be seen from table 5 that the average listening power consumption rises with the RDC frequency whereas the average transmission power falls resulting in a total average power consumption that also rises with frequency. This situation is understandable as the number of listening on-time pulses per second increase with frequency while the number of transmission packets per burst is reduced because the ack-packet is received earlier.

<table>
<thead>
<tr>
<th></th>
<th>CPU Power</th>
<th>LPM Power</th>
<th>Listen Power</th>
<th>Xmit Power</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDC at 4 Hz</td>
<td>0.105</td>
<td>0.160</td>
<td>0.387</td>
<td>0.123</td>
<td>0.776</td>
</tr>
<tr>
<td>RDC at 8 Hz</td>
<td>0.095</td>
<td>0.161</td>
<td>0.558</td>
<td>0.066</td>
<td>0.879</td>
</tr>
<tr>
<td>RDC at 16 Hz</td>
<td>0.095</td>
<td>0.161</td>
<td>0.974</td>
<td>0.041</td>
<td>1.271</td>
</tr>
</tbody>
</table>

Table 5  Summary of average power values at different RDC frequencies.

The result of these tests is that the lowest RDC frequency of 4 Hz is the most suitable for battery-powered nodes. However, this comes at a price. The lower frequency will also result in lower network response as the receiver becomes
more sluggish to detect packets translating into higher network latency. For most situations in a Home Automation System this will be acceptable as sensor values are monitored at relatively large time intervals (typically 1 minute). Slower network response during neighbour detection and setup and repair of RPL routes (Korte et al, 2012) may also remain within reasonable bounds especially with the relatively small number of nodes usually found with a Home Automation network.

Ultimately, a long-term test of the physical network will tell whether or not the network remains sufficiently responsive and stable.

As the number of network nodes grows, the number of nodes that is not within direct radio reach of the sink node will also increase. Packets to the sink from these nodes will have to be routed through more nearby nodes to find a multi-hop route to the sink. In principle, all nodes, besides running applications, can also act as a router. The RPL routing protocol, as a distance vector protocol, can establish a Destination-Oriented Directed Acyclic Graph (DODAG) that describes the possible routes through the network from the nodes to the sink (Winter et al 2011).

![Average Power Consumption](image)

Figure 3 Average Power Usage at an RDC frequency of 8 Hz.
Figure 4 Average Power Usage at an RDC frequency of 16 Hz.

Figure 5 Average Power Usage at an RDC frequency of 4 Hz.

The RPL protocol uses a so-called ‘objective function’ (OF) to describe the routing rules to be used. Although not identified as such in most of the Contiki RPL documentation, the default objective function used is the Minimum Rank with Hysteresis Objective Function (MRHOF) (Gnawali & Levis, 2012). This OF is based on a suitable routing metric which must be additive over the route
taken. In this case the number of expected (re-)transmissions (ETX) has been chosen. A possible alternative is the Objective Function Zero (OF0) (Thubert, 2012). It bases its very simple routing rule on the number of hops which it tries to minimise.

The routing parameter that is used to determine the best route from a node to the sink is expected to have a large influence on power consumption throughout the network. As mentioned above, by default, ETX is used here. This parameter provides a forecast of how many times a packet will need to be sent and possibly re-sent before it arrives at the destination; the optimum value is therefore 1. In a relatively small network for Home Automation, ETX might not be the sole optimal routing parameter to use. A relatively low number of lost packets to be retransmitted are not likely to clutter up the network. More important, once again, is the minimisation of energy used by the battery-powered nodes. This could be done by introducing an objective function that causes the routing protocol to avoid battery-powered nodes. However, in a small wireless Home Automation network, most, if not all, nodes will be battery-powered. An alternative would constitute the use of objective function zero (OF0). It will cause the routing protocol to minimise the number of hops in a multi-hop environment. This means that a smaller number of nodes will have to double as a router. Since the router function consumes power, router nodes will use more energy than mere sensor nodes and therefore deplete their batteries faster. As the router nodes are crucial to the operation of the network, depleted batteries may cause the entire network to collapse. This larger power consumption of routing nodes can be seen in the graphs of figure 3 and 5 which show an environment with a maximum number of 3 hops where the ‘spiking’ vertical bars belong to the routing nodes (figure 4 shows a 2-hop environment which results in less pronounced spikes).

Since a minimum number of hops in the network is of obvious advantage to minimisation of the overall power consumption, I have run a simulation of a 40-node network based on routing by the zero objective function. The RDC frequency has been reset to 8 Hz so as not to confuse results. Figure 6 shows the resulting graph which should be directly comparable to figure 3.

The number of routing nodes has now been reduced from 3 to 2. However, although the overall total network power consumption has decreased slightly, the routing nodes remaining now have more work to do than before and therefore use more power. Moreover, as the network grows to more nodes, an even larger number of hops will result. Table 6 shows the average differences between the objective functions as well as the specific power consumption of the routing node (node 2).

<table>
<thead>
<tr>
<th></th>
<th>CPU Power</th>
<th>LPM Power</th>
<th>Listen Power</th>
<th>Xmit Power</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>OF-ETX</td>
<td>0.095</td>
<td>0.161</td>
<td>0.558</td>
<td>0.066</td>
<td>0.879</td>
</tr>
<tr>
<td>OF-0</td>
<td>0.091</td>
<td>0.161</td>
<td>0.533</td>
<td>0.060</td>
<td>0.844</td>
</tr>
<tr>
<td>OF-0 (Node 2)</td>
<td>0.164</td>
<td>0.159</td>
<td>0.695</td>
<td>0.100</td>
<td>1.118</td>
</tr>
</tbody>
</table>

Table 6  Summary of average power values at different objective functions.
As discussed before, the entire network depends on the routers remaining functional. I therefore conclude that introducing objective function zero does not lead to improvement of network power consumption. The default ETX objective function remains the better choice between the two.
A still better solution to obtain optimal routing would be to ensure a small number of mains-powered nodes are installed at strategic places to act as routers for the battery-powered nodes. The effect of placing mains-powered routers in a small network is shown in figure 7. This figure shows the router nodes (yellow coloured) as being placed just within the maximum radio distance to the sink node (green). These routers can thus relay radio traffic from the more distantly placed sensor nodes, as well as their own sensor data, to the sink. The result of this arrangement is shown by figure 8.

The effect of the packet-gathering router nodes, 2, 3 and 4 can clearly be seen. Nodes 8 and 14 are sufficiently close to the sink to directly send their data to it.

Judiciously placing the routers, although of prime importance, should be aided by the routing algorithm so that mains-powered routers are preferred over battery-powered nodes while building the DODAG. However, the only way to distinguish between a node being mains or battery-powered would be by measuring the supply voltage which, for mains-powered nodes, would be slightly higher (approx. 3.5 Vdc) than for battery-powered nodes (2.8 – 3.2 Vdc). In practice, certainly while using the simulator, it will be simpler to use a flag variable to distinguish between the two.

The firmware changes necessary to implement the power algorithm are documented in annex 8. The main changes take place in objective function file ‘rpl-of-etx.c’ which will now update the RPL-DIO message metric container.
for energy rather than the ETX additive metric (Vasseur et al, 2010). Internal function ‘best_parent’, which selects between two parent nodes, has been supplied with additional code to choose a mains-powered parent if possible.

6.5 Physical implementation of a small-size network

As argued previously, most smart nodes in a Home Automation network must be battery-powered to allow for a flexible installation. From the simulations, it can be seen, however, that placing mains-powered router nodes at strategic points would be useful to extend the overall battery lifetime of the system. One of these mains-powered routers could be placed at each floor of a house to allow maintenance-free routing at that level. All remaining nodes can then be battery-powered with the exception of an outdoor node which could be solar-powered.

The data sink or LoWPAN Border Router (LBR) can be a USB-powered node, physically identical to the others but designated as “sink” by its internal firmware or it could be a more elaborate UNIX bridge with more powerful hardware and possibly incorporating a web server. This would allow the installation to function, even remotely monitored, without a dedicated server PC.

6.6 Home automation profile

A Home Automation profile must be seen as a set of network and routing parameters that are specifically tuned for use of the smart node network in a home automation setup. Simulations have shown that a number of parameters are of prime importance for a flexible, reliable, maintenance-free and long-lived Home Automation network:

- Use of an RDC frequency lower than the standard one of 8 Hz.
- Strategically placed mains-powered routers that have their radios continuously on.
- A routing objective function that favours mains-powered routers over battery-driven nodes.
- A secondary preference favouring routes with a low ETX value.

6.7 Using web services

The implementation of a Home Automation network has so far been based on distributed network nodes that report their sensor values and network parameters to a LoWPAN Border Router (LBR) node that periodically collects the data and forwards it to the server to which it is attached. It does this
through a USB connection that, by the use of special drivers, has been configured to act as a serial COM port. Although this arrangement works well it would be more advantageous if an Ethernet connection could be used instead so that the network’s IPv6 protocols could directly communicate with the internet.

Such an Ethernet connection requires hardware changes to the connecting node that presently are not feasible. However, an Ethernet connection could be emulated through the use of an older protocol named Serial-Line IP (SLIP) (Romkey, 1988). A modern Linux implementation of this protocol, named ‘tunslip6’ has been adapted for IPv6 and is available with the Contiki tools. This solution still uses the serial port but it will be “abstracted away” (except that the port still needs to be properly configured) and make both sides of the connection see IP datagrams rather than a serial data stream.

Furthermore, the reporting nodes, instead of working through an application on top of its internal IPv6 network stack, could have a web server installed that can directly deliver its functional data upon demand from a remote browser. Such a web service, due to the verbosity of SOAP, is usually set up in a REST configuration. This is more in line with the limited resources available.

A test version of a Contiki node firmware using a simple REST web service is freely available. It is possible to run this version in the Cooja simulator and then use tunslip6 from a Linux virtual machine (within a Windows PC) to create an emulated IP connection to the network. The web services on the nodes can in that way be accessed, in the same Linux VM, through a command line tool for transferring URL-based data named ‘curl’. I have re-programmed the web services on the nodes to provide a simple readout of sensor values. The LBR, as a special case, will show a list of detected nodes and routes by means of an HTML web page.

Figure 9. Screenshot of the simulator set-up for web services.
Figure 9 shows the simulator set-up, where it can be seen that node 6 is placed farther away so that the LBR can only establish a connection with it by using another node as an intermediate router. Figure 10 shows a screenshot of the LBR browser page. The ‘Routers’ block shows that the connection to node 6 is established through node 4.

Figure 10. Screenshot of the LBR page.

Figure 11 shows some curl commands and results obtained from the REST web services installed on the nodes. The last set shows that node 6 is responding to the commands as it should.

Figure 11. Screenshot of some ‘curl’ commands and results.
Figures 12 and 13 show the network protocol sequences when posing a REST service request and its corresponding response. In this case, a request for the temperature sensor value of node 3 is used. Note that the response of 6360 is the raw value of the sensor that still needs to be converted to degrees C or F.
These screenshots were obtained using the WireShark Network Protocol Analyser.

One problem seen with the use of web services on the nodes was that it stretches the amount of available memory to the limit. The web server nodes need to have an installation of the Operating System (Contiki), the MAC Layer (ContikiMAC), the IPv6 communication stack, the routing protocol (RPL), the HTTP server as well as the REST application code. I found that this would only just fit the available space with web services being limited to the four sensor values and the battery voltage value. Even adding code for floating-point conversion of the raw sensor data into humanly meaningful values was too much. This conversion must be left to the external monitoring software. Adding services to provide various node parameters such as hop count, ETX, rank, etc. was not possible. Another type of node with more program memory will be needed to extend services. Figure 14 shows the printout resulting from a ‘size’-command. The flash memory would have to accommodate the block indicated as ‘text’ and ‘data’ being $47802 + 336 = 48138$ bytes. Its current size is $48$ kB or $48 * 1024 = 49152$ bytes; thus only $1014$ bytes are left free.

![Output from the Contiki ‘size’ command.](image)

Given the problem with the currently used hardware combined with the impossibility of reworking the complete software in time for this report’s deadline, I have restricted research on the use of web services to the simulations as discussed above. Testing web services on actual hardware nodes, incorporating a larger program memory block, is beyond the possibilities of this study.

Listings of all Contiki modified files are provided in appendix A8.
7. Software Design and Implementation

7.1 System design

For the Home Automation project a software system needs to be designed that can issue commands to the LLN network, collect and store data and display current network status and sensor data to the user. The system should also be able to provide statistical data about sensors and network operation over time. The primary user interface will be through a web browser so that remote monitoring and network control becomes possible.

The software methodology used is Extreme Programming (XP), a version of agile development that is particularly well suited for relatively small projects. “XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements.” (Beck, 1999 p. 7). XP is based on almost continuous integration, testing, and refactoring (Martin, 2009). This continuous integration and testing of the product can be greatly facilitated by a good (preferably ‘one-button’) build process (Schuh, 2001). Interestingly, this simple way to compile and build rapid iterations of the product is found back in the IDE usually associated with component-based development environments such as Microsoft .Net which is used for part of this project.

For this project, a database needs to be designed that can store the periodically collected sensor and node data as well as other data and settings. In particular, the various types of nodes need to be enumerated together with the available sensors and actuators. The nodes must be specified with their location, type, function, status, etc. Each node will have a series of configuration settings such as the data collection interval and its associated reporting deviation time (random delta from the interval setting to avoid all nodes reporting at the same time).

Various types of sensors are used by the project such as sensors for visible light, infrared light, humidity, and temperature. These sensors need to be listed in the database together with the identification of the node on which they are located.

Actuators can be dimmers, on-off switches, regulators, etc.

Measurement units will need to be indicated in full as well as abbreviations used.

System users will need to be authenticated to avoid non-authorised access. Relevant user and administrator data will therefore need to be stored in the database.

Although a web application could be constructed that implements access to the server’s serial port to control the LLN network, the periodical reading of serial data is not a convenient web server task. Two applications are therefore implemented: a web site that allows a user access to monitoring and statistical functions as well as a Windows application that provides all serial communication functionality. Both applications can access the common
database and the web site can also give remote commands to the data collection application through a command text file. This setup is shown by the block diagram of figure 15.

![System block diagram](image)

**Figure 15.** System block diagram

### 7.2 Database design

The database needed for the Home Automation project has been designed to hold the data delivered by the network nodes and sensors as well as general information such as measurement units (like mW or °C) and names of parameters. Furthermore, user information is stored to allow authentication and access authorisation. The design has been fully normalised to ensure use of data tables without redundancies and multiple empty attribute fields. In this way, it allows efficient use with a variety of node types and sensors that may require different attributes to be stored. The Entity-Relationship Diagram in appendix A4 shows these details together with tables of the normalising stages.
Parameter tables are used to hold node and sensor attribute names rather than using pre-named fields to the data tables. This ensures that changes or additions to the nodes or sensors used within the network will not necessitate changes to the underlying database design (DDL) but only to changes, additions, or deletions to the contents of the parameter tables.

This design leads to the use of a separate record for each node and sensor data reading. Although this may seem extravagant, it has the advantage of the use of separated parameter tables as explained above as well as the use of a unit table to hold the required measurement units. As both node and sensor data tables use a 32-bit auto-increment integer as a primary key, a total number of approx. $4.3 \times 10^9$ records are possible. In the present Home Automation test setup with four network nodes and reading data records once every minute it would take approximately 720 years to exhaust these record ranges.

The use of small data records also leads to rapid and efficient read operations in the database as the single data value attributes as well as the accompanying system time fields are indexed.

Indexes have been set up on all attributes that may be accessed frequently, especially those that may form part of a ‘join’ expression. Foreign key attributes are inherently indexed. To ensure read and write operations to the database are simultaneously possible without long locking periods, the mySQL ‘innodb’ engine has been specified as it only locks rows rather than the more coarsely page or table locking.

### 7.3 Data collection design

Data collection for the Home Automation project is implemented by means of a MS Windows project named HomeCollect. It is written in C# using MS Visual Studio 10.

![HomeCollect sequence diagram](image)

Figure 16. HomeCollect sequence diagram
The requirement specifications and process/product quality measures for the data collection application are available in appendix A5.1 and A5.2.

A serial communication link is implemented using one of the computer’s COM-ports. Commands to the network nodes can be sent there so that they are received by the ‘sink node’ connected to the serial port and, if necessary, forwarded wirelessly to the remote nodes. Data received from the nodes is collected by the sink node and presented to the serial port from which, in turn, the data can be read by the HomeCollect application.

The serial port settings provide a communication rate of 115200 baud. At this high speed, I found that occasionally transmissions of a command will fail and the sink node will return a message that contains the phrase ‘command not found’. When this return phrase is detected, the command string will be sent repeatedly up to five times. If it then still fails, an error message will be displayed. After this system was put in place, I have not seen any more command error messages (except when mistyping the command).

The data received can be either command acknowledgements or records with collected node and sensor data. Complete commands usually consist of several sequential command strings separated by new line characters. For convenience, frequently used command sequences are stored in text files. This allows rapid access to the command sequence needed as well as easy script update maintenance without changes to the HomeCollect software itself. Currently, four text files are used:
• Open.txt holds the sequence of commands needed to open communication with the nodes and start periodical collection of node data.
• Manual.txt has a sequence of commands used for a single node data collection burst. This can be useful for test purposes.
• Close.txt will stop communication with the nodes and discard any buffered data.
• Reboot.txt sends a sequence of commands used to reboot all remote nodes as well as the sink node.
• Command.txt will be filled interactively by the user before its commands are executed.

As an example, the contents of file Open.txt is shown by listing 1.

```
netcmd killall
~K
killall
mac 0
time %t | null
collect | timestamp | binprint&
netcmd { repeat 0 60 { randwait 10 collect-view-data | send 31 } }
```

Listing 1. Contents of Open.txt text file

![Screen shot of the HomeCollect user console after manual data collection.](image)

Figure 18
When HomeCollect is started, a user console is presented that allows various actions to be performed such as rebooting the nodes, starting a periodical data collection process, halting this process, performing a single data collection burst from a single node or sending a manual sequence of commands to the node network. A screen shot of the HomeCollect user console after running a manual data collect sequence is shown by figure 18. Displayed is some of the data from node 4. Scrolling down the serial communication text box would reveal the remaining lines. The command input box shows a sample of a command sequence that could be sent to the network nodes by pressing the send command button. This sample set of commands would flash the nodes’ onboard leds three times including the ones on the sink node.

During periodical data collection, the current settings provide for collection of the data from each node to occur once per minute with a random time difference of 10 seconds. The manual collection of a single node’s data string delivery is set for 10 seconds (randomised within 2 seconds). This value is more in line with the testing character of this mode of operation. These settings mean that, even in manual mode, the various nodes’ data records continue to be delivered periodically to the serial input buffer. Each time the manual data collect button is pressed the next record is fetched from the buffer and displayed. The first time, the manual data collect button is pressed, the display will show ‘no data’ as the buffer is still empty. A further press on the button will display the first data record. Please note that display of the data only occurs in manual mode as during periodical data collection there is no necessity for display of the separate data records; instead they are silently added to the database.

The current time placeholder (%t) in the open.txt script is replaced by a Unix timestamp (number of seconds after 1st Jan. 1970) before being sent to the nodes with the data collection start commands. The nodes will use this timestamp as their initial time and can, in this way, deliver a valid systemTime parameter with their serial data strings.

After collecting a data record from one of the connected nodes, the string is tested for validity. Unfortunately, a checksum is not transmitted by the nodes so that other means of validation need to be used. In the case of this data sequence, a test is performed to ensure the first sub-string is ‘30’ (the number of sub-strings in the sequence) and another to ensure the last three sub-strings are all ‘0’. In case an invalid string is found, it is discarded and, when in manual mode, an error message is displayed. Figure 18 shows a sample serial data string.

When a valid string is found, it will be parsed to load the sub-string values into an associative array (or the nearest C# construct being a ‘dictionary’) where the keys are the names of the sub-string variables and the values are the transmitted sub-strings. The necessary keys are obtained from a string array as these are not available from the transmitted data string. Furthermore, some values are only used to provide the calculation basis for more complex variables. These calculations are performed after all sub-string values have been handled. A number of constants are used with the calculations that
depend on the particular type of nodes used. As for this study only one type of hardware node is used, these constants are hard-coded in the HomeCollect software. An example would be constant `POWERTRANSMIT` that equals 17.7 mW multiplied by the node supply voltage (also a constant) and which forms the basis for the calculation of the effective wireless node transmission power.

As a result, a set of key-value pairs are now available that can be stored into the Home database. Please note that all necessary parameter and measurement unit names are pre-loaded into the database and are referred to by the stored data records. These parameters are read from the database and stored in further dictionaries to allow them to be referenced by foreign keys in the sensor and node data records that are periodically stored. Please refer to the chapter on database design for further details.

To ensure that periodical data collection is properly started and functioning, a test function is implemented that reads the number of sensor data records from the database repeatedly. Only, when this number is seen to be increasing, the ‘Data Collection Active’ textbox will be shown (see figure 19). When this text does not show within approximately 20 seconds, a fail message will show instead and the data collection should be stopped, the nodes rebooted, and data collection restarted.

![Figure 19. Screen shot of HomeCollect while in periodical data collection mode.](image)

The execution time of the parse function is measured and displayed while in manual data collect mode by a simple instrumentation structure that measures
the execution time of the data parse routine. This allows an evaluation of the timely functioning of the routine especially when a higher number of network nodes are present. The parse function has been shown to usually take about 800 mSecs to handle each data record including database storage.

The HomeCollect software contains a separate class for all routines related to database access (Rached, 2009). An initialising method obtains the parameters necessary to connect to the associated database which is named ‘Home’ from the application’s configuration file. Further methods are available to open or close a connection to the database as well as a set of methods to allow the basic database access functions: insert, update, delete, and select. A separate ‘SingleSelect’ method allows a select query that returns a scalar value instead of an array. Finally, methods to backup or restore the entire database are implemented here. Buttons available on the user console give access to these backup and restore functions.

A third class contains methods pertaining to timing needed for the periodical collection of data strings. This timer will fire an event every 5 seconds to test for a newly arrived data string. Timer start and stop methods are available to control the timer function.

By using separate classes for the database and timer functions, cleaner and more structured code is obtained.

During its periodical data collection mode, the HomeCollect application can be remotely controlled in most of its main functions. This allows a remote monitoring application to start and stop periodical data collection, reboot the nodes, and send command sequences to the nodes. This mode of operation is achieved by having the timer that controls the periodical data collection to also check the contents of a text file for remote commands. When the text file is empty, nothing happens; when it contains a certain keyword, the associated action will be executed. The available keywords are ‘open’, ‘close’, ‘reboot’, and ‘blink’. After reading the keyword, the file is truncated to be ready for the next periodical inspection. An unknown keyword is silently discarded without further action. The remote application, in this case the web site, is responsible for writing the required keywords to the file.

7.4 Web-based monitoring design

In order to keep track of Home Automation network functioning and various sensor readings, an online web application has been developed that allows to execute these monitoring functions from anywhere through the internet. Although in principle, any browser can be used, the site has been optimised for FireFox.

The requirement specifications and process/product quality measures for the web-based monitoring design are available in appendix A5.3 and A5.4.

For the web site implementation, the Zend Framework has been used to standardise the development and offer Model-View-Controller (MVC) pattern
structure. Consequently, as Zend is based on PHP, this language has been used as a programming language (Vaswani, 2010) (Allen, 2009). Furthermore, the Doctrine Object to Relational Mapper (ORM) has been used to structure database access (Wage, 2009). The site runs on XAMPP that also contains mySQL and uses an Apache server on Windows Server 2003(R2). Security is provided by Windows ISA Server 2004. Finally, PHPUnit has been used to provide some rudimentary error checking.

Figure 20. eHome web site use case diagram

The home page of the eHome web site, which can be reached after a successful user (guest, user, admin) login, offers a choice of four main areas to monitor and control the Home Automation network:

- Network Control leads to a page that allows basic control of the data collection aspect of the network (admin access only).
- Home Monitor opens a page that allows inspection of recent measurement values of the various sensors of the network. This would normally be the most frequently visited page (guest, user, admin access).
- Show Statistics opens a page to select node statistics and to ensure the network is functioning properly (user and admin access).
Maintain Database leads to a number of simple CMS pages where various actions can be undertaken to keep the database in optimal condition (user read-only and admin full access).

The Network Control page contains links (buttons) to a number of simple network control actions such as rebooting the nodes, opening and closing the data collection, and blinking the node leds. The latter action represents a node activator control function and may later be extended to several individual activator functions such as control of a radiator thermostat or light controls. As it proved difficult to run the continuous serial data collection process in the background (PHP does not use a multi-threaded programming model), a different approach was used where the data collection process runs in a separate application (see the previous documentation on the HomeCollect application). Therefore, all links function by writing a particular keyword to the network communication text file. This keyword is picked up by the data collection application that must be running in the background. After writing the keyword, a 10 second loop is entered that checks whether the keyword has been read by the remote application and whether the text file is now empty. If it is, the loop is left and a communication success message is displayed. If, after 10 seconds, the text file still contains the keyword, a fail result message is displayed.

Ajax and JavaScript are used to handle the button functions. While a function runs, a revolving icon is displayed to show that a wait period is in progress. Please note that in line with the MVC principle, most of the php code that contains the functionality is placed in a model file while the controller only contains the code to call the class and function needed.

The Home Monitor page allows the choice of a particular node based on its location. Each node has its own link, consisting of a selectable image. When one of the images is selected, a new page will show a table of the most current values from all sensors associated with that particular location node. This page also provides a link to show a line chart of all stored values of these same sensors over the last 24 hours (or less, should the database have less than 24 hours worth of values). As there are a variety of sensor data with widely different values, the line charts have a choice of sensors that can be selected from a drop-down list. Please note that hovering the mouse over one of the chart’s lines will show its name, time and measured value at that particular point. A further drop-down list allows a choice of statistical data to be displayed in an associated textbox. A calendar form based on the Dojo toolkit allows inspection of data within different time intervals. To ensure the amount of data to be read from the database remains reasonable, the queries are laid out such that only one out of the number of days of records are fetched. In other words: for the data over one day (24 hours) all records are fetched; for a time interval of ten days, only one out of ten record blocks with the same timestamp are read from the database. In this way, the amount of data fetched will never exceed that of 24 hours.

The tables and line charts are based on the charts web services provided by Google. These web services are based on an AJAX JavaScript API. As usual
with Zend, the javascript functions cannot be placed directly in the html code of the view. Instead, a set of placeholders must be left embedded in the html view (or, as in this case, because of its use with several different pages, in the master.phtml layout code). The associated chart controller will refer to these placeholders and tie them to the Google-specified URL and its parameters (Stoimen, 2010).

The ‘Show Statistics’ page, like the Home Monitor page, allows the choice of node based on its location. Also, like the monitor page, tables and line charts are used based on Google Charts. However, here, instead of sensor values being displayed, node and network parameters are shown. Like with the sensor charts, there are a variety of node parameters with widely different values; hence, there is a choice of related sets that can be selected from a drop-down list. A secondary (dynamic) drop-down allows selection of a specific graph. Statistics are shown in a textbox. The default shown is a set of power-related graphs with the statistical value at zero. Here also, a calendar form allows inspection of data within different time intervals. By default, the last 24-hour period is shown. For longer time intervals, a record fetching limitation similar to that for sensors is applied.

The Maintain Database page implements a partial CMS system that allows displaying, creating, updating, and deleting records of the various database tables. Tables NodeReadings and SensorReadings that hold records from the automatic data collection process only have a delete function for all records before a specified date. This function can be used to delete records that, because of age, are no longer of interest.

To improve performance, a Memcache service has been installed on the server. As Doctrine has good support of Memcache, the Zend bootstrap file has been extended accordingly so that queries that have been parsed previously can be found in the Memcache memory and thus speeded up. A data result cache was not implemented as data will constantly change and therefore not be cachable. Please note that the mySQL internal caching mechanism has been disabled as it uses frequent global locks on the entire database which could interfere with the data collection process.

Error checking, in particular of the controllers, has been implemented to some extend. Figure 21 shows the result of a PHPUnit test run. The tests are based on a web article (Cargnelutti, 2008) as mentioned in this report’s bibliography.
Figure 21. PHPUnit controllers test run.
8. Project Evaluation and Conclusions

Home Automation can be more flexible, reliable, and easier to install when using proven, open-source protocols for the network stack and routing of messages yet have a sufficiently small footprint to be used by resource-constrained smart nodes. The simulations have shown that a Home Automation setup can profit from carefully adjusted parameters. In particular, minimising power consumption of the nodes is of prime importance.

Installing a small number of mains-powered nodes in strategically chosen places can aid in limiting the overall power consumption of the battery-powered nodes which form the majority of the network nodes.

The artifacts developed for this project allow monitoring of the sensor values locally or from any remote location. Furthermore, network parameters can be monitored so that early correction of a problem (e.g. a low battery) becomes possible. Long-term graphs and display of statistical values aid in early diagnosis of errors.

Testing during a time period where the clocks where adjusted to winter time (27-28 Oct) showed that no problems occurred. The time period shown was 23 hours, in accordance with the loss of one hour during the night.

Additionally, some experiments have been conducted to evaluate a direct Ethernet network connection as well as the use of web services on the nodes. These experiments show that an Ethernet connection can be simulated by means of a SLIP protocol. It has also been shown that the web services that can provide sensor and parameter data on-demand from a remote web-site or other application are very feasible and would provide a more flexible configuration. However, for this thesis, these changes, which would include extensive re-programming of the node’s web service firmware, the external eHome web-site, and use of hardware with more available memory, come “too late in the game”.

9. Future Work

This project can be seen as an important first step towards the design and installation of a Home Automation System. However, many tasks remain to build a more complete system going beyond mere monitoring of some important environmental parameters. Some of these tasks are listed here:

- Installation of more nodes in the physical network will allow better and more complete monitoring of essential environmental values. In particular, an outdoor sensor would add value to the simple network as it exists. A different type of hardware for the nodes may provide more application memory space.

- Acting on measured values such as needed for heating regulation or raising/lowering blinds will be important and go far beyond the simple “led flashing” example that has been implemented. This will have a large impact on the complexity of the application firmware in the nodes. Closed-loop control software between, say, a temperature sensing node and another node regulating a heater valve will need to be reliable and free of oscillation.

- Activator nodes reacting to measured values elsewhere in the network would need direct communication between the nodes involved, not just “dumping” values at a sink node.

- The use of web services offers an exciting possibility to increase flexibility and would alleviate the need for a separate data collection application.

- A further field of investigation would be a means of remote firmware updating to the nodes “in-situ” through the wireless network rather than having to physically collect them for connection to a usb-hub.

- Some of the conclusions of this study such as stability and reliability can only be definitely proven by long term operation of the physical system incorporating a large number of active nodes using sensors as well as activators.
10. Bibliography


11. Appendices

A1. List of software used

Zend Framework 1.11.3
PHP 5.3.8
Doctrine ORM 1.2
XAMPP 1.77
mySQL 5.5.16
Apache 2.2.11 VC9
PHPUnit 3.5.14
MS Visual Studio 10
MS Project 2003
VMWare Player 5.0.0
Ubuntu OS 12.04
Contiki 2.6
WireShark 1.7.2

A2. List of abbreviations used with smart node networks

LLN Low-power and Lossy Network
RPL IP Routing Protocol for LLNs
BER Bit Error Rate
PDR Packet Delivery Ratio
PLC Power Line Communication
DODAG Destination Oriented Directed Acyclic Graph
ETX Expected number of Transmissions
LBR LoWPAN Border Router
DIS DODAG Information Solicitation
DIO DODAG Information Object
DAO DODAG Destination Advertisement Object
MP2P Multipoint-To-Point
P2P Point-To-Point
6LoWPAN IPv6 over Low Power Wireless Network
ND Neighbour Discovery
LPM Link Power Management
JTAG Joint Test Action Group
DSSS Direct-Sequence Spread Spectrum
OCP Objective Code Point
A3. Project plan and timeline

My project plan has been prepared using Microsoft Project and is shown in figure A3.1. Please note that, although some tasks could have been started earlier and run in parallel, this is of course not possible with a single person’s involvement; thus, all task are depicted sequentially, the only exception being task 8 (Setup Development Environment) which can be done at any convenient time.

![Project Plan](image)

Figure A3.1 Project Plan

A4. Database design.

The Home database tables before normalisation are shown below (only important attributes are shown).
Possible multiple occurrences of attributes are shown in angle brackets.

Home (nodeId, nodeId, nodeTypeId, nodeTypeName, available, locationId, locationName, [nodeReadingId, systemTime, nodeParamId, nodeParamName, paramValue, unitId, unitName], [sensorId, sensorTypeId, sensorTypeName, sensorParamId, sensorParamName, sensorValue, unitId, unitName, enabled])
The Home database tables in first normal form:

Node (nodeId, nodeTypeId, nodeTypeName, available, locationId, locationName)
NodeReading (nodeReadingId, nodeId, systemTime, nodeParamId, nodeParamName, paramValue, unitId, unitName)
Sensor (sensorId, nodeId, sensorTypeId, sensorTypeName, sensorParamId, sensorParamName, sensorValue, unitId, unitName, enabled)

As there are no multiple keys, the second normal form is equivalent to the first.

The Home database tables in third normal form:

Node (nodeId, locationId, typeId, available)
Location (locationId, locationName)
NodeType (nodeTypeId, nodeTypeName)
NodeReading (nodeReadingId, nodeId, systemTime, paramId, paramValue, unitId)
NodeParam (nodeParamId, nodeParamName)
Sensor (sensorId, nodeId, typeId, enabled)
SensorReading (sensorReadingId, nodeId, systemTime, paramId, sensorValue, unitId)
SensorParam (sensorParamId, sensorParamName)
SensorType (sensorTypeId, sensorTypeName)
Unit (unitId, unitName)

Finally, as an example, some of the table data contents are shown below.

<table>
<thead>
<tr>
<th>id</th>
<th>mUnit</th>
<th>unitName</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>lx</td>
<td>lux</td>
<td>incident light level</td>
</tr>
<tr>
<td>2</td>
<td>°C</td>
<td>degrees Celsius</td>
<td>temperature reading</td>
</tr>
<tr>
<td>3</td>
<td>°F</td>
<td>degrees Fahrenheit</td>
<td>temperature reading</td>
</tr>
<tr>
<td>4</td>
<td>% RH</td>
<td>relative humidity</td>
<td>humidity reading</td>
</tr>
<tr>
<td>5</td>
<td>mW</td>
<td>milliwatt</td>
<td>power level</td>
</tr>
<tr>
<td>6</td>
<td>V</td>
<td>volts</td>
<td>voltage level</td>
</tr>
<tr>
<td>7</td>
<td>s</td>
<td>secs</td>
<td>Seconds</td>
</tr>
<tr>
<td>8</td>
<td>none</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>

Table A4.1 Unit Table

<table>
<thead>
<tr>
<th>id</th>
<th>paramName</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Light1</td>
<td>Sensor Light 1</td>
</tr>
<tr>
<td>2</td>
<td>Light2</td>
<td>Sensor Light 2 (IR)</td>
</tr>
<tr>
<td>3</td>
<td>Temperature</td>
<td>Temperature Sensor</td>
</tr>
<tr>
<td>4</td>
<td>Humidity</td>
<td>Humidity Sensor</td>
</tr>
</tbody>
</table>

Table A4.2 SensorParam Table
<table>
<thead>
<tr>
<th>id</th>
<th>paramName</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SeqNo</td>
<td>IP sequence number</td>
</tr>
<tr>
<td>2</td>
<td>Hops</td>
<td>No. of hops through the network</td>
</tr>
<tr>
<td>3</td>
<td>Latency</td>
<td>Network latency</td>
</tr>
<tr>
<td>4</td>
<td>CPUPower</td>
<td>Power in mW</td>
</tr>
<tr>
<td>5</td>
<td>LPMPower</td>
<td>Power in mW</td>
</tr>
<tr>
<td>6</td>
<td>ListenPower</td>
<td>Power in mW</td>
</tr>
<tr>
<td>7</td>
<td>TransmitPower</td>
<td>Power in mW</td>
</tr>
<tr>
<td>8</td>
<td>AveragePower</td>
<td>Power in mW</td>
</tr>
<tr>
<td>9</td>
<td>PowerMeasureTime</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>BatteryVoltage</td>
<td>Voltage of node battery</td>
</tr>
<tr>
<td>11</td>
<td>BatteryIndicator</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>BestNeighborId</td>
<td>Best network neighbour</td>
</tr>
<tr>
<td>13</td>
<td>BestNeighborEtx</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>RtMetric</td>
<td>Equivalent to node rank</td>
</tr>
<tr>
<td>15</td>
<td>NumNeighbors</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>BeaconInterval</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Rssi</td>
<td></td>
</tr>
</tbody>
</table>

Table A4.3 NodeParam Table

<table>
<thead>
<tr>
<th>id</th>
<th>sensorId</th>
<th>systemTime</th>
<th>paramId</th>
<th>sensorValue</th>
<th>unitId</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1332845491</td>
<td>4</td>
<td>39.821</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1332845491</td>
<td>3</td>
<td>21.39</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1332845491</td>
<td>1</td>
<td>194.285714</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1332845491</td>
<td>2</td>
<td>170</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>1332845493</td>
<td>4</td>
<td>40.631</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>1332845493</td>
<td>3</td>
<td>21.48</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>1332845493</td>
<td>1</td>
<td>184.285714</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>1332845493</td>
<td>2</td>
<td>175.714285</td>
<td>1</td>
</tr>
</tbody>
</table>

Table A4.4 sensorReading Table

<table>
<thead>
<tr>
<th>id</th>
<th>nodeId</th>
<th>systemTime</th>
<th>paramId</th>
<th>paramValue</th>
<th>unitId</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>133285438</td>
<td>1</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>133285438</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>133285438</td>
<td>11</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>133285438</td>
<td>14</td>
<td>511</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>133285438</td>
<td>15</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>133285438</td>
<td>16</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>133285438</td>
<td>17</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>133285438</td>
<td>4</td>
<td>0.09069551</td>
<td>5</td>
</tr>
</tbody>
</table>

Table A4.5 nodeReading Table
A5.1 Data Collection application requirements specifications

The following requirements can be attributed to the data collection application:

- Data collection must take place through a USB port configured as a serial (COM) port.
- Commands must be sent to the “sink” node for further distribution to the network sensor nodes if so required.
• Command string reliability must be improved by automatic re-sending commands for a predefined number of times if communication errors occur.
• Sets of frequently used commands must be predefined to be readily available.
• Commands must be recognised irrespective of having a local or remote source.
• Returned data strings must be tested for validity. Invalid strings must be silently discarded.
• Valid data strings must be split into a number of clearly identified variables. Complex variables may consist of constants and previously read simple variables.
• All variables must be stored into a properly normalised database for later reference.
• Periodical data collection must take place once every minute for each sensor node.
• Single burst (manual) data collection must take place once every 10 seconds for each sensor node.
• All predefined parameters must be stored in an XML configuration file to enable easy reference and changes.

A5.2 Data collection application process and product quality measures

Process quality can be improved by laying down a number of tasks to be executed in the specified order in the development process:

• The writing of a vision statement. It is important to occasionally remind oneself of the overall goal of the project and to provide a high-level view on the eHome development project.
• The development of use cases ensures that user interaction with the system is orderly taken care of.
• System Requirements Specifications (SRS) must be solidly based on the previously established system architecture, vision statement and use cases. Only then does the process provide some assurance that the SRS reflects the real objective of the product and the needs of its users.
• As part of the preliminary design, an entity-relationship diagram should be drawn up for the database part of the product. With its normalised structure it ensures a database without redundancy and its associated unpleasant maintenance problems.
• Process quality also means adhering to timelines. A timing schedule must therefore be available at the earliest possible moment and kept updated. See annex A3.
• Use a development framework such as Visual Studio together with an object-oriented language to keep unrelated functionality separated and thus improve code quality and future maintenance.
Frequently implement testing functions. Early testing ensures early finding of coding defects and so improves quality of the completed system.

Re-factor the code and integrate and build frequently to catch errors as early as possible.

Post-mortem analysis may not increase quality for this project but it will be very helpful to increase quality of the process used (now and in the future).

Product quality can be improved by a number of measures during the development cycle such as:

- After building the project’s database it must be thoroughly tested using SQL insert, update, delete, and select statements. Particular attention should be given to cascading effects while updating or deleting records.
- Repeat the database tests using the application’s database access routines.
- Compare database fields to text displayed by the application’s interface for consistency.

A5.3 eHome website requirements specifications

The following requirements can be attributed to the eHome website:

- Access to the eHome website is only given to logged-in guests, users or administrators.
- Guests will only be given access to the Home Monitoring pages.
- Users will only be granted read-only access to the Maintain Database section whereas administrators can access all functions and pages of the site.
- The home page will show a set of icons to allow a choice of the task areas.
- The Home Monitoring page will show a series of icons that allow selection of the different sensor locations.
- Each location will show a table of current sensor values including their description and unit of measurement.
- A link will provide access to a page showing a chart of historical sensor data over time. The time period is selectable by calendar form. Furthermore, as there are many parameters with widely spread values, a dropdown list will allow selection of a particular set of associated values such as those from the light sensors.
- Each location can also show a table of node parameters or a chart of historical node parameters over time. The default time period is the most current 24 hours but the time period can also be selected through a calendar form. A selection of different sets of related node parameter values is also available.
• Statistical data of each sensor and node graph will be available in a text box.
• A logged-in admin can control the serial data collection process through selection of a link associated with a certain data communication action such as opening or closing the serial data collection, rebooting the network nodes, or sending a command. A feedback message will show the success or failure of the action undertaken.
• A CMS structure is included to allow create/read/update/delete (CRUD) actions on the database tables associated with the website. Non-reading actions are restricted to logged-in administrators.
• Web services will be used whenever possible and appropriate. Tables and charts, for example, will be shown by making use of Google Charts. Most user accessible pages will also show a small weather forecast icon that depicts weather at the home location for the next 2 hours.

A5.4 eHome website process and product quality measures

Process quality can be improved by laying down a number of tasks to be executed in the specified order in the development process. As many of these tasks were already listed in section A5.2, they are not repeated here. Some tasks, however, need to be added.

• Use a development framework such as Zend with its inherent MVC design to keep unrelated functionality separated and thus improve code quality and future maintenance.
• Frequently implement testing functions. Early testing ensures early finding of coding defects and so improves quality of the completed system.

Product quality can be improved by a number of measures during the development cycle such as (formerly listed items are not repeated here):

• After doing the initial build of the web pages, test them for compatibility with FireFox and IE as well as functioning of the hyperlink navigation. This avoids later problems during coding and improves quality by ensuring an intuitive user interface.
• Compare database fields to content fields on the web pages for consistency.
• Input data to the database through the CMS web interface. Also edit and delete data.
A6. Attached files

Database design files:

Home ER_Diagram.vsd
SQL DDL Home.sql
Test Inserts Home.sql
Test Selects Home.sql

Simulator files:

Data-Collect-40-4.png
Data-Collect-40-8.png
Data-Collect-40-16.png
Data-Collect-40-of0.png
Power-25-16.png
Power-40-4.png
Power-40-8.png
Power-40-16.png
Power-40-of0.png

Modified RPL routing files:

rpl-of-etx.c
rpl-conf.h
udp-router.c
udp-sender.c

Modified web service file:

rest-server.c

Miscellaneous files:

Using JavaScript within a Zend framework website
Zend dynamic drop-down boxes with JavaScript
Project extended proposal (EPP)
Interim project report (IPP)
All image files
### Listing A7.1  Node info chart at RDC of 4 Hz.
Listing A7.2  Node info chart at RDC of 8 Hz.
Listing A7.3  Node info chart at RDC of 16 Hz.
Listing A7.4  Node info chart using OF0 (8 Hz).

A8. Listings of modified Contiki files

File: udp-router.c

/*
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the Institute nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 */
* without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND
CONTRIBUTORS ''AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR
CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF
* SUCH DAMAGE.
*
* This file is part of the Contiki operating system.
*
*/

/* Modified by J. de Hoog to allow preference of mains-powered nodes over
battery-powered ones. */

#include "contiki.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "net/uip-udp-packet.h"
#include "net/neighbor-info.h"
#include "net/rpl/rpl.h"
#include "dev/serial-line.h"
#include "net/netstack.h"
#if CONTIKI_TARGET_Z1
#include "dev/uart0.h"
#else
#include "dev/uart1.h"
#endif
#include "collect-common.h"
#include "collect-view.h"

#include <stdio.h>
#include <string.h>
#define UDP_CLIENT_PORT 8775
#define UDP_SERVER_PORT 5688

#define DEBUG DEBUG_PRINT
#include "net/uip-debug.h"

/*-------------------------------------------------
-------------------------------------------
-------------
Added code to show type of node */
#define ENERGY_TYPE 0 // this is a mains-powered router mote
extern uint8_t energy = ENERGY_TYPE;
/*-------------------------------------------------
-------------------------------------------
-------------
*/

static struct uip_udp_conn *client_conn;
static uip_ipaddr_t server_ipaddr;

/*-------------------------------------------------
--------------------------
-----
Unmodified code
-----
*/
PROCESS_THREAD(udp_client_process, ev, data)
{
    PROCESS_BEGIN();

    PROCESS_PAUSE();

    set_global_address();

    PRINTF("UDP client process started\n");

    print_local_addresses();

    /* The routers can run with a 100% duty cycle in order to ensure high
    packet reception rates. */
    NETSTACK_RDC.off(1);
    /*
    */

    /* new connection with remote host */
client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);
udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));

PRINTF("Created a connection with the server ");
PRINT6ADDR(&client_conn->ripaddr);
PRINTF(" local/remote port %u/%u\n",
      UIP_HTONS(client_conn->lport), UIP_HTONS(client_conn->rport));

while(1) {
    PROCESS_YIELD();
    if(ev == tcpip_event) {
      tcpip_handler();
    }
}

PROCESS_END();

/*-------------------------------------------------
--------------------------*/

File: udp-sender.c

/*
   * Redistribution and use in source and binary forms, with or without
   * modification, are permitted provided that the following conditions
   * are met:
   * 1. Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
   * 2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.
   * 3. Neither the name of the Institute nor the names of its contributors
      may be used to endorse or promote products derived from this software
      without specific prior written permission.
   *
   * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND
   * CONTRIBUTORS ``AS IS'' AND
   * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
   * LIMITED TO, THE
   * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
   * FOR A PARTICULAR PURPOSE
   * ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR
   * CONTRIBUTORS BE LIABLE
   * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
   * EXEMPLARY, OR CONSEQUENTIAL
   * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
   * OF SUBSTITUTE GOODS
   * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
   * INTERRUPTION)
   */
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* This file is part of the Contiki operating system.
*
*/

/* Modified by J. de Hoog to allow preference of mains-powered nodes over
battery-powered ones. */

#include "contiki.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "net/uip-udp-packet.h"
#include "net/neighbor-info.h"
#include "net/rpl/rpl.h"
#include "dev/serial-line.h"
#include "collect-common.h"
#include "collect-view.h"
#include <stdio.h>
#include <string.h>

#define UDP_CLIENT_PORT 8775
#define UDP_SERVER_PORT 5688

#define DEBUG DEBUG_PRINT
#include "net/uip-debug.h"

/*-------------------------------------------------
-----------------------------------------*/
/* Added code to show type of node */
#define ENERGY_TYPE 1  // this is a battery-powered mote
extern uint8_t energy = ENERGY_TYPE;
/*-------------------------------------------------
--------------------------*/

static struct uip_udp_conn *client_conn;
static uip_ipaddr_t server_ipaddr;

/*-------------------------------------------------
-----------------------------------------*/
PROCESS(udp_client_process, "UDP client process");
AUTOSTART_PROCESSES(&udp_client_process, &collect_common_process);
/*-------------------------------------------*/

-----
Unmodified code
-----

PROCESS_END();
}
/*-------------------------------------------*/

File: rpl-conf.h

/*
 * Copyright (c) 2010, Swedish Institute of Computer Science.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the Institute nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF
* SUCH DAMAGE.
*
* This file is part of the Contiki operating system.
*
* \file
* Public configuration and API declarations for ContikiRPL.
* \author
* Joakim Eriksson <joakime@sics.se> & Nicolas Tsiftes <nvt@sics.se>
*
* */

/* Modified by J. de Hoog to allow preference of mains-powered nodes over
battery-powered ones. */

#ifndef RPL_CONF_H
#define RPL_CONF_H

/* Set to 1 to enable RPL statistics */
#ifndef RPL_CONF_STATS
#define RPL_CONF_STATS 0
#endif /* RPL_CONF_STATS */

/
* Select routing metric supported at runtime. This must be a valid
* DAG Metric Container Object Type (see below). Currently, we only
* support RPL_DAG_MC_ETX and RPL_DAG_MC_ENERGY.
* /
#endif RPL_CONF_DAG_MC
#define RPL_DAG_MC RPL_CONF_DAG_MC
#endif RPL_CONF_DAG_MC

Unmodified code

-----

File: rpl-of-etx.c

/**
 * \addtogroup uip6

69
*/
 */
* Copyright (c) 2010, Swedish Institute of Computer Science.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the above copyright
* notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in the
* documentation and/or other materials provided with the distribution.
* 3. Neither the name of the Institute nor the names of its contributors
* may be used to endorse or promote products derived from this software
* without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND
* CONTRIBUTORS "AS IS" AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
* FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR
* CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
* OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
* WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF
* SUCH DAMAGE.
*
* This file is part of the Contiki operating system.
*
*/
/**
* \file
* The minrank-hysteresis objective function (OCP 1).
*
* This implementation uses the estimated number of
* transmissions (ETX) as the additive routing metric.
*/
/* Modified by J. de Hoog to allow preference of mains-powered nodes over battery-powered ones. */

#include "net/rpl/rpl-private.h"
#include "net/neighbor-info.h"

#define DEBUG DEBUG_NONE
#include "net/uip-debug.h"

uint8_t energy;

static void reset(rpl_dag_t *);
static void parent_state_callback(rpl_parent_t *, int, int);
static rpl_parent_t *best_parent(rpl_parent_t *, rpl_parent_t *);
static rpl_dag_t *best_dag(rpl_dag_t *, rpl_dag_t *);
static rpl_rank_t calculate_rank(rpl_parent_t *, rpl_rank_t);
static void update_metric_container(rpl_instance_t *);

rpl_of_t rpl_of_etx = {
    reset,
    parent_state_callback,
    best_parent,
    best_dag,
    calculate_rank,
    update_metric_container,
    1
};

-----
Unmodified code
-----

static rpl_parent_t *
best_parent(rpl_parent_t *p1, rpl_parent_t *p2)
{
    rpl_dag_t *dag;
    rpl_path_metric_t min_diff;
    rpl_path_metric_t p1_metric;
    rpl_path_metric_t p2_metric;

    dag = p1->dag; /* Both parents must be in the same DAG. */

    /*--------------
     // Added code to prefer mains-powered routers
     uint8_t flags, p1_energy, p2_energy;
     */
flags = dag->instance->mc.obj.energy.flags >> RPL_DAG_MC_ENERGY_TYPE;
p1_energy = p1->mc.obj.energy.flags >> RPL_DAG_MC_ENERGY_TYPE;
p2_energy = p2->mc.obj.energy.flags >> RPL_DAG_MC_ENERGY_TYPE;
PRINTF("RPL: FLAGS = %u\n", flags);

if(p1_energy == 0 && p2_energy == 1) {
    // p1 is a router; p2 is a battery node
    PRINTF("RPL: P1 FLAGS = %u P2 FLAGS = %u P1 preferred.\n", p1_energy, p2_energy);
    return p1; }
if(p1_energy == 1 && p2_energy == 0) {
    // p2 is a router; p1 is a battery node
    PRINTF("RPL: P1 FLAGS = %u P2 FLAGS = %u P2 preferred.\n", p1_energy, p2_energy);
    return p2; }
if(p1_energy == 0 && p2_energy == 0) {
    // both nodes are mains-powered
    if(p1->rank == ROOT_RANK(p1->dag->instance)) {
        PRINTF("RPL: P1 FLAGS = %u P2 FLAGS = %u P1 is sink.\n", p1_energy, p2_energy);
        // p1 is the sink
        return p1; }
    else if(p2->rank == ROOT_RANK(p2->dag->instance)) {
        // p2 is the sink
        PRINTF("RPL: P1 FLAGS = %u P2 FLAGS = %u P2 is sink.\n", p1_energy, p2_energy);
        return p2; }
}
/*------------------------------------------------*/
min_diff = RPL_DAG_MC_ETX_DIVISOR / PARENT_SWITCH_THRESHOLD_DIV;
p1_metric = calculate_path_metric(p1);
p2_metric = calculate_path_metric(p2);

/* Maintain stability of the preferred parent in case of similar ranks. */
if(p1 == dag->preferred_parent || p2 == dag->preferred_parent) {
    if(p1_metric < p2_metric + min_diff &&
        p1_metric > p2_metric - min_diff) {
        PRINTF("RPL: MRHOF hysteresis: %u <= %u <= %u\n", p2_metric - min_diff,
            p1_metric,
            p2_metric + min_diff);
        return dag->preferred_parent;
    }
}
return p1_metric < p2_metric ? p1 : p2;

}

static void
update_metric_container(rpl_instance_t *instance)
{
    rpl_path_metric_t path_metric;
    rpl_dag_t *dag;

#if RPL_DAG_MC == RPL_DAG_MC_ENERGY
    uint8_t type;
#endif

    instance->mc.flags = RPL_DAG_MC_FLAG_P;
    instance->mc.aggr = RPL_DAG_MC_AGGR_ADDITIVE;
    instance->mc.prec = 0;

    dag = instance->current_dag;

    if (!dag->joined) {
        /* We should probably do something here */
        return;
    }

    if (dag->rank == ROOT_RANK(instance)) {
        path_metric = 0;
    } else {
        path_metric = calculate_path_metric(dag->preferred_parent);
    }

#if RPL_DAG_MC == RPL_DAG_MC_ETX
    instance->mc.type = RPL_DAG_MC_ETX;
    instance->mc.length = sizeof(instance->mc.obj.etx);
    instance->mc.obj.etx = path_metric;
    PRINTF("RPL: My path ETX to the root is %u.%u\n",
           instance->mc.obj.etx / RPL_DAG_MC_ETX_DIVISOR,
           (instance->mc.obj.etx % RPL_DAG_MC_ETX_DIVISOR * 100) / RPL_DAG_MC_ETX_DIVISOR);
#endif

#elif RPL_DAG_MC == RPL_DAG_MC_ENERGY
    instance->mc.type = RPL_DAG_MC_ENERGY;
    instance->mc.length = sizeof(instance->mc.obj.energy);

    /*-----------------------------------------------
    -------------------------------*/
    /* changed to include mains-powered routers (energy == 0) */
}
if((dag->rank == ROOT_RANK(instance)) || (energy == 0)) {
    type = RPL_DAG_MC_ENERGY_TYPE_MAINS;
} else {
    type = RPL_DAG_MC_ENERGY_TYPE_BATTERY;
}

/*-------------------------------------------------''--------------------------------*/

instance->mc.obj.energy.flags = type << RPL_DAG_MC_ENERGY_TYPE;
instance->mc.obj.energy.energy_est = path_metric;

#else
#error "Unsupported RPL_DAG_MC configured. See rpl.h."
#endif /* RPL_DAG_MC */

File: rest-server.c

/**
 * \file
 *    Light and temperature sensor web demo
 * \author
 *    Niclas Finne   <nfi@sics.se>
 *    Joakim Eriksson <joakime@sics.se>
 *    Joel Hoglund   <joel@sics.se>
 */

/** modified by J. de Hoog */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "contiki-net.h"
#include "rest.h"

#if defined (PLATFORM_HAS_LIGHT)
#include "dev/light-sensor.h"
#endif
#if defined (PLATFORM_HAS_BATTERY)
#include "dev/battery-sensor.h"
#endif
#if defined (PLATFORM_HAS_SHT11)
#include "dev/sht11-sensor.h"
#endif

#define DEBUG 1
#if DEBUG
#include <stdio.h>
#define PRINTF(...) printf(__VA_ARGS__)
#define PRINT6ADDR(addr) PRINTF("%02x%02x:%02x%02x:%02x%02x:%02x%02x
%02x%
%02
%02x", ((uint8_t *)addr)[0], ((uint8_t *)addr)[1], ((uint8_t *)addr)[2], ((uint8_t *)addr)[3], ((uint8_t *)addr)[4], ((uint8_t *)addr)[5], ((uint8_t *)addr)[6], ((uint8_t *)addr)[7], ((uint8_t *)addr)[8], ((uint8_t *)addr)[9], ((uint8_t *)addr)[10], ((uint8_t *)addr)[11], ((uint8_t *)addr)[12], ((uint8_t *)addr)[13], ((uint8_t *)addr)[14], ((uint8_t *)addr)[15])
#define PRINTLLADDR(lladdr) PRINTF("%02x:%02x:%02x:%02x:%02x:%02x", (lladdr)->addr[0], (lladdr)->addr[1], (lladdr)->addr[2], (lladdr)->addr[3], (lladdr)->addr[4], (lladdr)->addr[5])
#else
#define PRINTF(...)
#define PRINT6ADDR(addr)
#define PRINTLLADDR(addr)
#endif
char buf[100];

/* Resources are defined by RESOURCE macro, signature: resource name, the http methods it handles and its url.
 * For each resource defined, there corresponds an handler method which should be defined too.
 * Name of the handler method should be [resource name]_handler.
 */
#if defined (PLATFORM_HAS_LIGHT)
/* Returns the reading from light sensor1 */
uint16_t light_photosynthetic;
RESOURCE(light1, METHOD_GET, "light1");

void read_light_sensor1(uint16_t* light_1)
{
    *light_1 = 10 * light_sensor.value(LIGHT_SENSOR_PHOTOSYNTHETIC) / 7;
}

void light1_handler(REQUEST* request, RESPONSE* response)
{
    read_light_sensor1(&light_photosynthetic);
    printf(buf,"%u\n", light_photosynthetic);

    rest_set_header_content_type(response, TEXT_PLAIN);
    rest_set_response_payload(response, (uint8_t*)buf, strlen(buf));
}
#endif

/* Returns the reading from light sensor2 */

uint16_t light_solar;

RESOURCE(light2, METHOD_GET, "light2");

void
read_light_sensor2(uint16_t* light_2)
{
    *light_2 = light_sensor.value(LIGHT_SENSOR_TOTAL_SOLAR);
}

void
light2_handler(REQUEST* request, RESPONSE* response)
{
    read_light_sensor2(&light_solar);
    sprintf(buf, "%u\n", light_solar);

    rest_set_header_content_type(response, TEXT_PLAIN);
    rest_set_response_payload(response, (uint8_t*)buf, strlen(buf));
}

#endif /*PLATFORM_HAS_LIGHT*/

#if defined (PLATFORM_HAS_SHT11)

/* Returns the reading from the temperature sensor */

uint16_t temperature;

RESOURCE(temp, METHOD_GET, "temp");

void
read_temp_sensor(uint16_t* temp)
{
    *temp = sht11_sensor.value(SHT11_SENSOR_TEMP);
}

void
temp_handler(REQUEST* request, RESPONSE* response)
{
    read_temp_sensor(&temperature);
    sprintf(buf, "%u\n", temperature);

    rest_set_header_content_type(response, TEXT_PLAIN);
    rest_set_response_payload(response, (uint8_t*)buf, strlen(buf));
}

/* Returns the reading from the humidity sensor */
uint16_t humidity;

RESOURCE(hum, METHOD_GET, "hum");

void read_hum_sensor(uint16_t* humidity)
{
    *humidity = sht11_sensor.value(SHT11_SENSOR_HUMIDITY);
}

void hum_handler(REQUEST* request, RESPONSE* response)
{
    read_hum_sensor(&humidity);
    sprintf(buf, "%u\n", humidity);

    rest_set_header_content_type(response, TEXT_PLAIN);
    rest_set_response_payload(response, (uint8_t*)buf, strlen(buf));
}

#endif /*PLATFORM_HAS_SHT11*/

#if defined (PLATFORM_HAS_BATTERY)

/* Returns the reading from the battery sensor */

uint16_t battery;

RESOURCE(batt, METHOD_GET, "batt");

void read_batt_sensor(uint16_t* batt)
{
    *batt = (uint16_t)battery_sensor.value(0);
}

void batt_handler(REQUEST* request, RESPONSE* response)
{
    read_batt_sensor(&battery);
    sprintf(buf, "%u\n", battery);

    rest_set_header_content_type(response, TEXT_PLAIN);
    rest_set_response_payload(response, (uint8_t*)buf, strlen(buf));
}

#endif /*PLATFORM_HAS_BATTERY*/
PROCESS(rest_server, "Rest Server");
AUTOSTART_PROCESSES(&rest_server);

PROCESS_THREAD(rest_server, ev, data)
{
    PROCESS_BEGIN();

    #ifdef WITH_COAP
    PRINTF("COAP Server\n");
    #else
    PRINTF("HTTP Server\n");
    #endif

    rest_init();

    #if defined (PLATFORM_HAS_LIGHT)
    SENSORS_ACTIVATE(light_sensor);
    rest_activate_resource(&resource_light1);
    rest_activate_resource(&resource_light2);
    #endif

    #if defined (PLATFORM_HAS_SHT11)
    SENSORS_ACTIVATE(sht11_sensor);
    rest_activate_resource(&resource_temp);
    rest_activate_resource(&resource_hum);
    #endif

    #if defined (PLATFORM_HAS_BATTERY)
    SENSORS_ACTIVATE(battery_sensor);
    rest_activate_resource(&resource_batt);
    #endif

    PROCESS_END();
}