Design and Implementation of Internet of Things for Home Environment

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Engineering in

Electronics and Computer Systems Engineering

at Massey University, Manawatu,

New Zealand.

Sean Kelly

2013

Abstract

An integrated framework for smart home monitoring towards internet of things based on ZigBee and 6LoWPAN wireless sensor networks is presented. The system was developed to retrofit existing sub systems of wireless technologies in order to reduce cost, and complexity. The practical internetworking architecture and the connection procedures for reliable measurement of smart sensors parameters and transmission of sensing data via internet are presented. A ZigBee based sensing system was designed and developed to see the feasibility of the system in home automation for contextual environmental monitoring. The ubiquitous sensing system is based on combination of pervasive distributed sensing units and an information system for data aggregation and analysis. Results related to the home automation parameters and execution of the system running continuously for long durations is encouraging. The prototype system (ZigBee based) was tested to generate real-time graphical information rather than using a simulator or a test bed scenario. A trail has also been performed with 6LoWPAN technology to provide functionality as the ZigBee based system.

The overall internetworking architecture describes the integration of a low power consumption wireless sensor network with the internet. The proposed prototype has advantages in terms of low cost, flexibility of usage. The design of the integrated framework provides a template for other applications related to the Internet of Things.

Acknowledgements

I would like to thank my family for supporting me during my course of study.

I would like to thank my supervisor Prof. Subhas Mukhopadhyay for his patience and support.

I would like to thank Nagender Suryadevara for his technical help and support.

Abstract i

Acknowledgements ii

List of Figures viii

List of Tables xiii

1 Introduction 1

1.1 Outline of Thesis 1

2 Literature Review 3

2.1 Wireless Sensor Networks integrated with Internet of Things 3

2.2 Existing Internet of Things Systems 4

2.3 Home Automation through Internet of Things 5

2.3.1 Existing Wireless Technologies used for Home Automation 6

2.3.2 Existing Wireless Sensor Network Architectures for Home Automation 6

3 Contribution to the topic 11

4 System Structure 12

4.1 Introduction 12

4.1.1 Integrated Platform for IoT 13

4.2 Smart sensors requirements 13

4.2.1 Signal conditioning 14

4.2.2 Signal Input and Output Processing 15

4.2.3 Network communication 15

4.3 XBee Structure 16

4.3.1 XBee Stack Specifications 16

4.3.2 XBee Topologies 17

4.3.3 XBee Hardware Specification 18

4.3.4 IoT adaptation 19

4.4 6LoWPAN Structure 19

4.4.1 6LoWPAN Stack Requirements 20

4.4.2 6LoWPAN Topologies 21

4.4.3 6LoWPAN Hardware Specifications 21
4.4.4 6LoWPAN Hardware Options 22
4.5 IoT Gateway 22
4.5.1 Gateway Hardware 22
4.5.2 WSN Interface 23
4.5.3 IoT communication 23
4.6 IoT Server 23
4.6.1 Minimum requirements 24
4.6.2 Network connectivity 24
4.6.3 Data Reception, Storage and Web server 24
4.7 Summary 24
5 Implementation 26
5.1 Software Development Environment 26
5.1.1 Code Composer Studio Setup – 6LoWPAN Module 26
5.1.1.1 Obtaining and installing Code Composer Studio 26
5.1.1.2 Project creation for CC430 26
5.1.1.3 Debugging a CC430 Project 28
5.1.2 OpenWRT Toolchain with Eclipse IDE 31
5.1.2.1 Windows OpenWRT Toolchain Setup 32
5.1.2.1.1 Cygwin Installation 32
5.1.2.1.2 Windows configuration 34
5.1.2.1.3 OpenWRT tool chain compilation 36
5.1.2.2 Eclipse 38
5.1.2.2.1 Required directories from the OpenWRT toolchain 38
5.1.2.2.2 Eclipse Setup for Windows/Linux 38
5.1.2.2.3 Eclipse Project Creation 39
5.1.2.2.4 Eclipse Remote Debugging 42
5.1.3 Visual Studio Setup 51

5.1.3.1 Installation 51
5.1.3.2 Project Creation 51
5.1.3.3 Project Debugging 54
5.2 Software Configuration 55
5.2.1 Windows Apache MySQL and PHP (WAMP) 55
5.2.1.1 WAMP Installation 55
5.2.1.2 WAMP Configuration 55
5.2.2 Open source embedded Linux installation 57
5.2.2.1 OpenWRT Compilation 57
5.2.2.2 OpenWRT installation 59
5.2.3 OpenVPN 63
5.2.3.1 OpenVPN Installation 63
5.2.3.2 OpenVPN Certificate and Key Generation 64
5.2.3.3 OpenVPN Server Configuration 68
5.2.3.4 OpenVPN Client Configuration 71
5.2.4 XBee Configuration 74
5.2.4.1 XBee Configuration Software 75
5.2.4.2 XBee Coordinator Configuration 76
5.2.4.3 XBee End Device Configuration 79
5.3 Network Interface Implementation Techniques 82
5.3.1 Tunnel Driver Interface 82
5.3.2 Parsing raw data from buffers 83
5.3.2.1 Host and network byte order 83
5.3.2.2 XMacro technique 85
5.3.2.3 XStruct technique 87
5.4 ZigBee IoT Platform Implementation 91
5.4.1 ZigBee Hardware Implementation 91
5.4.1.1 XBee Gateway 92

5.4.1.2 XBee Temperature Sensor Module 96
5.4.1.3 XBee Hot Water System Monitor 98
5.4.2 Address Translation 109
5.4.3 IPv6 UDP Encapsulation 110
5.4.4 Software implementation on IoT Gateway for XBee 114
5.4.4.1 Initialise Serial and Tunnel 115
5.4.4.2 Select from Tunnel and Serial 116
5.4.4.3 Processing Tunnel Data 118
5.4.4.4 Serial Has Data 120
5.4.5 ZigBee IoT Platform Summary 125
5.5 6LoWPAN IoT Platform Implementation 125
5.5.1 6LoWPAN Development Hardware 126
5.5.1.1 6LoWPAN Node 126
5.5.1.2 6LoWPAN Gateway 127
5.5.2 6LoWPAN Node Firmware Implementation 129
5.5.2.1 6LoWPAN stack 129
5.5.2.2 Microcontroller Radio Interface for 6LoWPAN stack 131
5.5.2.3 Application to acquire and send sensor data using 6LoWPAN 135
5.5.3 6LoWPAN Gateway Implementation 135
5.5.3.1 Radio interface for 6LoWPAN gateway 135
5.5.3.2 Software for 6LoWPAN Gateway 136
5.6 IoT Server Implementation 136
5.6.1 Sensor Data Acquisition and Storage 136
5.6.1.1 Receiving Sensor Samples 136
5.6.1.2 Storing Sensor Samples 137
5.6.2 Graphical Web Interface 137
5.6.2.1 Retrieving Sensor Samples from Database 137
5.6.2.2 Plotting Retrieved Sensor Samples in Graph 138

5.6.2.3 Controlling WSN Nodes 138
5.7 Summary 138
6 Experimental Results 140
6.1 Fabricated Sensor Modules for monitoring the Smart Home 140
6.2 Efficient mechanism for sensor data storage 145
6.3 Quality of Service parameters for XBee IoT Platform 146
6.3.1 Reliability 146
6.3.2 Throughput 147
6.3.3 Jitter 149
7 Conclusion 150
8 Challenges and Opportunities 151
8.1 Future Works 151
8.1.1 ZigBee based IoT Platform Future Works 151
8.1.2 6LoWPAN based IoT Platform Future Works 151
9 References 153
10 Publications 164

List of Figures

Figure 1 Topology and stack based approches 7
Figure 2 Structure of integrated IoT platform 13
Figure 3 ZigBee start topology 17
Figure 4 ZigBee Mesh topology 17
Figure 5 XBee Module pin numbering (top view) 18
Figure 6 IPv6 and 6LoWPAN stact operation to give Server application to Sensor Application communication 20
Figure 7 6LoWPAN and IPv6 network topology 21
Figure 8 Code Composer Studio - Create new project. 26
Figure 9 Code Composer Studio - Create new project options for “Hello World” project. 27
Figure 10 Code Composer Studio - "Hello World" project contents. 28
Figure 11 Code Composer Studio - Debug button to start debug process. 28
Figure 12 Code Composer Studio - Loading "Hello World" project onto microcontroller. 29
Figure 13 Code Composer Studio - No debugging hardware connected to the computer Error. 29
Figure 14 Code Composer Studio - Debugging hardware failing to communicate with microcontroller Error 29
Figure 15 Code Composer Studio - Microprocessor variant incorrectly set for current microprocessor connected Error. 30
Figure 16 Code Composer Studio - Procedure for setting the microprocessor variant for "Hello World" project. 30
Figure 17 Code Composer Studio state after sucessfully starting a debug session. 31
Figure 18 Selecting a Cygwin package to install 33
Figure 19 Selecting a Cygwin package to install with source code 33
Figure 20 Selecting the libncurses version 5.7-16 Cygwin package for installation 34
Figure 21 Editing registry to make windows case sensitive. 35
Figure 22 Adding Cygwin bin directory to the path 35
Figure 23 Adding CYGWIN environment variable to suppress warnings 35
Figure 24 Selecting build options for OpenWRT. 37
Figure 25 Eclipse - packages required for remote debug of OpenWRT software. 39
Figure 26 Eclipse - new project dialog setting the project name and type 40
Figure 27 Eclipse - selecting "Paths and Symbols" in project properties to configure header and library paths. 40
Figure 28 Eclipse - setting the include path for OpenWRT toolchain. 41

Figure 29 Eclipse - setting the library path for the OpenWRT toolchain. 41
Figure 30 Eclipse - setting the cross-compiler prefix and path for the OpenWRT toolchain. 41
Figure 31 Creating a new source file main.c 42
Figure 32 Eclipse - output of a successful build of the project "Hello World". 42
Figure 33 Eclipse - displaying Remote Systems view 43
Figure 34 Eclipse - selecting the remote resource type "Linux" for the OpenWRT router. 44
Figure 35 Eclipse - setting the Host name and Connection name for a remote resource. 44
Figure 36 Eclipse - configuring remote resource file access. 44
Figure 37 Eclipse - configuring remote resource process access method. 44
Figure 38 Eclipse - configuring remote resource shell access method. 45
Figure 39 Eclipse - configuring remote resource terminal access method. 45
Figure 40 Eclipse - User name and password for connecting to a remote resource (OpenWRT router). 45
Figure 41 Eclipse - sucessful connection to a remote resource "192.168.1.100" which is the OpenWRT router. 46
Figure 42 Eclipse - Menu to show debug configurations. 46
Figure 43 Eclipse - Main remote application debugging configuration for "Hello World" project. 47
Figure 44 Eclipse - Setting Location of GDB produced by OpenWRT toolchain. 48
Figure 45 Eclipse - "Hello World Debug" configuration location. 48
Figure 46 Eclipse - Confirming automatic change to Debug Perspective. 48
Figure 47 Eclipse - Unable to locate source files error. 49
Figure 48 Eclipse - Debug perspective for the debugging of "Hello World" project. 50
Figure 49 Eclipse - Program output in a debugging session. 50
Figure 50 Visual Studio - Creating a new project. 51
Figure 51 Visual Studio – New project creation dialog for “Hello Word” project. 52
Figure 52 Visual Studio - Opening reference manager for "Hello World" project. 52
Figure 53 Visual Studio - Adding MySQL.Data reference to "Hello World" project. 53
Figure 54 Visual Studio - Adding a breakpoint to the "Hello World" project. 54
Figure 55 Visual Studio - Button for starting debug session. 54
Figure 56 Visual Studio - Debugger stopped at "Hello World" breakpoint. 54
Figure 57 Visual Studio - "Hello World" Executable output. 54
Figure 58 WAMP - "Put Online" to enable access to WAMP from external IP addresses. 55
Figure 59 WAMP - Enabling "php_sockets" from the menu. 56
Figure 60 WAMP - opening a MySQL console session. 56

Figure 61 MySQL - succesfully chaning the root users password. 56
Figure 62 WAMP - phpMyAdmin in Internet Explorer. 57
Figure 63 OpenWRT - Uploading and flashing OpenWRT on WRT54GL router. 60
Figure 64 PuTTY settings for connecting to WRT54GL after OpenWRT installation. 60
Figure 65 OpenWRT - telnet connection to set root password. 61
Figure 66 PuTTY settings for SSH access to the WRT54GL router. 61
Figure 67 PuTTY key mismatch error. 62
Figure 68 OpenWRT - Secure Shell session login. 62
Figure 69 OpenWRT - IP Configuration for joing local network. 63
Figure 70 OpenVPN - Starting a command prompt. 64
Figure 71 OpenVPN - Changing to easy-rsa directory and running init-config. 64
Figure 72 OpenVPN - Running "vars.bat" and "clean-all.bat" to setup the environment. 65
Figure 73 OpenVPN - Running "build-ca.bat" to create the main key and certificate for the OpenVPN server. 65
Figure 74 OpenWRT - Generating server certificate and key. 66
Figure 75 OpenVPN - Generating IoT application gateway certificates and keys. 67
Figure 76 OpenVPN - Generating Diffie Hellman parameters. 67
Figure 77 OpenVPN GUI application on the server. 69
Figure 78 OpenVPN - Sucessfully creating a tunnel service. 69
Figure 79 OpenVPN - Method to show status window. 70
Figure 80 Opening networking and Sharing Center 70
Figure 81 Opening properties for OpenVPN tap driver. 71
Figure 82 Selecting IPv6 properties. 71
Figure 83 Changing IPv6 address for OpenVPN tap driver. 71
Figure 84 OpenVPN - Downloading certificates and keys onto WRT54GL router. 72
Figure 85 OpenVPN - sever status window showing sucessful router connection. 74
Figure 86 X-CTU - XBee module configuration software showing connection settings and module settings. 76
Figure 87 X-CTU - Selecting module type and firmware for coordinator. 76
Figure 88 X-CTU - Setting network parameters for coordinator XBee module. 77
Figure 89 X-CTU - Setting serial connection parameters for coordinator XBee module. 77
Figure 90 X-CTU - upper is programming coordinator XBee module and lower is configuring coordinator XBee module. 78

Figure 91 X-CTU - serial configuration for testing connection to coordinator XBee module for the OpenWRT router. 78
Figure 92 X-CTU – Verifying settings by reading them from coordinator XBee module. 79
Figure 93 X-CTU Setting module type and firmware for End Device. 80
Figure 94 X-CTU Setting PAN ID for End Device. 80
Figure 95 X-CTU Setting Sleep Mode settings for End Deivce 81
Figure 96 X-CTU Setting Input/Output settings for End Device. 82
Figure 97 WRT54GL - serial port pins provided on the board obtained from [73] 92
Figure 98 WRT54GL router with XBee Coordinator attached to serial port using fabricated PCB 93
Figure 99 Schematic and PCB design for hardware interface between XBee S2 coordinator module and WRT54GL board 94
Figure 100 Raspberry PI with fabricated PCB for XBee coordinator to connect to serial port 94
Figure 101 Raspberry PI header pins from [94] 95
Figure 102 Schematic and PCB design for hardware interface between XBee S2 coordinator module and Raspberry PI 96
Figure 103 XBee temperature sensor hardware interface schematic 98
Figure 104 XBee temperature sensor hardware interface PCB 98
Figure 105 Hot water system and solar heating system 100
Figure 106 Graph of temperature Vs Voltage for hot water cylinder probe 101
Figure 107 Schematic for circuit to condition signal from hot water cylinder probe 101
Figure 108 LTSpice simulation results for water cylinder temperature probe voltage signal conditioning 104
Figure 109 Graph of temperature Vs votlage for solar heater probe 105
Figure 110 Schematic for circuit to condition signal from solar heater probe 106
Figure 111 XBee to IPv6 Address translation technique 110
Figure 112 Flow of sample information from XBee end device to Server 112
Figure 113 Converting an XBee S2 API packet to an IPv6 UDP packet 112
Figure 114 Flow of information from Server to XBee Module 113
Figure 115 Converting an IPv6 UDP command packet to an XBee API command packet 113
Figure 116 Flow of information through software and hardware elements between XBee module and Server 114
Figure 117 Flowchart of XBee IoT gateway custom software implementation 115
Figure 118 Flowchart of select process in XBee IoT gateway software 117
Figure 119 Flowchart of reading tunnel data in XBee IoT Gateway software 118

Figure 120 Flowchart for writing an XBee API packet to the serial port 120
Figure 121 Flowchart for reading an XBee API packet form the serial file handle in the XBee IoT gateway software 122
Figure 122 Flowchart for the process to parse different types of XBee API Packets 123
Figure 123 Texas instruments EM430F6137RF900 (left) and Olimex MSP430-CCRF (right) 126
Figure 124 MSP430 Launchpad attached to the Olimex MSP430-CCRF for firmware upload and debugging 127
Figure 125 6LoWPAN gateway using a WRT54GL router connected to a Texas Instruments CC430F6137 development board with MSP430 JTAG debugger attached. 128
Figure 126 6LoWPAN gateway using the Raspberry PI connected to an Olimex CC-RF development board 129
Figure 127 Flow chart for process of transmitting data using CC1101 radio 132
Figure 128 Flowchart for process of receiving a packet using the CC1101 radio 134
Figure 129 Real-time Graph of sensor information for the solar water heater, and hot water cylinder 141
Figure 130 Real-time graph of sensor information on website for light intensity and ambient temperature 142
Figure 131 Real-time graph of sensor information on website for voltage and current useage of electric pump 143
Figure 132 Graph of temperature data from three temperature sensors for 45 days 143
Figure 133 Correlation of sensor information depicting two days typical usage 144
Figure 134 Real-time graph of temperature data from a temperature sensor for 45 days 145
Figure 135 Throughput of an electrical sensing unit for a period of one month 147
Figure 136 Throughput of temperature sensor units 148
Figure 137 Jitter for a sensing unit for a period of one month 149

List of Tables

Table 1 Comparison of various wireless sensor network technologies [47] 9
Table 2 Etherios Device Cloud Services pricing (in NZD) 9
Table 3 ZigBee Stack 16
Table 4 XBee Module pin information 19
Table 5 6LoWPAN Development Hardware 22
Table 6 Memory storage locations of the value 0x12345678 with different endianness 84
Table 7 IPv6 header in network byte order 84
Table 8 IPv6 header structure in memory, or host byte order 85
Table 9 TMP3x parameters from [96] 96
Table 10 Temperature range for TMP3x sensor when using XBee module 97
Table 11 IPv6 and UDP header structure 111
Table 12 CC430 Development Boards Comparison 127
Table 13 6LoWPAN header for sending UDP packet to server 130
Table 14 IEEE 802.15.4 header for 6LoWPAN UDP packet 131
Table 15 Comparison of compressed and uncompressed data for temperature sensors 146
Table 16 Reliability of the data transmission in the integrated ZigBee –IPv6 networks of two sensing units for a period 31 days 147

1 Introduction

Home automation means the monitoring and control of household objects intelligently for effective usage. In order to have an intelligent home monitoring and control system (smart home), the household objects should be intelligently interconnected as well as provide information for better operation. The existing home automation systems consist of embedding household objects with sensors for getting usage information. The interconnection of various household objects have been realized into a sensor network for collective information gathering to perform home automation.

Home automation augmented with the Internet of Things (IoT) provides better flexibility in managing and controlling household objects in a wider aspect. This will support the interconnectivity of a large number of smart homes for better resource utilization in wider area.

The present work describes the integration of home automation with the IoT. The amalgamation of WSN and IoT involves the application of various wireless technologies with internetworking mechanisms.

1.1 Outline of Thesis

Design and development of the integrating the WSN with the IoT for Home automation is organised in the following chapters.

Chapter 2 is the literature review which provides the background information about the integration of WSN and the IoT. Existing IoT integration systems related to Home Automation is also presented. Commercially available systems with their features have also been described.

Chapter 3 highlights the important tasks that have been successfully implemented with the developed prototype.

Chapter 4 describes the system structure of the integrated WSN-IoT platform. It also, specifies the requirements needed to have a compatible integrated networking architecture. The hardware and software components required for the design and development of the WSN-IoT system for Home Automation have been described. It also specifies low power wireless communication technologies to be used for the home automation.

Chapter 5 details the implementation strategy for developing the components of the integrated system. It includes the following:

- Software tools required for configuring and managing various hardware resources and used for controlling the developed sensor modules.

- Configuration of wireless technologies.
- Techniques for internetworking the WSN with the IoT.

Chapter 6 provides details of the experimental evaluation of the integrated WSN-IoT Quality of Service (QoS).

Chapter 7 and 8 describes the conclusion derived from the developed prototype and suggests improvements in the form of future work for an effective utilisation of integration WSN-IoT platform.

2 Literature Review

2.1 Wireless Sensor Networks integrated with Internet of Things

The term Internet of Things (IoT) was initially given by Kevin Ashton [1] in a presentation in 1998. The IoT "allows people and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using any network and any service" [2]. It is expected that 50 to 100 billion devices will be connected to the Internet by 2020 [3]. According to the BCC Research [3], global market for sensors was around $56.3 billion in 2010 [3]. In 2011, it was around $62.8 billion [3]. Global market for sensors is expected to increase up to $91.5 billion by 2016, at a compound annual growth rate of 7.8% [3]. Due to the large number of internet connected devices, the connection, configuration, and management of these devices is not feasible if not done automatically.

The sensor networks are the vital elements of the IoT. Sensor networks have one or more sensor nodes which communicate between each other using wired or wireless means [4]. Each sensor node may have the capabilities to sense, communicate and process data either locally or remotely. In sensor networks, sensor nodes may be homogeneous or heterogeneous. The sensors nodes are installed in densely manner around the phenomenon which we want to sense [4].

With the advancement of software and hardware technologies, the combinations of Wireless Sensor Networks (WSN) and "intelligent" objects of real-world entities, with the Internet capabilities have been a realistic approach. The IoT is the integration of distinctively recognizable Smart Objects, fabricated devices of real semantic objects and their realizations to the Internet. IoT developers work in-conjunction with the hardware of WSNs, but the procedures for installing and interpreting the sensor devices as smart (intelligent) objects are not trivial. Application development for the IoT is related to the usage features of WSNs in particular context. Design and development of IoT systems require addressing the basic issues such as [5]:

- Networking and connectivity issues related to hardware and software heterogeneity,
- application flexibility and scaling,
- standardized communication services and their descriptions,
- Automation procedures,
- Handling big data management.

The IoTs have the capabilities to deploy trillions of low cost wireless sensor nodes supported with internet protocol (IP), this will enable sensor nodes to detect and monitor every object, or entity around the real world [6]. The combination of sensing entities will allow us to interact with the

environment around us easily [6]. Each device connecting to the internet, requires an IP address. The present IPv4 has 32-bit address space (i.e.) around 4.3 billion unique IP addresses, less than the present world population. In order to overcome the limitation of 32-bit address space problem, a new version known as IPv6 is active and playing an important role in the implementation of IoTs. IPv6 is capable of addressing over 340 undecillion addresses (128-bit address space). Hence, the IPv6 is capable of identifying trillions of sensor nodes for WSN [7].

Internet technologies and Wireless Sensor Networks (WSN), a new trend in the era of ubiquity is being realized. Enormous increase in users of Internet and modifications on the internetworking technologies enable networking of everyday objects [8]. “Internet of Things (IoT)” is all about physical items talking to each other, machine-to-machine communications and person-to-computer communications will be extended to “things” [9]. Key technologies that will drive the future IoT will be related to Smart sensor technologies including WSN, Nanotechnology and Miniaturization.

2.2 Existing Internet of Things Systems

In recent times, there has been significant activities in the context of combining WSNs and IoT [10] [11] that lead to a number of trials with enormous experiments and fabrication of systems dedicated on gathering sensing data from a diversity of sources. Global Sensor Network (GSN1) [12] is cluster of data streaming engines. Xively (formerly cosm) [13] is a secure, scalable platform that connects devices and products with applications to provide real-time control and data storage. Xively provides an online database service which allows developers to build their own applications based on the Xively sensor-derived data. Simple Measuring and Actuation Profile sMap [14] is a web service which allows instruments and other producers of physical information to directly publish their data. Its strengths are that it is easy to consume, easy to implement for new device types, and simple to process. SenseWeb [15] is a prototype system which provides a .NET API. Applications using SenseWeb can initiate and access sensor data streams from shared sensors across the entire Internet. SenseWeb allows multiple applications to share concurrent common sensing resources. Sen.se is a platform to not only integrate the IoT but the “Internet of Everything where Humans, Nature, Machines, Objects, Environments, Information, Physical and Virtual spaces all mix up” [16]. It provides services to publish data to feeds and subscribe to feeds. The data in the feeds may be coming from or going to the IoT, and/or people, and/or the internet. Etherios Device Cloud (formerly iDigi Device cloud) allows the connection of any device to any application, anywhere [17]. The platform allows the development of embedded devices that connect with the device cloud, and providing access to the devices in applications using a RESTful interface. Management of the devices in the cloud is also provided.

At present, Commercial WSN frameworks in practise are similar in terms of their hardware architecture [18], which are fabricated using either the 16bit MSP430 microcontroller unit (MCU) or the 8bit ATmega128 MCU and the communication medium using the 802.15.4 Zigbee based network architecture, such as EPIC, IRIS, MicaZ, and Mica2, Sky, Tmote, and TelosB [19]. The popular use of the microcontrollers either MSP430 or ATmega128 is due to several reasons such as: Ultra Low Power energy consumption, community support, open source compilers based on GNU-GCC, and TinyOS support and flexible to design WSN platforms [20].

2.3 Home Automation through Internet of Things

Humans usually inside their home interact with the environment settings like light, air, etc., and regulate accordingly. If the settings of the environment can be made to respond to human behaviour automatically, then there are several advantages. The automation of home settings to act according to the inhabitant requirements is termed as intelligent home automation system. Ambient intelligence responds to the behaviour of inhabitants in home and provides them with various facilities [21].

In general, intelligent home automation system consists of cluster of sensors, collect different types of data, regarding the residents and utility consumption at home. Systems with computing capabilities analyse the assimilated data to recognize the activities of inhabitants or events. These can automate the domestic utilizations effectively and also can support the inhabitant by reducing the costs and improving the standard of living. In the recent past, several research activities were actively involved with IoT such as [22] [23] [24]. Most of the research activities related to IoT are confined to management of resource constraint devices [25], and different mechanisms of interconnection [26] [27].

The future cyber-age networked infrastructures of household appliances in homes are likely to be reliant on sensors embedded in/on the infrastructure. Such technologies will act as a catalyst to the evolution of a new generation services that will have a great impact on the social and technological eco-system. According to [21], it can be envisaged that the next generation systems and services will encompass several domains such as e-Governance, Health Care, Transportation, Waste Management, Food Supply Chains, and Energy & Utilities. New technologies and applications built on top of smart devices may fulfil the vision of Intelligent Infrastructure.

There are several examples of intelligent home automation or “Smart Home Monitoring” in research labs around the world, such as the GatorTech Smart House [28], Casas Smart Home [29], iDorm [30], Georgia Tech Aware Home [31], Place Lab [32], etc. To date, there has been no complete development of a monitoring smart home of commercial perspective, nor any investigation into how such a house is perceived by either the inhabitants or their careers. The smart homes designed so far are for different

purposes such as information collection and decision support system for the wellbeing of the inhabitants [33] [34], storing and retrieving of multimedia data [34] and surveillance, where the data is captured from the environment and processed to obtain information that can help to raise alarms, in order to protect the home and the inhabitants from burglaries, theft and natural disasters [34].

2.3.1 Existing Wireless Technologies used for Home Automation

In order to have low-power consumption, the ZigBee protocol follows the physical and data link layer stack of IEEE 802.15.4. On the other hand, it has limitations on network and application layer functionalities such as addressing, routing and interoperability with the internet. Alternatively, adapting to IPv6 Low Power Personal Area Network (6LoWPAN) protocol, help us to have better end-end communication with the sensing devices. However, translation mechanisms such as SOAP/REST, GRIP [35] will increase the complexity of the network system.

The concurrence impact of wireless sensor network on the IEEE 802.15.4 devices was assessed in [36]. Studies in [37] [37] have been proven theoretically that WSN performance is more perceptible to reduce when interfered with other radio networks and likelihood of faults in 802.15.4 network is high. In [38], the authors have studied the coexistence of IEEE 802.15.4 and other radio networks, based on outage probability, packet loss rate and changes in RSSI value.

Research for internetworking 802.15.4 with IP networks has been conducted. 6LowPAN [39] provides well-defined method for transferring IPv6 packets over 802.15.4 network. However, complexity to deploy in 802.15.4 network nodes is very difficult [40]. IPv6 over 6LoWPAN is proposed by the Internet Engineering Task Force (IETF) working group to accomplish the concept of IP-based WSN. A new layer is incorporated between IPv6 network layer and 802.15.4 MAC layer, which is entitled adaptation layer. It was observed that the adaptation layer, in particular the fragmentation process may increase the energy consumption of a sensor node by 5 to 10 percent [18]. As mentioned above there are many issues related to integrating IPv6 with the WSN.

The advantages of interconnecting WSN with IoT model is for remote monitoring of a contextual environment, where in heterogeneous data will be capable to work together and deliver collective facilities. According to an internetworking perception, a WSN can be fully integrated into the IoT by the kind of integration approach used for both the infrastructures.

2.3.2 Existing Wireless Sensor Network Architectures for Home Automation

The integration approaches of WSN and IoT can be categorized in two different ways: i) stack-based [7] and ii) topology-based [8]. In the first approach, the integration between the IoT and a WSN is contingent on the connections among their network stacks [41]. The second type of integration

approach (topology based) categorization depends on the definite position of the nodes in the environment that can provide access to IoT [42].

The exterior IoT hosts and WSN nodes certainly not communicate openly. The sensor node is entirely autonomous from the IoT. This has its specific customary protocols such as WirelessHART [43]. Interconnections among the external entities and the WSN are accomplished by centralized, coordinator (base station). The base station will collect the data arising from the sensor node, and it can well send data to the external entities through Web Services [15]. Also, requests coming from IoT hosts will come through the central coordinator.

The second approach (topology based interconnecting), reflects the presence of a base- station and perform as application layer gateway. This will interpret the TCP/IP routing information from one socket to alternative socket. The IoT hosts and WSN nodes can interchange info without establishing really a thru association. In this approach, the Wireless sensor nodes are self-regulating from the IoT.

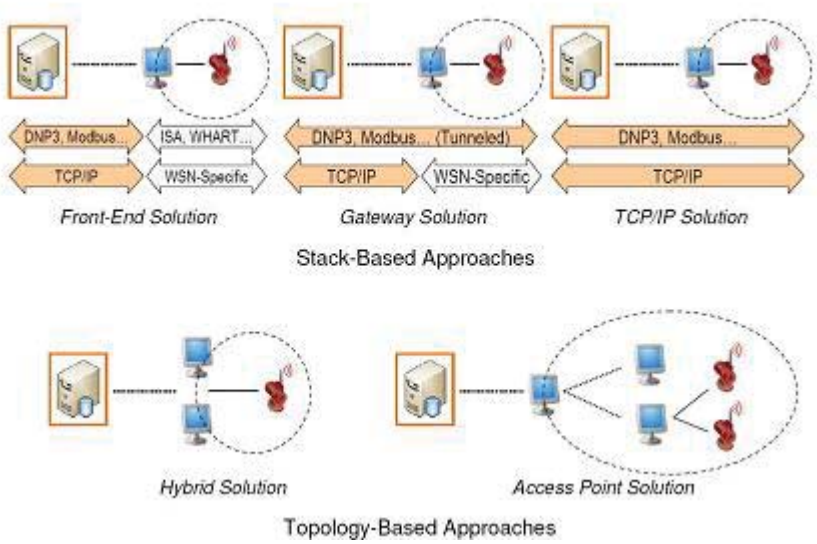

Figure 1 Topology and stack based approaches

In the developed system a third method, i.e. using only IP inter-connection a WSN is integrated with the IoT. The WSN nodes implement the TCP/IP stack (using compatible protocols such as ZigBee / 6LoWPAN stacks of 802.15.4). This method completely assimilates the wireless sensor nodes with the IoT. Wireless sensor nodes can be easily interconnected to the central coordinator (base station), and have communication vice-versa. This will also have the facility to implement different substation protocols onto a subgroup of the wireless sensor network.

The following are the No-IP based solutions related to wireless technologies available in the market.

- ZigBee is just a wireless technology produced by the ZigBee Alliance for low-data rate and short-range applications [44]. ZigBee stack consists of 04 basic levels: i) physical level, ii)

medium access control level, network level, and the application level. Physical and medium access control level of ZigBee is defined by the IEEE 802.15.4 standard, whilst the remaining portion of the stack is defined by the ZigBee specification.

- Z-Wave is just a wireless protocol design produced by ZenSys and promoted by the Z-Wave Alliance for automation in home and commercial environments. The key intent behind Z-Wave is allowing reliable transmission of short messages from a controller device to a number of nodes in network [45].
- INSTEON [46] is a solution developed for home automation by SmartLabs and promoted by the INSTEON Alliance. Distinctive options that come with INSTEON is the provision of a mesh topology consists of RF and power line links. Smart objects could be RF/power line/ has the ability to support different kinds of communication.
- Wavenis [47] is just a wireless protocol stack produced by Coronis Systems for control and monitoring applications in many environments, including home and building automation. It defines the functionality of physical, link, and network layers. Wavenis services could be accessed from upper layers via application programming interface (API).

Table 1 shows a comparison of existing wireless sensor network technologies with respect to their functionality. It can be seen that apart from 6LoWPAN a translational gateway is required for internet interconnectivity.

Table 1 Comparison of various wireless sensor network technologies [48]

		ZigBee	6LoWPAN	Z-Wave	INSTEON	Wavenis
Network layer	Multihop solution	Mesh routing, tree routing, and source routing	RPL	Source routing	Simulcast	Tree routing
	Hop limit	30/10/5 (mesh routing/tree routing/source routing)	255	4	4	N/A
	Multihop solution state	$O(N)$ (mesh routing), $O(1)$ (many-to-one routing)	$O(N)$ (root), $O(N_{DAGs})$ (other devices)	$O(N^2)$ (controller), $O(N_{prec})$ (routing slaves), no state (slaves)	No state	$O(N)$ (root), $O(1)$ (other devices)
End-to-end reliability		ACKs and control of duplicate packets	TCP/UDP/other	ACKs	ACKs and NAKs	—
Application layer	Command space	65,536 (clusters)	—	32,768	65,536	—
	Device type space	65,536	—	N/A	65,536	—
Security		Integrity, confidentiality, access control, and key management	Integrity, confidentiality, and access control (IEEE 802.15.4). Key management not currently supported.	128 bit AES encryption (400 series chip)	Encryption (e.g., rolling codes)	3DES and 128 bit AES encryption
Translation gateway needed for Internet connectivity		Yes	No	Yes (not needed for IP-Wave)	Yes	Yes
Specification publicly available		Yes	Yes	No	No	No

Table 2 shows the cost of a commercially available internetworking to connect wireless sensor network to the internet. As an example a system with 100 sensors and 10 gateways would have an initial cost of $7300 NZD. The monthly cost for running the system, with data storage, and 1000 transactions per day would be approximately $5700 NZD per month.

Table 2 Etherios Device Cloud Services pricing (in NZD)

Communication Module Cost	**Gateway Cost**	**IoT Services [49]**		
		Devices (per month)	Web Services	Data Storage
$45 **[50]**	$280 [51]	1-5 devices: Free 6-100 devices : $1.99 101-10000 devices: $0.74	$0.19 per 1000 transactions $2.49 monthly subscription	$0.11 per day

3 Contribution to the topic

The following tasks have been implemented in integrating WSN with IoT for the purpose of Home Automation.

- An effective low-cost and flexible solution for condition monitoring and energy management in home is presented.
- The basic operations include remote monitoring of household appliances or conditions through IoT.
- The novelty of the developed system is the internetworking mechanisms, which are practicable to integrate with co-modules like intelligent home monitoring systems for wellness determination of inhabitants [52] [53] [54] [55] [56].
- Performance measurements of network QoS for the integrated WSN with IoT with the proposed design have been presented.
- Design and development of a front-end for effectively isolating the WSN sensors from the Internet.
- Design and development of a gateway, allowing direct data exchange between sensors and a centralized server.
- Implementation of high-level internetworking technologies and integration of open source software for effective middleware management of the integrated WSN – IoT.

4 System Structure

4.1 Introduction

An integrated platform for IoT allows things to communicate with the internet. There are several challenges associated with connecting things to the internet (IoT). These are:

- A Network to connect things, which requires a communication bus and protocol.
- Providing internet connectivity to the network of things, which is achieved by an internet application gateway.
- Discovering and accessing services provided by the things, which requires a software service that can run on a server.
- Storing data produced by things, which requires a storage location which can be a server.

The end result of the internet of things is to provide people with access to information and services provided by things. Accessing things (from a client) can be done either directly or via a server. A server provides a centralised access point for an IoT platform, making service discovery, data storage, data access and security simple. Directly connecting to things from a client requires more advanced service discovery techniques, and more security measures. This thesis presents an integrated IoT platform that is server based, meaning clients communicate with things through a server.

4.1.1 Integrated Platform for IoT

The Integrated platform for IoT presented in this thesis consists of mainly four components, which is shown in Figure 2. The first component is the WSN as seen in Figure 2, which consists of sensor nodes and a coordinator. The second component is the Application gateway, as seen in Figure 2, which is connected to the coordinator of the wireless sensor network, and the internet. The application gateway provides the wireless sensor network with internet connectivity. The third component is a service running on a server, which is inter-connected via the internet to the application gateway. The Server stores data obtained from the WSN and provides access to the WSN and the stored data. The fourth component is a website, hosted on the server, which provides access to the stored data and the WSN. There are mainly two network domains involved with the integrate platform for IoT, the wireless sensor network and the internet. The application gateway bridges between these two network domains.

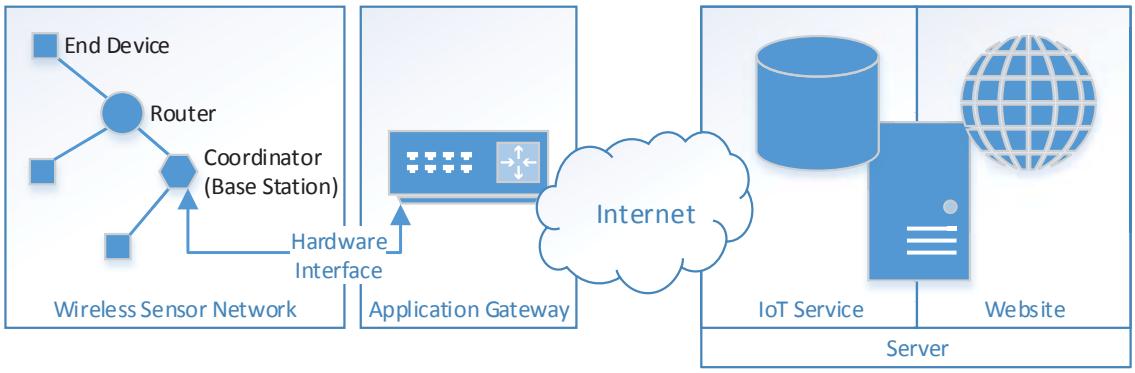

Figure 2 Structure of integrated IoT platform

The smart sensors consist of an XBee module and sensing device(s) OR a 6LoWPAN module and sensing device(s). The structure and requirements for the following components is given in the respective sections:

- ZigBee based implementation is given in section 4.3.
- 6LoWPAN based implementation is given in section 4.4.
- IoT Gateway is in section 4.5.
- IoT Server is in section 4.6.

4.2 Smart sensors requirements

A smart sensor is a sensor that can condition a signal, digitize the signal and communicate the signal. In order to do this a smart sensor consists of a sensor, microcontroller and communication bus. An analogue sensor converts a measurable quantity into an analogue voltage or current, this signal is converted to a digital representation of the signal by the microcontroller, and the digital representation

of the signal is sent via the communication bus. A digital sensor produces a digital value that represents the measurement, which fed into the microcontroller to be sent via the communication bus.

The following are types of sensors and examples of each type:

- Electrical – Voltage, Current, Phase Angle, Inductance, Capacitance, Resistance etc.
- Physical – Force, Movement, Acceleration, Orientation, Position etc.
- Environmental – Temperature, Humidity, Light intensity, Rain fall etc.

The communication is facilitated by either ZigBee standard or custom 6LoWPAN stack.

The combination of a sensor and communication bus produces a smart sensors. In the current context, examples of smart sensors and their use are:

- Electrical – Home appliances monitoring.
- Force Sensors – Home object usage monitoring.
- Motion detector – Inhabitant movement monitoring.
- Temperature, Humidity, Light intensity – Environmental monitoring.

4.2.1 Signal conditioning

Signal conditioning is required from analogue and digital inputs from sensors. Analogue inputs are provided in the form of a range of voltages, which must be conditioned to meet the input requirements for an analogue to digital converter (ADC) of the microcontroller. Analogue outputs are given to actuators in the form of a voltage produced by a digital to analogue converter (DAC). Digital inputs are provided in the form of voltages that indicate on or off, which must be conditioned to meet the input requirements of the microcontroller.

Analogue inputs and outputs may need condoning such as increasing or decreasing the gain, removing or adding an offset and filtering. Increasing or decreasing the gain of a signal may be need to allow the signal to use the full range of the ADC or DAC in the microcontroller to achieve the desired input or output. Adding or removing an offset may also be required to use the full range of the ADC or DAC. To get an accurate measurement from the signal, noise filtering may be required as there are various sources of noises that are added to an analogue signal.

Digital inputs and outputs are in the form of two distinct voltage levels, which are determined by the specifications. Adjustment of these levels may be required in order to satisfy the requirements of the microcontroller for a digital input. For digital outputs the microcontroller may produce voltages that are not sufficient or exceed the voltage levels required.

4.2.2 Signal Input and Output Processing

Signal Input processing involves the conversion of signals to a digital form, then packaging and transmitting the digital form. Signal output processing involves receiving and un-packaging signals in a digital form, then converting the digital form to a signal.

An analogue signal requires conversion with an ADC to obtain a digital measurement of the signal. The accuracy and granularity of the measurement depends on the specifications of the ADC. After being obtained the digital measurement must be packaged and transmitted via the communication bus.

To obtain an analogue output that represents a digital value received by the communication bus must be converted by a DAC. The digital value representing the analogue output must be in a packaged form in order to be received by the communication bus. The packaged form must be unpackaged to obtain the digital value, which can be converted by the DAC to obtain the analogue signal.

To produce digital output requires a packaged digital value to be received on the communication bus. The digital value must be unpacked in order to produce the digital output represented by the digital value.

Digital Input must be sampled and packaged in order to be transmitted on the communication bus. The digital input sample will be a digital value representing the state of the input, this value must be packaged in order to be transmitted on the communication bus.

The XBee modules provide analogue input and digital input or output, including the packing and unpacking of digital representations of the signals.

The 6LoWPAN modules use custom implementation to provide similar functionality to the XBee modules.

4.2.3 Network communication

The communication bus provides communication between smart sensors, and this requires management in order to create a network of smart sensors. In most cases a dedicated unit connected to the communication bus is required to manage and maintain the network of smart sensors. There are two requirements of a network of smart sensors, the first is that smart sensors can participate in this network, and the second is that smart sensors can send and receive sensor information on this network.

In order to create a network of smart sensors a dedicated unit connected to the communication bus is required. This dedicated unit is usually called the coordinator, as it coordinates the data obtain from smart sensors to form a network. The coordinator requires configuration of parameters that specify

the identity and operation of the network, and these settings must be identical for the smart sensors in order for them to join the network.

The primary function of a network of smart sensors is to provide access to sensors and actuators. This requires that information is exchanged between the coordinator of the network and the smart sensors. Smart sensors will produce measurements that must be sent to the coordinator, and to operate actuators information from the coordinator must be received.

4.3 XBee Structure

The XBee WSN consists of end devices and a coordinator which communicate using modulated radio signals. The arrangement of the end devices, coordinator and their links is the topology. There are two topologies, mesh and star, provided by the XBee modules. A ZigBee stack facilitates the communication between XBee modules, network configuration and management. This ZigBee stack is provided by the XBee module. Sensor output sampling is provided by an internal ADC of the XBee module.

4.3.1 XBee Stack Specifications

The XBee module provides a ZigBee stack as defined in Table 3. The structure of the stack follows the Open System Interconnection model, where the upper layers are the ZigBee stack and the lower layers are the IEEE 802.15.4 standard. The XBee ZigBee stacks operation is proprietary and details are not provided.

Table 3 ZigBee Stack

ZigBee Layer	Description
Physical	Defines the operation of the radio device which includes receive sensitivity, channel rejection, output power, number of channels, chip modulation, and transmission rate specifications. Most ZigBee applications operate on the 2.4 GHz ISM band at a 250kbps data rate as per the IEEE 802.15.4 specifications [57].
Media Access Control	Manages transactions between neighbouring devices (point to point). The MAC includes services such as transmission retry and acknowledgment management, and collision avoidance techniques.
Network	Adds routing capabilities that allows data packets to traverse multiple devices to route data packets from source to destination.
Application Support Sub layer	Application layer that defines various addressing objects including profiles, clusters, and endpoints.
ZigBee Device Object	Application layer that provides device and service discovery features and advanced network management capabilities

4.3.2 XBee Topologies

The XBee modules provide two network topologies, Star and Mesh. The star topology is a simple topology, which consists of a coordinator and end devices connected to it, as shown in Figure 3. Communication between end devices is by means of the coordinator from destination to source. Each node (coordinator or end device) in the network a packet passes through is called a hop, therefore there are two hops in order for end devices to communicate with each other.

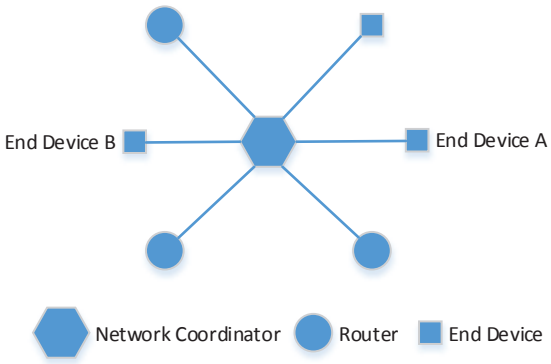

Figure 3 ZigBee start topology

All nodes in the star network communicate directly with the coordinator and not with each other. For example if end device A wants to communicate with end device B they communicate via the coordinator.

The mesh topology is a complex topology, consisting of a coordinator, routers and end devices, as shown in Figure 4. Each device can communicate with any device adjacent to it, therefore communication is via routing devices in a path between destination and source. The coordinator can also function as a routing device. The number of devices in this path between destination and source is called the number of hops a packet has to make to reach its destination. The advantage of a mesh topology is that it increases range, with the disadvantage that it is complex to implement.

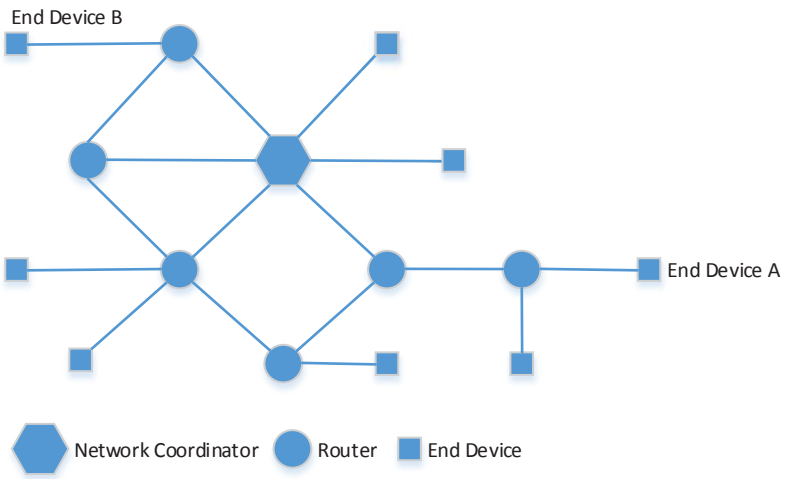

Figure 4 ZigBee Mesh topology

End devices communicate with routers if the destination is not adjacent the end device, the routers determine the route a packet will take to reach its destination. For example if end device A wants to communicate with end device B then any routers between them will be used to as a route for the communication to take place. The number of routers between device A and B are the number of hops.

When in operation the XBee modules may use a mesh topology or star topology with the appearance of each node be a single hop away. This means that routing decisions and packet transmission techniques are provided by the XBee module so that all packet transmissions appear as a single hop from peer to peer or peer to coordinator.

4.3.3 XBee Hardware Specification

The XBee module has 20 pins, which are used for powering the module, controlling the module, communicating with the module and input/output from the module. The details of these pins are given in Table 4 and their position on the XBee module given in Figure 5. There are 4 analogue input channels with 10 bit resolution capable of reading from 0 to 1.2 volts. These analogue channels can be configured as digital inputs or outputs, 7 additional digital inputs/outputs can also be configured.

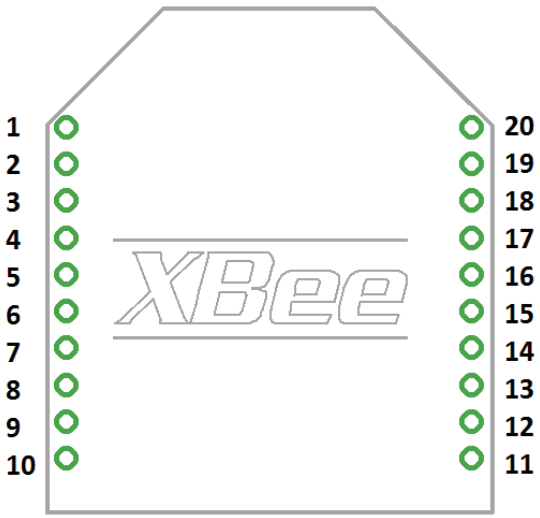

Figure 5 XBee Module pin numbering (top view)

Table 4 XBee Module pin information

Pin#	Name	Direction	Default State	Description
1	VCC	-	-	Power supply
2	DOUT	Output	Output	UART Data Out
3	DIN / CONFIG	Input	Input	UART Data In
4	DIO12	Both	Disabled	Digital I/O 12
5	RESET	Both	Open-Collector with pull-up	Module Reset (reset pulse must be at least 200ns)
6	RSSI PWM / DIO10	Both	Output	RX Signal Strength Indicator / Digital IO
7	DIO11	Both	Input	Digital I/O 11
8	[reserved]	-	Disabled	Do not connect
9	DTR / SLEEP_RQ/ DIO8	Both	Input	Pin Sleep Control Line or Digital IO 8
10	GND	-	-	Ground
11	DIO4	Both	Disabled	Digital I/O 4
12	CTS / DIO7	Both	Output	Clear-to-Send Flow Control or Digital I/O 7. CTS, if enabled, is an output.
13	ON / SLEEP	Output	Output	Module Status Indicator or Digital I/O 9
14	VREF	Input	-	Not used for EM250. Used for programmable secondary processor. For compatibility with other XBEE modules, we recommend connecting this pin voltage reference if Analog sampling is desired. Otherwise, connect to GND.
15	Associate / DIO5	Both	Output	Associated Indicator, Digital I/O 5
16	RTS / DIO6	Both	Input	Request-to-Send Flow Control, Digital I/O 6. RTS, if enabled, is an input.
17	AD3 / DIO3	Both	Disabled	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Both	Disabled	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Both	Disabled	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0 / Commissioning Button	Both	Disabled	Analog Input 0, Digital IO 0, or Commissioning Button

4.3.4 IoT adaptation

In order to provide internet connectivity to an XBee based WSN an application gateway is required to translate ZigBee to IPv6 as there are no mechanisms for connecting the XBee WSN to the internet without a gateway.

4.4 6LoWPAN Structure

A 6LoWPAN WSN consists of nodes and an edge router (application gateway) which communicate over a medium that uses modulated radio signals. The way in which the nodes communicate with the edge

router and each other is the type of topology. A star topology was implemented for simplicity as there is a high complexity in implementing nodes that can act as routers for the mesh or tree topologies. The nodes require a 6LoWPAN stack, and the edge router requires a 6LoWPAN-IPv6 translation stack.

4.4.1 6LoWPAN Stack Requirements

A 6LoWPAN stack enables application to application communication between an application on a web server and an application on a 6LoWPAN node, as shown in Figure 6. The stack on the edge router must translate between IPv6 format and 6LoWPAN format and the node must decode and encode the 6LoWPAN format to achieve application to application communication. The stacks allow the abstraction of the communication to a simple peer to peer connection between the applications. Similar to ZigBee the 6LoWPAN stack operates above a MAC and PHY layer, defined by the IEEE 802.15.4 standard.

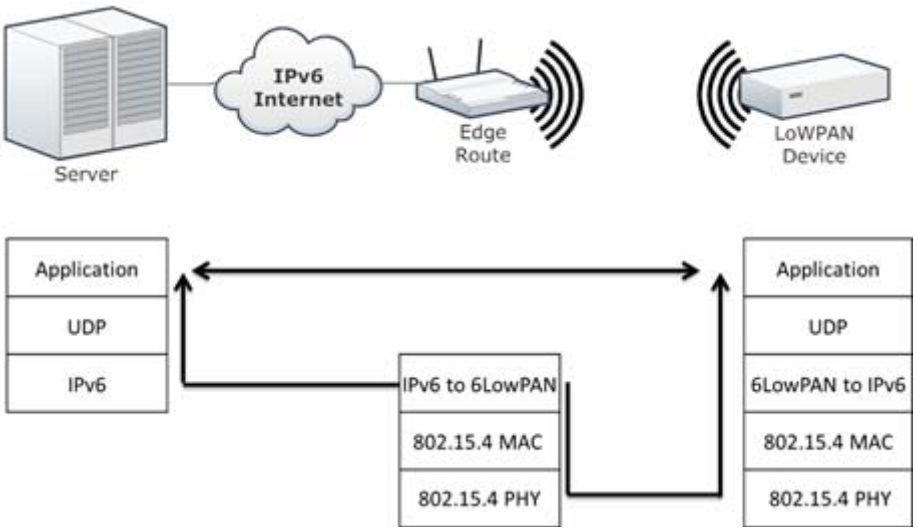

Figure 6 IPv6 and 6LoWPAN stack operation to give Server application to Sensor Application communication

The IEEE 802.15.4 Standard defines radio parameters in the PHY layer, and in the MAC layer network management, and communication techniques are defined. The 6LoWPAN stack requires the functionality provided by the IEEE 802.15.4 MAC layer to operate.

4.4.2 6LoWPAN Topologies

A simple star topology is possible using the IEEE 802.15.4 MAC and PHY layers as they allow for node to node communication. Additional mechanisms in the 6LoWPAN stack for other network topologies are in the draft stage and therefore were not used. These mechanisms allow for a tree or mesh topology as nodes can perform routing, which is in the draft stage. Figure 7 shows a possible topology of 6LoWPAN with IPv6 using the draft standards for routing between 6LoWPAN nodes.

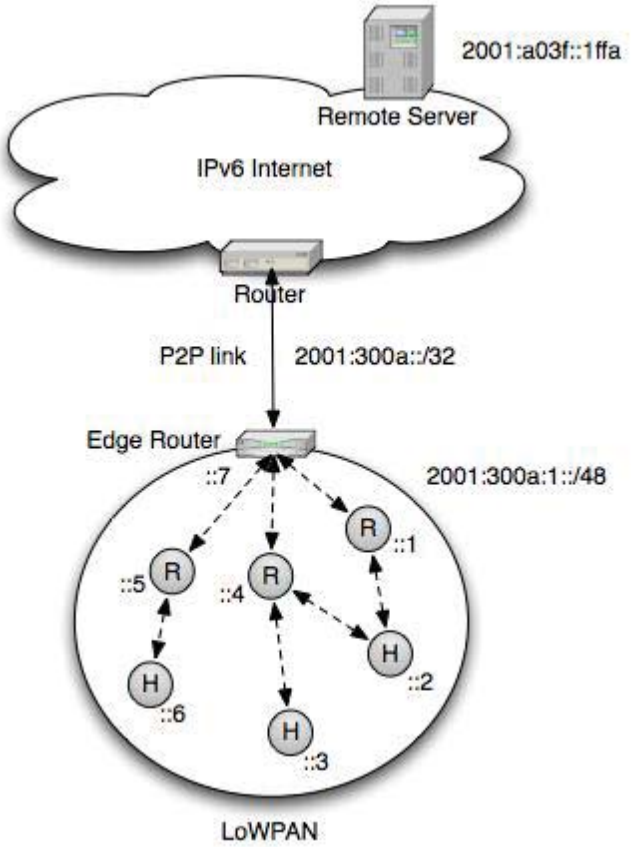

Figure 7 6LoWPAN and IPv6 network topology

4.4.3 6LoWPAN Hardware Specifications

A microprocessor attach to a radio transceiver is required to create a 6LoWPAN node. The 6LoWPAN stack will run on the microprocessor using the radio transceiver to communicate as per IEEE 802.15.4. The hardware specifications for a 6LoWPAN node have been made similar to that of an XBee module in order for them to operate with the same sensors.

At least 4 analogue input channels must be supported by the 6LoWPAN node, and must be of at least 10 bit resolution to sample a range of 0 to 1.2 volts. In addition to the analogue input, 11 digital inputs and outputs are required. These requirements will allow sensors for the XBee based IoT platform to be used with the 6LoWPAN based IoT platform.

In order to develop and test the 6LoWPAN stack on the microprocessor a debugging hardware interface is required. In addition to this an Integrated Development Environment (IDE) that supports the debugging hardware is required.

4.4.4 6LoWPAN Hardware Options

There are several hardware platforms that can support the development of a 6LoWPAN stack, these are listed in Table 3. The CC430 platform by Texas Instruments was chosen for development of a 6LoWPAN stack and the implementation details are in section 5.5.2. It was chosen for the following reasons:

- Low cost of the modules and debugging hardware. The cc430 based modules are similar in cost to em250 based modules however the debugging hardware is much cheaper.
- Low current consumption and high power output. The cc430 based modules provide higher power output for a similar current consumption when compared to other modules. This is because it operates in the 900MHz band.

Table 5 6LoWPAN Development Hardware

Hardware	Price in NZD			Microcontroller Parameters			Radio Parameters			
	Module	Number of Modules	Debugging Hardware	Frequency (MHZ)	RAM(KB)	ROM(KB)	Frequency	Power Output (dBm)	Transmit Current (mA)	Receive Current (mA)
AVR Raven	226.6 [58]	2 [58]	81.30 [59]	20 [60]	16 [60]	128 [60]	2.4GHz [61]	3 [61]	17.0 [61]	16.0 [61]
STM32W	62.50 [62]	1 [62]	34.82 [63]	24 [64]	16 [64]	256 [64]	2.4GHz [64]	3 [64]	31.0 [64]	27.0 [64]
EM250	45.00 [50]	1 [50]	4175.00 [65]	24 [66]	5 [66]	128 [66]	2.4GHz [66]	5 [66]	36.0 [66]	36.0 [66]
CC430	47.20 [67]	1 [67]	7.00 [68]	20 [69]	4 [69]	32 [69]	900MHz [69]	10 [69]	32 [69]	15.0 [69]

4.5 IoT Gateway

The integrated platform requires IoT gateway to connect the WSN with Internet, by translating between the two networks. The gateway needs an interface to the WSN, and an interface to the internet. The interface to the WSN is a hardware based as it needs to interface with an XBee coordinator for the ZigBee based system, or a radio transceiver for the 6LoWPAN system. This means that a suitable hardware platform is required to support network interfaces and hardware interfaces.

4.5.1 Gateway Hardware

The gateway needs to be compact and low cost with network capabilities, and therefore an embedded platform is required. Adequate processing power and storage to support an embedded operating

system is required in order to provide networking functionality. The embedded platform needs to provide a serial interface for communicating with the XBee Coordinator for the ZigBee based platform and the 6LoWPAN radio. The 6LoWPAN platform will provide access to a radio transceiver via a serial port for simplicity. To support internet connectivity either a WiFi or Ethernet port is required. A WRT54GL router and a Raspberry PI were used for the gateway hardware, details of these are in sections 5.4.1.1.

4.5.2 WSN Interface

To interface with the ZigBee based WSN, an XBee Coordinator is required. The coordinator provides a serial interface to communicate with the module. The IoT gateway must communicate with the coordinator over this serial interface. The customised software running on the IoT gateway must interface with the XBee coordinator using the XBee API packet format. The implementation details of the hardware interface created to interface with the XBee coordinator are in section 5.4.1.1.

To interface with the 6LoWPAN based WSN, a radio interface is required. The radio interface must be capable of sending and receiving packets from the 6LoWPAN nodes and provide these packets to the IoT gateway. 6LoWPAN coordinator. Management of the 6LoWPAN network is done by the IoT gateway. The implementation details of the hardware interface created to interface with the 6LoWPAN radio are in section 5.5.3.1.

4.5.3 IoT communication

The WSN Interface described in the previous section will allow for packets form the WSN to be sent and received. These packets need to be processed in order to provide the WSN with internet access. The processing will involve address translation of addresses, encapsulation of data to form an internet packet and unpacking of internet packets to form a WSN packet. The address translation requires the translation of WSN network addresses to an Internet addresses and vice-versa. The encapsulation of data requires the extraction of data from a WSN packet and encapsulation of this data in an internet packet. The unpacking of internet packets requires the extraction of data from an internet packet and the encapsulation of this data in a WSN packet. The implementation details of the XBee based IoT gateway are in section 5.4.4, and the implementation details of the 6LoWPAN IoT gateway are in section 5.5.3.

4.6 IoT Server

In order to collect and store data from the WSN via the IoT gateway an internet connected server is required. Controlling the WSN via the IoT gateway must also be performed by the server. To access the stored data and control the WSN a website hosted on the server is required.

4.6.1 Minimum requirements

A server is a computer running an operating system, running a customised software service for the integrated platform. In order to support the operating system and customised service the following minimum requirements need to be met:

- 1 GHZ Processor.
- 512 MB of RAM.
- 20 GB hard drive for storage.
- Ethernet port or WiFi for internet Connectivity.

4.6.2 Network connectivity

An internet connection is required to communicate with the IoT gateway and provide access to the website. The internet connection must support high download and upload rates in order to decrease network delay and handle traffic generated by the IoT gateways and website.

The IoT gateways will connect to the server using a Virtual Private Network over the internet. The VPN connection must be secure in order to protect the communication between the server and IoT gateway from intrusion and attempts to acquire sensor data or disrupt the network. The software used to create a VPN was OpenVPN, further details are in section 5.2.3.

4.6.3 Data Reception, Storage and Web server

A custom software service on the server is required to collect and store data from the WSN via the IoT gateways. The service must receive and unpack sensor data, and then store this data accordingly in a database. Sensor data must be stored in a way that the sensor that produced the data, and the time the data was produced can be retrieved. Additional information such as conversion formulae and sensor information must also be stored in the database. The implementation of the software service is given in section 5.6.1.

The sensor data stored in the database must be accessible from a website hosted on the server. The website is required to display individual sensor data in a graph, which requires the selection of a sensor and a time period to display. Graphed sensor data must be in the unit that the sensor measures, which requires the conversion formulae stored in the database to be used. Details of the implementation of the website are given in section 5.6.2.

4.7 Summary

A smart sensor system requires signal conditioning, processing and networking. The XBee modules provide this functionality, and the 6LoWPAN modules must be implemented to provide similar functionality. The XBee modules require an IoT gateway to provide internet connectivity to the XBee

WSN. The 6LoWPAN modules are require an implementation that will operate in a similar way to the XBee system in order to make a comparison. An IoT gateway for the 6LoWPAN modules is required to provide internet connectivity.

The core part of the integrated platform is the interconnection between WSN and the internet which must be provided by the IoT Gateway. This means the IoT gateway is required to interface with the WSN, connect to the internet, and provide a service that translates between the WSN and internet protocols.

A server running on a computer is required to collect data from the WSN connected to the internet and display this data on a website. The data collection must be achieved by the WSN communicating through the IoT gateway with a software service running on the server, in order to store sensor data on the server. The stored data must be accessible through a website that can display it in a graphical form.

5 Implementation

This chapter deals with the implementation required to create a WSN-IoT platform. The first two sections (5.1 and 5.2) provide information about the software tools required to develop the platform, and the software packages to support the platform. Details of techniques for network interfacing and parsing network packets are given in Section 5.3. The Implementation of the ZigBee based platform in is section 5.4, and the 6LoWPAN based platform is in section 5.5. The IoT server which aggregates and displays sensor information from either the ZigBee base platform, or 6LoWPAN platform is given in section 5.6.

5.1 Software Development Environment

There were several tools required to create a software development environment for developing the software required for the IoT platform. These tools are incorporated into an Integrated Development Environment (IDE) which is used to manage, edit, compile and debug source code. The following sections give details on how to setup and configure the tools in order to develop the software required for each component of the IoT platform. Working examples are given to illustrate various configuration steps to create an executable and debug the executable.

5.1.1 Code Composer Studio Setup – 6LoWPAN Module

The 6LoWPAN Modules required software to be developed, which needs an IDE with microcontroller debugging capabilities. Code Composer Studio (CCS) was used to create the firmware and debug the firmware on the microcontroller. The following sections give details on how to obtain CCS, creating an example project and debugging the example project.

5.1.1.1 Obtaining and installing Code Composer Studio

Code Composer Studio can be obtained from [70] which requires an account with TI. The account can be registered for free. Install Code composer with the default options, and when asked for a licensing option use the free code limited option.

5.1.1.2 Project creation for CC430

1. Create a new Code Composer Studio Project by clicking File, New and CCS Project as shown in Figure 8. This displays a dialogue box with configuration options for a new project.

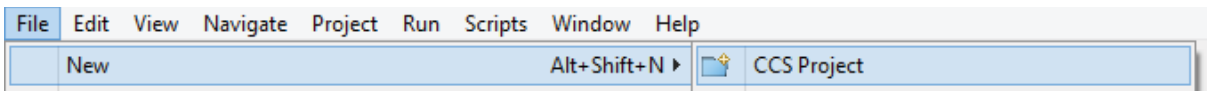

Figure 8 Code Composer Studio - Create new project.

2. The configuration options for a simple “Hello World” project are shown in Figure 9. The project name is the name that will appear in the project list in Code Composer Studio and is “Hello

World". The output type must be "Executable" as this will be the main program running on the microcontroller. In the device section the Family is "MSP430" and the variant must be chosen according to the microcontroller used. The CC430F5137 must be used for the OLIMEX MSP430-ccrf development board, and the CC430F6137 must be used for the TI CC430 development board. The connection configuration option should only be used when there are more than one debuggers attached, so the default option is used when only one debugger is connected. The project template allows for additional project types to be created to give additional features. Select the "Empty Project (with main.c)" to create a blank project with a main.c source code file. Click "Finish" to create the "Hello World" project.

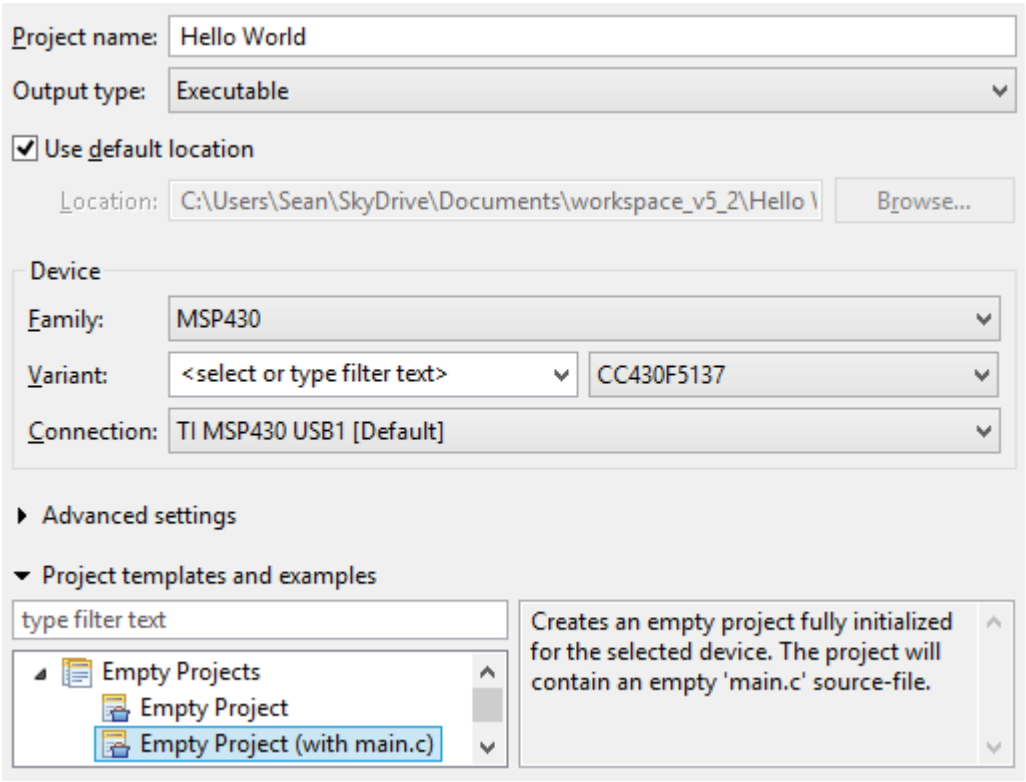

Figure 9 Code Composer Studio - Create new project options for "Hello World" project.

3. The newly created "Hello World" project will appear in the CCS Project Explorer as shown in Figure 10. When a project is selected it becomes the active project for debugging which is shown by "[Active – Debug]" appearing next to the project. The configuration options for debugging the microcontroller are stored in a ccxml file, with the name of the file corresponding to the type of microcontroller. When "[Active]" appears next to this file it means that the configuration is being used for the current debug settings, this allows for multiple debug configurations for multiple microcontrollers to be present. The debug configuration contains the type of microcontroller and what type of method is used to debug

it. In addition to debug configuration, a compiler configuration is needed which has the extension cmd. The compile configuration contains memory organisation information.

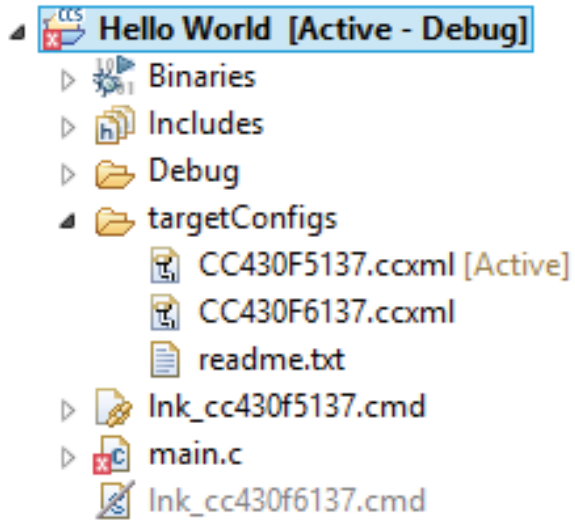

Figure 10 Code Composer Studio - "Hello World" project contents.

4. For a simple demonstration of debugging the following code was placed in the main.c file:

```
#include <msp430.h>

int main(void)
{
        WDTCTL = WDTPW | WDTHOLD;
        P1DIR = BIT0;
        P1OUT = BIT0;
        return 0;
}
```

This code disables the watch dog timer and sets the output of a pin connected to an LED to make it glow.

5.1.1.3 Debugging a CC430 Project

1. To start a debug session click the debug button as shown in Figure 11.

Figure 11 Code Composer Studio - Debug button to start debug process.

2. The first step to establish a debug session is the code must first be compiled into an executable, which is done automatically. The output for the compilation is shown in the Console. If there are errors in the code a dialogue box appears with the title “Errors in Workspace” asking if it should proceed from the launch. Clicking cancel will return to the

workspace in order to rectify the problem, after which repeating step 1 will restart the debugging process.

3. The next step is loading the compiled program onto the microcontroller and initialising debug parameters, which is done automatically. The dialogue box shown in Figure 12 will appear showing the status.

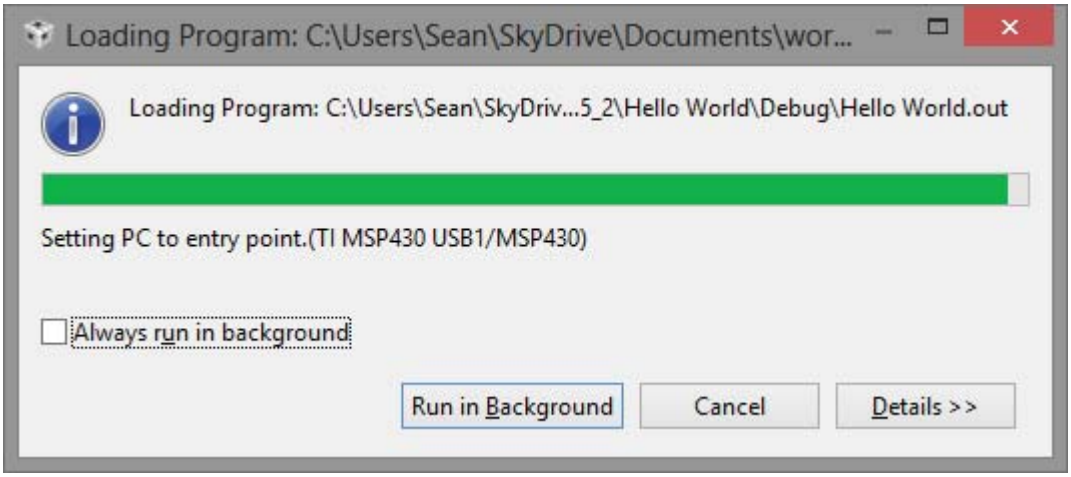

Figure 12 Code Composer Studio - Loading "Hello World" project onto microcontroller.

If the debugger is not connected to the computer the error shown in Figure 13 will appear. Check that the debugging hardware is connected with the computer.

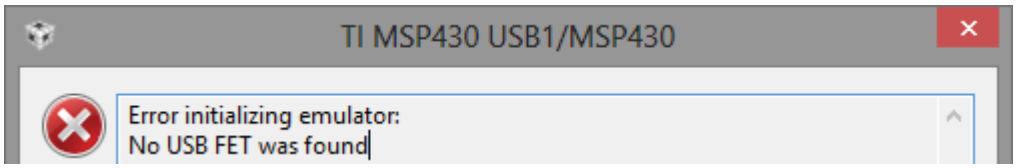

Figure 13 Code Composer Studio - No debugging hardware connected to the computer Error.

If there is a problem with the debugging hardware communicating with the microcontroller the error shown in Figure 14 will appear. Ensure that all the hardware connection to the required pins for debugging are correct, the microcontroller is powered and that the debugging hardware is correctly grounded with the microcontroller.

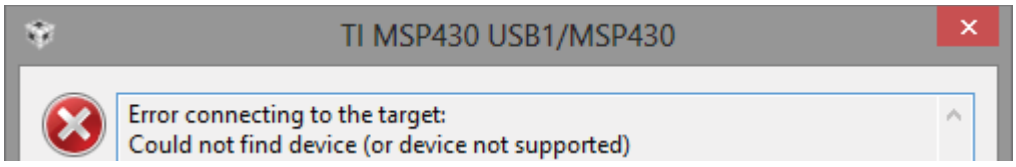

Figure 14 Code Composer Studio - Debugging hardware failing to communicate with microcontroller Error

If the incorrect microprocessor variant is selected then the error in Figure 15 appears. The microprocessor variant can be changed by right clicking on the project, selecting properties, and in the properties window in the general section select the correct microprocessor variant as shown in Figure 16.

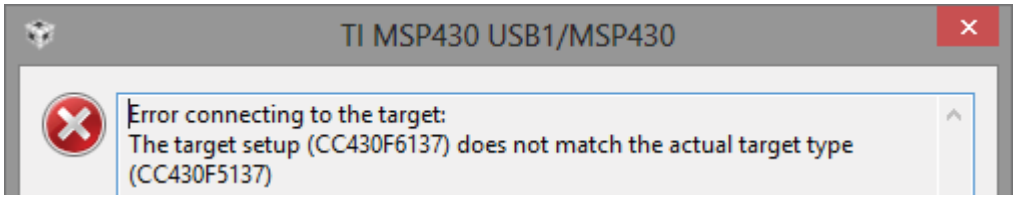

Figure 15 Code Composer Studio - Microprocessor variant incorrectly set for current microprocessor connected Error.

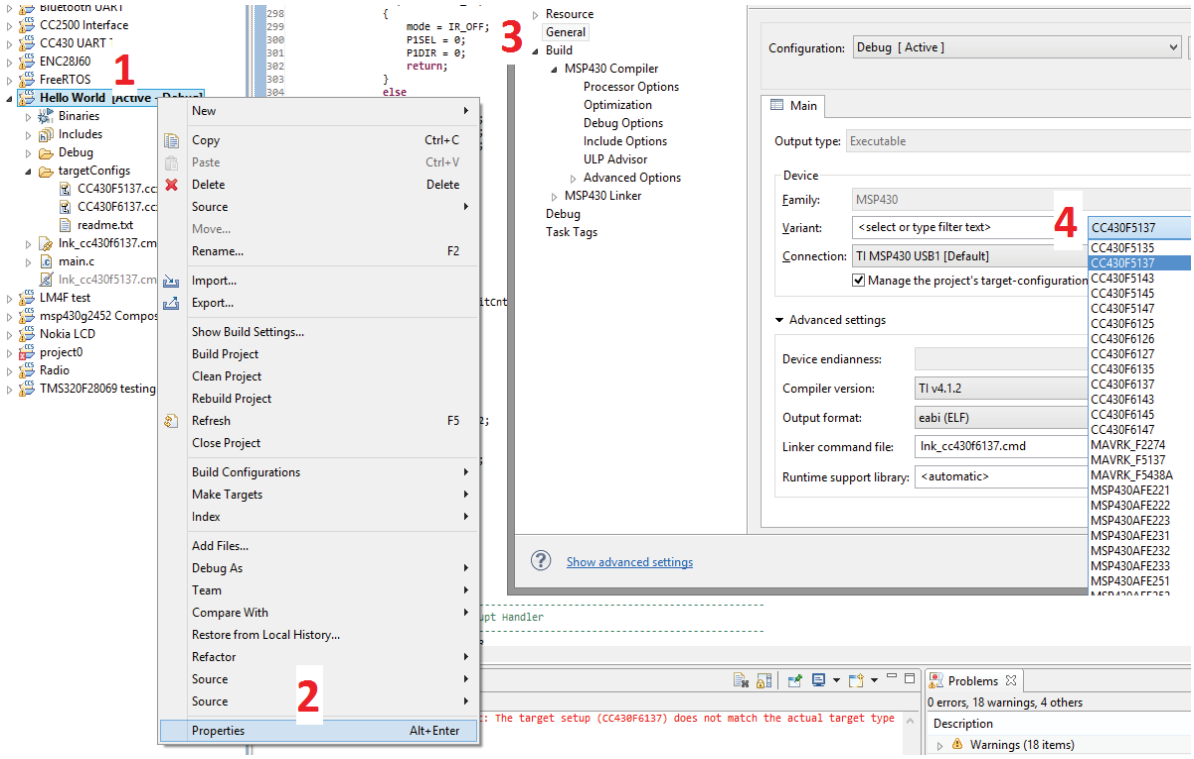

Figure 16 Code Composer Studio - Procedure for setting the microprocessor variant for "Hello World" project.

Figure 17 shows the state of code composer after the previous steps have taken place. The debugger has attached to the hardware and is ready for debugging. By default the debugger stops on the first line of code, as shown by Figure 17, where the line of code is highlighted green and an arrow appears next to that line.

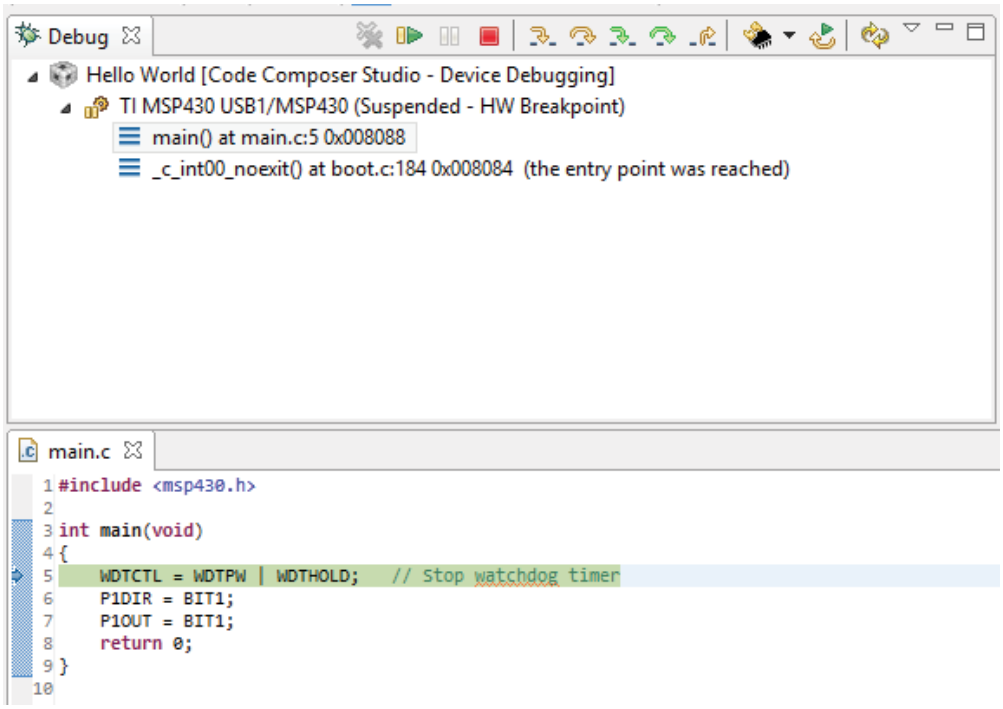

Figure 17 Code Composer Studio state after successfully starting a debug session.

Program flow is controlled by the buttons at the top of the Debug tab. In order to step through the program one line at a time the step button can be used. The resume button will execute the program until a break point is reached. Break points can be added by double clicking in the left column next to the line to break on.

4. Press the resume button to run the program and the led on the development board should light up.

This concludes the steps required to setup a development environment for the 6LoWPAN modules with a working example.

5.1.2 OpenWRT Toolchain with Eclipse IDE

The OpenWRT toolchain compiles source code into native programs that can be executed on hardware running OpenWRT. Coupling this toolchain with the Eclipse IDE provides a streamlined and effective way to write and debug software for OpenWRT based routers.

5.1.2.1 Windows OpenWRT Toolchain Setup

OpenWRT is an embedded version of Linux and therefore the OpenWRT toolchain was developed in Linux. In order to get the toolchain to work with Windows a Linux environment that runs in Windows is required. Cygwin provides this environment, however Linux programs must be recompiled in Cygwin to work with Windows. In the following chapters details on the installation of Cygwin and compilation of the OpenWRT toolchain will be given.

5.1.2.1.1 Cygwin Installation

1. Download and run the Cygwin setup, it can be obtained from [71]. The Cygwin setup consists of a wizard for configuring various options.
2. The first panel in the wizard is an introduction to the setup, click next.
3. The second panel configures download options. Cygwin consists of packages that need to be downloaded from the internet, the download options for these are selected on the second panel. Select “Install from Internet” and click next.
4. The third panel configures the installation directory, by default Cygwin is installed to “C:\cygwin”. Note the directory Cygwin is installed to, it will be used for configuring the OpenWRT toolchain and Eclipse.
5. The fourth panel configures a directory for the setup to store downloaded Cygwin packages.
6. The fifth panel configures the internet connection method, select direct connection. Connection through a proxy can be used however it is not possible to download the OpenWRT source code for the toolchain through a proxy.
7. The sixth panel configures mirrors to download the Cygwin packages from. Selecting a mirror that is geographically closer will usually increase download speeds. For example in New Zealand the mirror “http://ucmirror.cantebury.ac.nz” can be used. After clicking next the setup will download a package list from this mirror.
8. The seventh panel provides the selection of packages to be downloaded and installed. The packages are organised into categories. To install a package click “skip” in the new column of the package, a tick should appear in the “bin?” column, for example in Figure 18 the package “git” is selected to be installed. Searching for the package name makes finding and selecting packages easier, this is shown in Figure 18. The following packages are required to build the OpenWRT toolchain.

   From the “devel” category:

   - git
   - subversion
   - flex

- gcc-core
- gcc-g++
- make

From the “lib” category:

- zlib
- libiconv

From the “python” category:

- python

From the “web” category:

- wget

From the “archive” category

- unzip

From the “utils” category:

- patch
- util-linux

Figure 18 Selecting a Cygwin package to install

9. The OpenWRT toolchain requires source code for particular packages. To install the source code in addition to the package click on the check box in the “Src?” column as shown in Figure 19. The following packages and their source code need to be installed.

   From the category “libs”:

   - libmpc-devel
   - libgmp-devel
   - libmpfr-devel

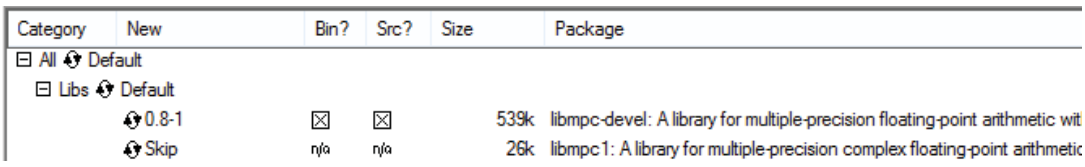

Figure 19 Selecting a Cygwin package to install with source code

10. Once all the Cygwin packages have been selected click next to continue to the eighth panel.

11. The eighth panel contains a list of additional packages that need to be installed to support the packages selected. After clicking next the setup will download and install all the Cygwin packages. This may take a significant period of time depending on the bandwidth of the internet connection.
12. Re-run setup.exe all the settings (installation directory, download directory, proxy and mirror) are saved from the previous setup. Proceed through and install the package libncurses-devel (under devel) binary and source with version 5.7-16. The version can be changed by clicking the number, the result is shown in Figure 20.

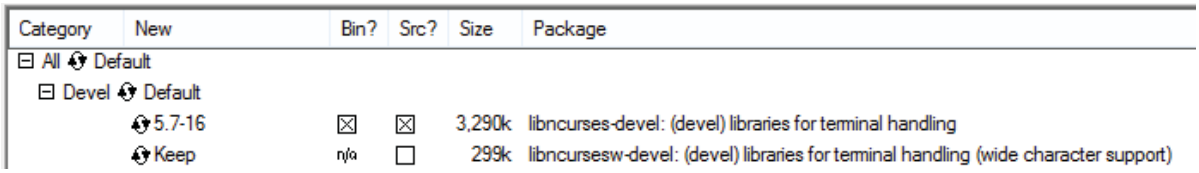

Figure 20 Selecting the libncurses version 5.7-16 Cygwin package for installation

13. A Cygwin tool called make used to build projects in Eclipse produces errors. An updated version is available at [72]. This fixes an error caused when multiple source files belonging to a project are built using make.

5.1.2.1.2 Windows configuration

Modifications to the windows configuration are required in order for windows to operate with Cygwin correctly. Cygwin provides Linux functionality which requires a case sensitive file system as Linux provides this. Additionally access to the cygwin binaries is required, and therefore their path must be added to the system so that their location is known.

5.1.2.1.2.1 Making Windows Case-Sensitive

The OpenWRT toolchain requires a case sensitive operating system. A case sensitive operating system means that files and folders are case sensitive. Windows is not a case sensitive operating system, this means that files and folders are not case sensitive, however the file system windows uses has the capability to be case sensitive. The following steps configure Windows to be case sensitive.

1. Open the registry editor, to do this press the windows key and “r” to open run dialog, enter “regedit” into the text box and click OK.
2. Navigate to
   “HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Kernel”
   in the registry editor.
3. Double click on the key name “obcaseinsensitive” and change its’ value from 1 to 0 as shown in Figure 21.

4. Restart the computer in order for the file system to become case sensitive.

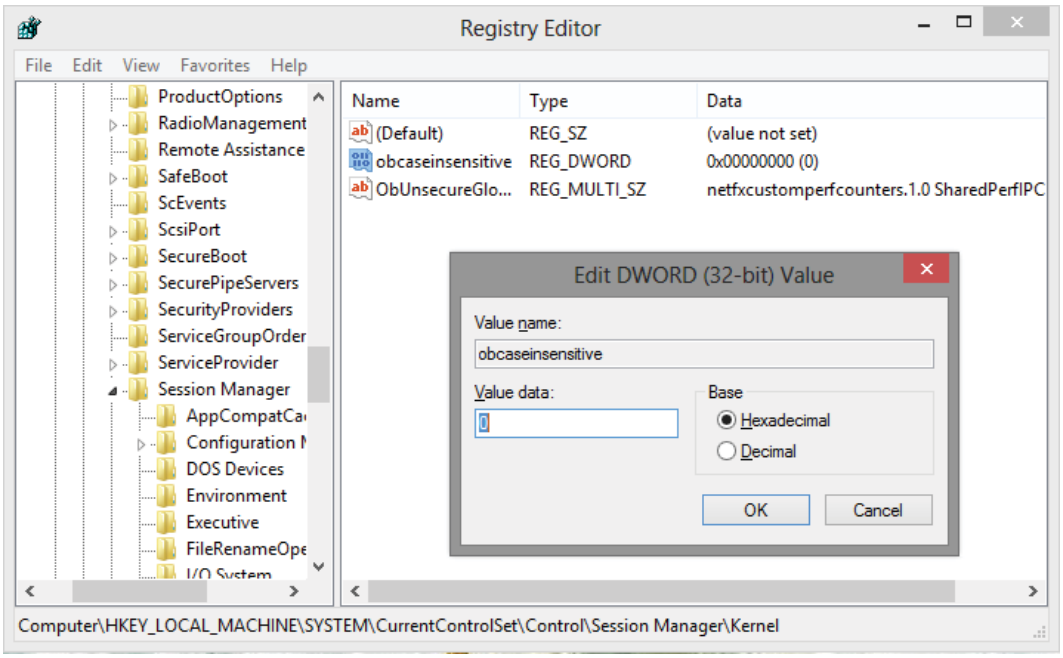

Figure 21 Editing registry to make windows case sensitive.

5.1.2.1.2.2 Adding Cygwin to the Path

The path of Cygwin Executables needs to be known to Windows, and therefore needs to be added to the path. To add a directory to the path open the “System Properties” and navigate to the advanced tab. Click the “Environment Variables…” button and add the path of the Cygwin executables as shown in Figure 22. To suppress file path warnings the User variable shown in Figure 23 can also be added.

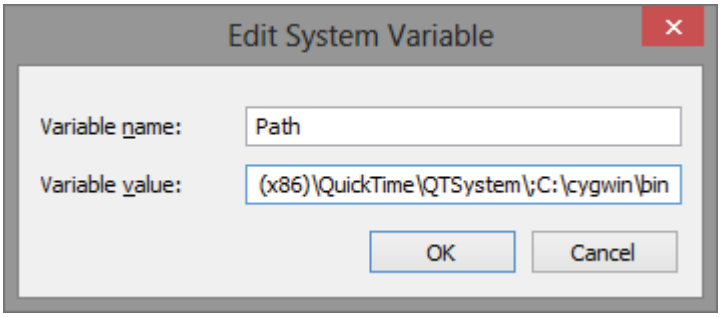

Figure 22 Adding Cygwin bin directory to the path

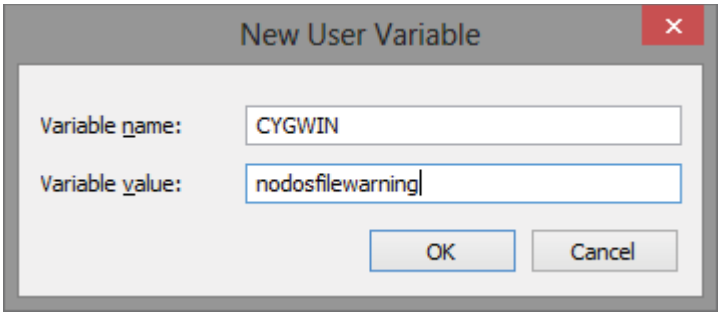

Figure 23 Adding CYGWIN environment variable to suppress warnings

5.1.2.1.3 OpenWRT tool chain compilation

1. Open a Cygwin console and create a directory for OpenWRT:

```
mkdir ~\openwrt
cd ~\openwrt
```

This directory is located in the Cygwin home directory for the user, which is for example C:\cygwin\home\Sean\openwrt

2. Checkout OpenWRT source code:

```
svn co svn://svn.openwrt.org/openwrt/trunk/
https://dev.openwrt.org/wiki/GetSource
```

Checking out the trunk from the SVN will download all the files needed to compile OpenWRT and the toolchain. These files will be located in a subdirectory “trunk” of the directory created in step 1, for example:

C:\cygwin\home\Sean\openwrt\trunk

3. Update and install feeds:

```
cd attitude_adjustment
./scripts/feeds update
./scripts/feeds install
```

In OpenWRT feeds are the source code and methods to build additional packages [73]. These packages act in the same way as packages for other distributions of Linux.

4. Modify scripts\patch-specs.sh by finding this section:

```
1   echo -n "Locating cpp ... "
2   for bin in bin usr/bin usr/local/bin; do
3       for cmd in "$DIR/$bin/"*-cpp*; do ←Add * to end of –cpp*
4               if [ -x "$cmd" ]; then
5                       echo "$cmd"
6                       CPP="$cmd"
7                       Break
8               Fi
9       done
10  done
```

Add “*” to –cpp on the third line, this is because the windows file system uses the extension “.exe” for an executable, however Linux doesn’t, so the script will fail to find executables in windows without this modification.

5. Start the configuration using the command:

```
make menuconfig
```

This performs a dependency check (packages installed in the Cygwin installation section 5.1.2.1.1 are used). If a package is missing use the steps in the Cygwin installation section 5.1.2.1.1 to install the required package. The package “ncurses” commonly gives problems as the Cygwin installer uninstalls the source files, therefore after installing any packages a final and separate install of ncurses is needed.

6. After the dependency check a graphical menu will appear, it can be navigated with the arrow keys. In the menuconfig select and configure the following options, as shown in Figure 24.
    a. “Target System”, for example the WRT54GL router target is “Broadcom BCM947xx/BCM953xx” [74].
    b. Advanced configuration options (for developers), select Toolchain options, and select “Build gdb”.
    c. “Build the OpenWRT based Toolchain”.

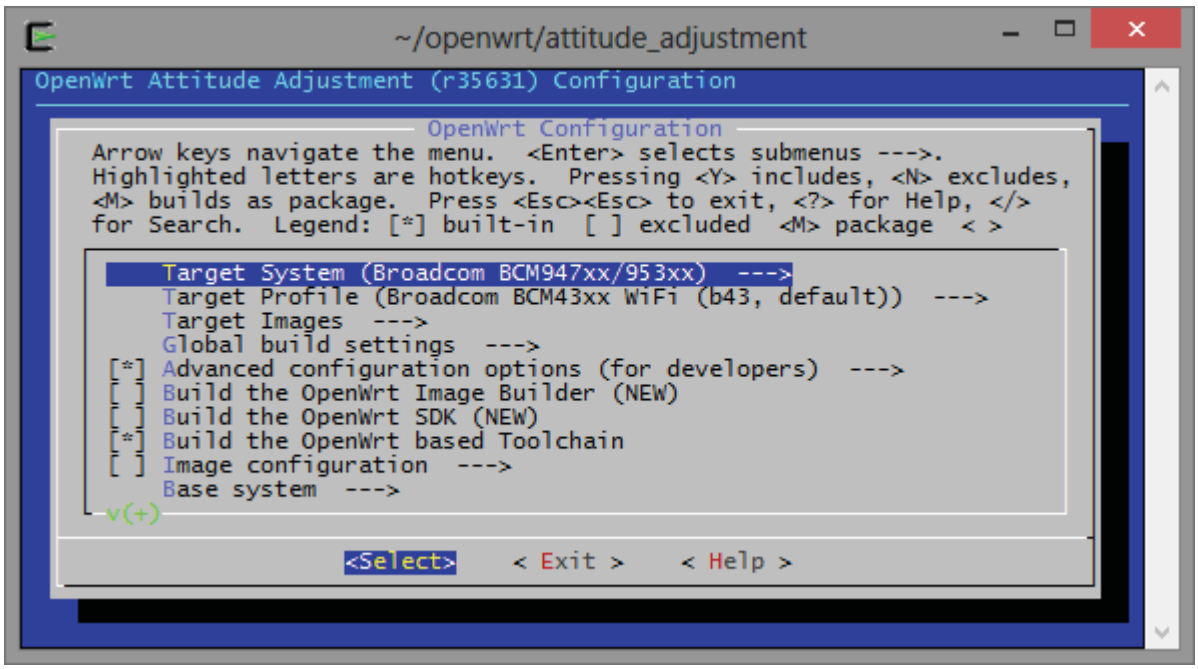

Figure 24 Selecting build options for OpenWRT.

7. Build and install the tools needed to build the toolchain using the commands:

```
make tools/libtool/install
make tools/autoconf/install
make tools/pkg-config/install
make tools/xz/install
make tools/automake/install
make tools/gmp/install
make tools/mpfr/install
make tools/mpc/install
make tools/libelf/install
make tools/flex/install
make tools/bison/install
make tools/mklibs/install
make tools/sstrip/install
make tools/ipkg-utils/install
make tools/genext2fs/install
```

8. Build the toolchain using the command:

```
make toolchain
```

Building the toolchain takes a considerable amount of time, as it is creating all the tools required to develop software for a router with OpenWRT installed.

5.1.2.2 Eclipse

Eclipse is a cross platform IDE that is used to edit, build and debug source code. It can be used to cross compile and remotely debug OpenWRT applications. The complex procedures to compile and debug OpenWRT software are handled by the Eclipse IDE, which makes designing software streamlined and efficient.

5.1.2.2.1 Required directories from the OpenWRT toolchain

The OpenWRT toolchain contains various tools to compile and debug software for OpenWRT. Eclipse uses these tools and therefore their locations need to be known. A list of these tools and examples of their locations is as following.

- The location of the toolchain, for example: "C:\cygwin\home\Sean\openwrt\attitude_adjustment\staging_dir\toolchain-mipsel_gcc-4.6-linaro_uClibc-0.9.33.2"
- The location of GDB, for example: "C:\cygwin\home\Sean\openwrt\attitude_adjustment\staging_dir\toolchain-mipsel_gcc-4.6-linaro_uClibc-0.9.33.2\bin\mipsel-openwrt-linux-gdb.exe"
- The location of the headers, for example:
- "C:\cygwin\home\Sean\openwrt\attitude_adjustment\staging_dir\toolchain-mipsel_gcc-4.6-linaro_uClibc-0.9.33.2\lib"
- The location of the libraries, for example: "C:\cygwin\home\Sean\openwrt\attitude_adjustment\staging_dir\toolchain-mipsel_gcc-4.6-linaro_uClibc-0.9.33.2\include"

5.1.2.2.2 Eclipse Setup for Windows/Linux

1. Eclipse can be obtained from [75], select Eclipse IDE for C/C++ developers as this has additional packages required for C/C++ development. Eclipse does not have an installer and is downloaded in an archive (zip file). The downloaded archive can be extracted to any location for example "C:\eclipse". A Java Runtime Environment (JRE) is required to run Eclipse, this can be obtained from [76].
2. Run Eclipse by executing "eclipse.exe" (a shortcut can be placed on the desktop/start menu for easier access). When prompted to select a workspace click ok and use the default workspace, for example "C:\Users\Sean\workspace". This is where the projects and source code is stored.
3. Additional packages are needed to cross compile using the OpenWRT toolchain and remotely debug software. These can be installed by click on the "Help" menu then "Install New Software". In the Install New software dialog select all available sites from the drop down box

as shown in Figure 25, this will download a package list. Once the list has downloaded check the boxes next to “C/C++ GCC Cross Compiler Support” and “Remote System Explorer End-User Runtime” as shown in Figure 25.

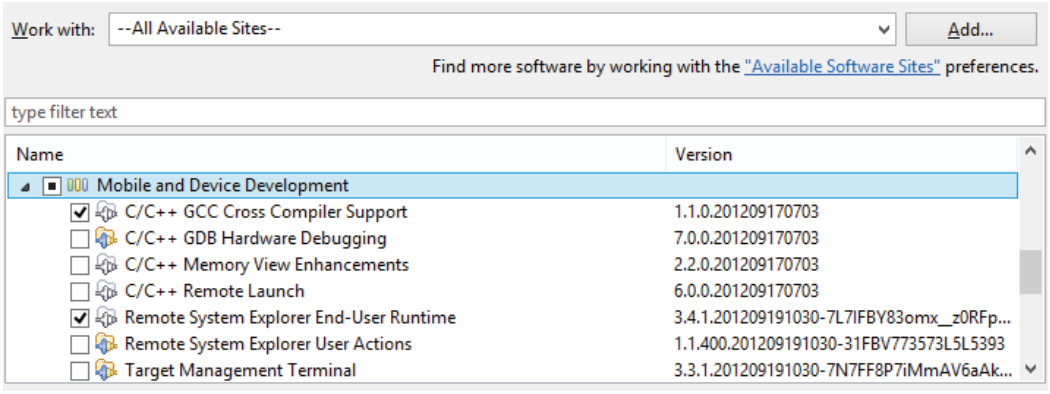

Figure 25 Eclipse - packages required for remote debug of OpenWRT software.

4. To complete the installation of the additional packages click next and agree to the terms and conditions, and finally click finish.

5.1.2.2.3 Eclipse Project Creation

Eclipse uses projects to manage source code, and compilation of software. The steps to create a simple “Hello World” project are presented in order to show how to configure Eclipse to compile and debug software for OpenWRT.

1. To create a new project, open eclipse and in the main window click File, New, and C Project. This brings up the new project dialog.

2. The first panel configures the project name and type. This is a simple example project so the name given is “Hello World’. Select Empty Project under the Executable category for the project type and “Cross GCC” for the toolchain as shown in Figure 26. Click next to proceed to the next panel.

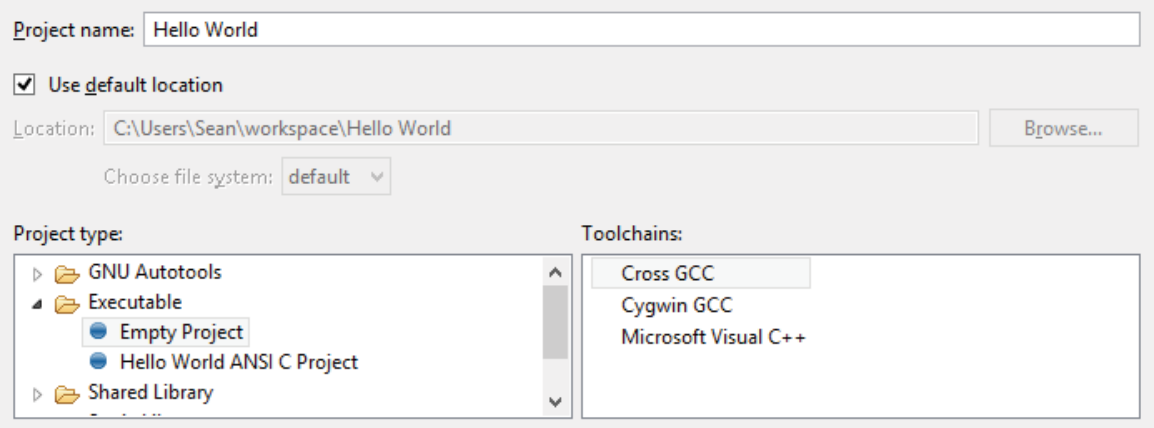

Figure 26 Eclipse - new project dialog setting the project name and type

3. The second panel provides access to the advanced project properties. Click on the “Advanced settings…” button to display the advanced project properties.
4. The paths of the headers files and libraries provided by the OpenWRT toolchain need to be added to the projects configuration because the Cygwin headers and libraries cannot be used for OpenWRT. These settings are found in the properties window for the project (from the previous step). In the tree view on the left of the properties window expand the Category “C/C++ General” and click “Paths and Symbols”, this displays a number of tabs for configuring the paths and symbols as shown in Figure 27.

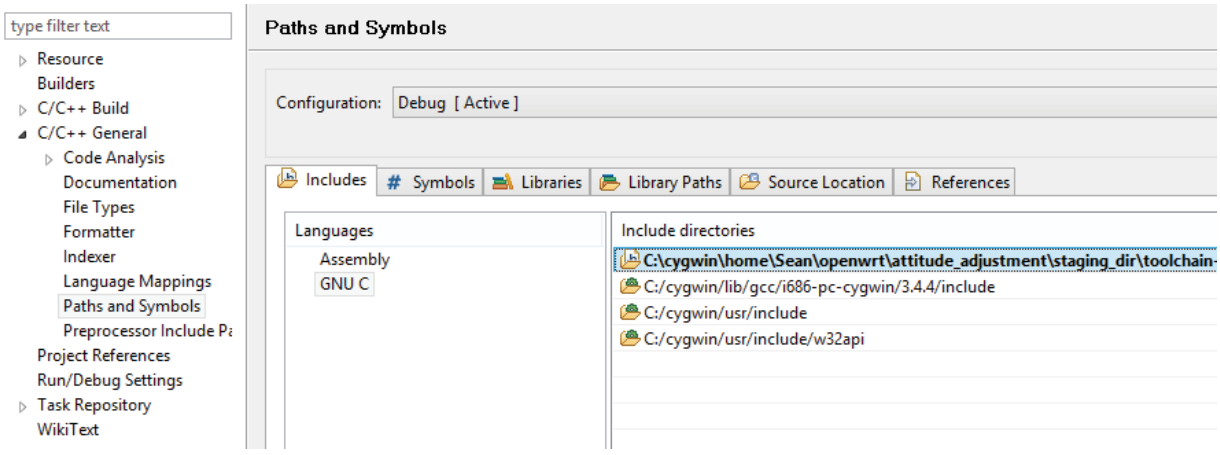

Figure 27 Eclipse - selecting "Paths and Symbols" in project properties to configure header and library paths.

The paths can be added as following:

a. To add the OpenWRT toolchains’ headers path click on the “Includes” tab. Ensure that under languages (on the left) “GNU C” is selected. Click add and enter the directory

that contains the header files for the OpenWRT toolchain, for example: “C:\cygwin\home\Sean\openwrt\attitude_adjustment\staging_dir\toolchain-mipsel_gcc-4.6-linaro_uClibc-0.9.33.2\include”

The result is shown in Figure 28.

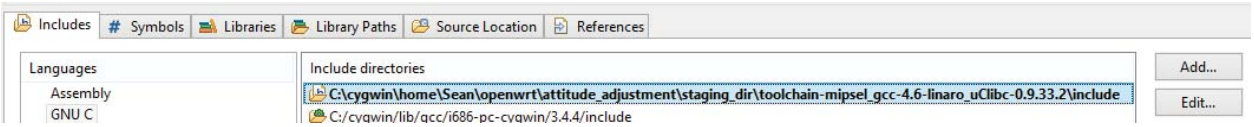

Figure 28 Eclipse - setting the include path for OpenWRT toolchain.

b. To add the OpenWRT toolchain libraries click on the “Library Paths” tab, and click add. Enter the directory that contains the libraries for the OpenWRT toolchain for example: “C:\cygwin\home\Sean\openwrt\attitude_adjustment\staging_dir\toolchain-mipsel_gcc-4.6-linaro_uClibc-0.9.33.2\lib”

The result is shown in Figure 29.

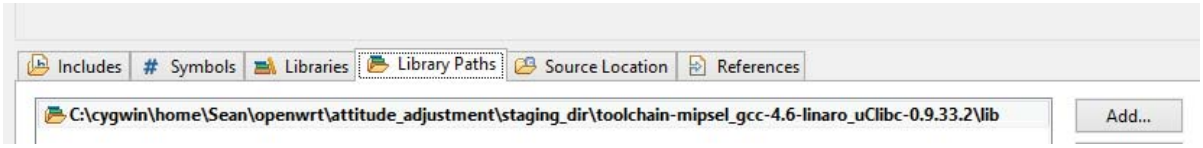

Figure 29 Eclipse - setting the library path for the OpenWRT toolchain.

Click ok in the bottom right to save the include and library paths, this will close the dialog and return to the second panel. Click next to proceed to the third panel.

5. The third panel configures the cross compiler, which is the OpenWRT toolchain. The cross compiler prefix is a prefix given to the tools used to compile in order to differentiate them from the native compilation tools. Set the Cross compiler prefix to the OpenWRT toolchain prefix, for example:

   “mipsel-openwrt-linux-“

   Set the Cross compiler path to the OpenWRT toolchain directory, for example: “C:\cygwin\home\Sean\openwrt\attitude_adjustment\staging_dir\toolchain-mipsel_gcc-4.6-linaro_uClibc-0.9.33.2”

   Figure 30 shows the settings for the cross compiler. Click finish to create the project.

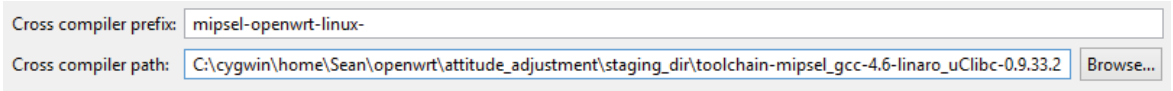

Figure 30 Eclipse - setting the cross-compiler prefix and path for the OpenWRT toolchain.

6. To test the cross compiler settings source code to compile is needed. A simple “hello world” program will be used to test it. Right click on the newly created project and select “New” and

then “Source File”. Set the Source file to main.c as shown in Figure 31. Click “Finish” to create a new source file.

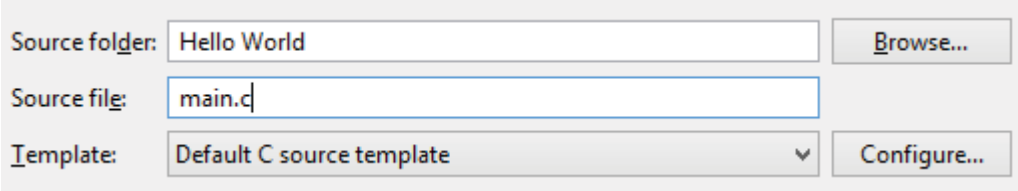

Figure 31 Creating a new source file main.c

Add the following lines of code to “main.c”:

```
#include <stdio.h>

int main()
{
        printf("hello world\r\n");
        return 0;
}
```

Save main.c so that it can be compiled.

7. To test that the cross compiler settings are correct the project needs to be built. Right click on the project and click Build Project to start the build process. Figure 32 shows the output of a successful build. The build process compiles and links the source code into an executable that will run on the OpenWRT based router.

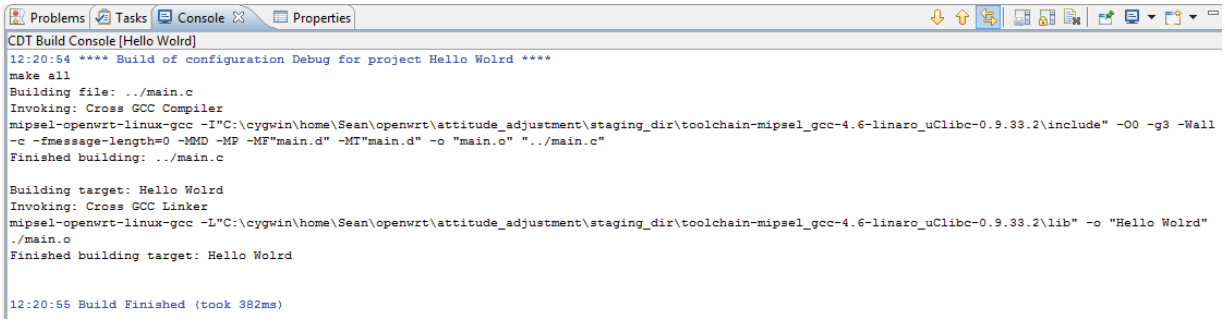

Figure 32 Eclipse - output of a successful build of the project "Hello World".

5.1.2.2.4 Eclipse Remote Debugging

Debugging software written for OpenWRT is essential to streamlining and improving efficiency. The problem is that the embedded hardware the software is designed for is very limited and does support graphical user environments for debugging. Therefore the debugging must be done remotely from Eclipse running on a computer. Eclipse provides remote debugging capabilities using the OpenWRT toolchain, the following steps setup remote debugging for the “Hello World” Project in the previous section.

5.1.2.2.4.1 Remote Resource

Eclipse needs a connection to target system in order to remotely debug it. These connections are called remote resources, which is managed by the Remote Systems. The following steps show how to configure an OpenWRT router on a local network as a remote resource. This can then be used to configure remote debugging, which is in the next section 5.1.2.2.4.2.

1. The OpenWRT router is a remote resource which needs to be added to the Remote System Explorer in eclipse. By default the Remote System Explorer is hidden. To show the explorer click “Window”, “Show View”, and then “Other” as shown in Figure 33. This will bring up a new window containing a list of views that can be added to the eclipse interface. Type “remote” into the search box at the top of the window, select “Remote Systems” as shown in Figure 33 and then click OK. This will put the Remote Systems view at the bottom of the main Eclipse window if the C/C++ perspective is active.

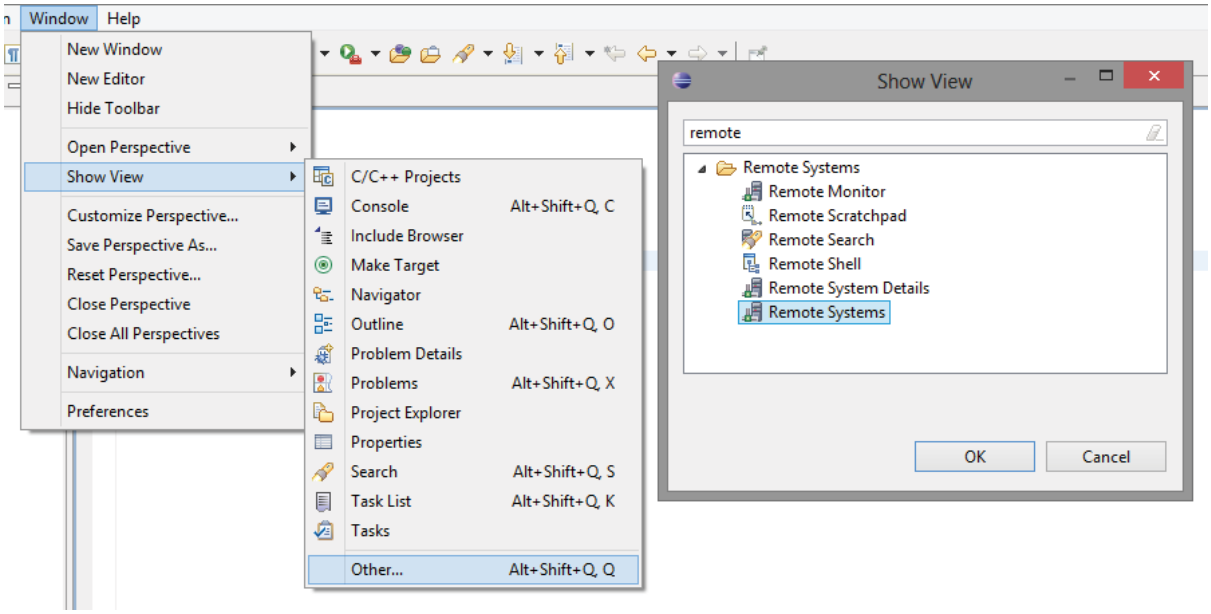

Figure 33 Eclipse - displaying Remote Systems view

2. To create a remote resource right click in the Remote Systems view and select new. This will show a wizard to configure a new remote resource for the OpenWRT router.
3. The first panel shows the types of remote resources available, select “Linux” from the “General” category, as shown in Figure 34. Click next to proceed to the second panel of the wizard.

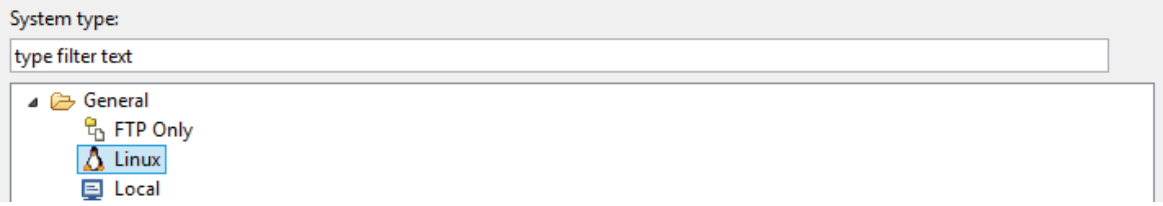

Figure 34 Eclipse - selecting the remote resource type "Linux" for the OpenWRT router.

4. The second panel configures the host name and connection name. The host name can either be a host name given by a domain name service (not applicable in most cases) or the IP address. The IP address of the OpenWRT router on the local network which was configured with an IP address of 192.168.1.100 which can be used as shown in Figure 35. The connection name is the user friendly name given to the connection, by default it is the host name given, as shown in Figure 35. Click next to proceed to the third panel of the wizard.

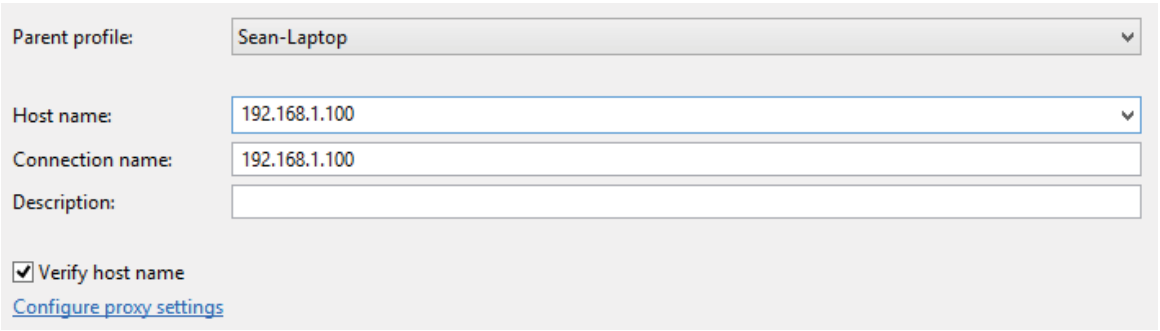

Figure 35 Eclipse - setting the Host name and Connection name for a remote resource.

5. The third panel of the wizard configures the method to upload files to the remote resource. Select “ssh.files” as the method to use, as shown in Figure 36. This uses a Secure Shell (ssh) to upload files to the OpenWRT router. Click next to proceed to the fourth panel.

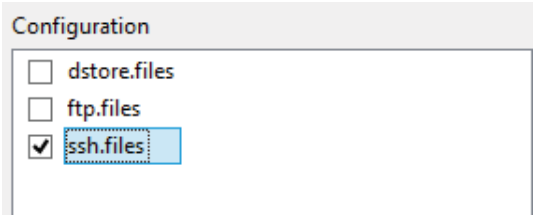

Figure 36 Eclipse - configuring remote resource file access.

6. The fourth panel configures the method to manage processes on the remote resource. Select “processes.shell.linux” as the method to use, as shown in Figure 37. This uses a shell to manage the processes on the OpenWRT router. Click next to proceed to the fifth panel.

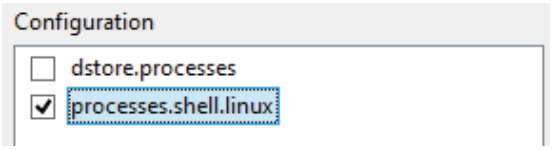

Figure 37 Eclipse - configuring remote resource process access method.

7. The fifth panel configures the method to connect to a shell on the remote resource. Select “ssh.shells” as the method to user, as shown in Figure 38. This uses a Secure Shell (ssh) to access the shell on the OpenWRT router. Click next to proceed to the sixth panel.

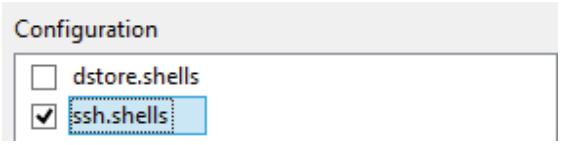

Figure 38 Eclipse - configuring remote resource shell access method.

8. The sixth panel configures the method to connect to a terminal on the remote resource. Select “ssh.terminals” as the method to use, as shown in Figure 39. This uses a Secure Shell (ssh) to access the terminal on the OpenWRT router. Click finish to complete the wizard.

Figure 39 Eclipse - configuring remote resource terminal access method.

9. The remote resource will appear in the Remote Systems view, to test it right click on the newly created remote resource and click connect. The first time a connection is made a user name and password dialog will appear, as shown in Figure 40. The user name and password are for a user on the OpenWRT router, and a user with root access is preferable. Selecting “Save user ID” and “Save password” will save the user name and password so that they do not have to be entered each time a connection is made.

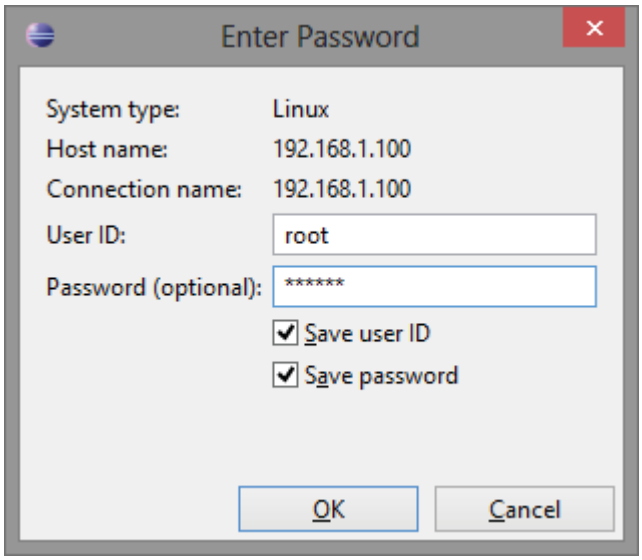

Figure 40 Eclipse - User name and password for connecting to a remote resource (OpenWRT router).

If the connection is successful a green plus icon will appear next to the remote resource in the Remote Systems view, as shown in Figure 41.

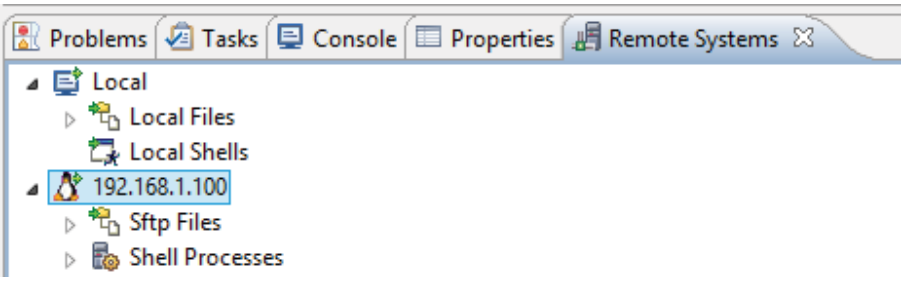

Figure 41 Eclipse - successful connection to a remote resource "192.168.1.100" which is the OpenWRT router.

5.1.2.2.4.2 Remote Debugging

To remotely debug the "Hello World" project configuration of the remote debugging in eclipse is required. The following steps give the details of configuring remote debugging and starting a remote debugging session to test the configuration.

1. A debug configuration needs to be added in order to remotely debug the "Hello World" program. The debug configuration can be added by clicking the arrow next to the debug icon and selecting "Debug Configurations…" as shown in Figure 42. This brings up the Debug Configuration dialog.

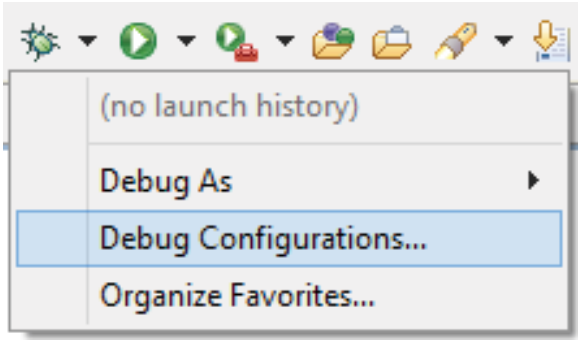

Figure 42 Eclipse - Menu to show debug configurations.

2. The Debug Configuration dialog allows for several types of debug configuration to be added. Click the "C/C++ Remote Application" category in the tree view on the left, as shown in Figure 43. To add a configuration click the "New launch configuration" button which is the left most button above the tree view. This will add a new configuration called "Hello World Debug" under the "C/C++ Remote Application" category as shown in Figure 43. The right side of the dialog will show parameters for this configuration. In the main tab the following settings need to be set (which can be seen in Figure 43):
    a. C/C++ Application needs to be set to the path of the executable generated by the compilation. Click the "Search Project" button to locate the executable. If different architectures are used an executable for each architecture will be present, and the according executable needs to be selected.

b. The Connection needs to be set to the remote connection configured in the previous section. This is connection used to upload the executable and debug the executable remotely.

c. The “Remote Absolute File Path for C/C++ Application” is the name that the executable will be given when uploaded to the OpenWRT router. “hello_world” was used as spaces are not allowed in the file names.

Click the “Apply” button at the bottom of the dialog to save these settings.

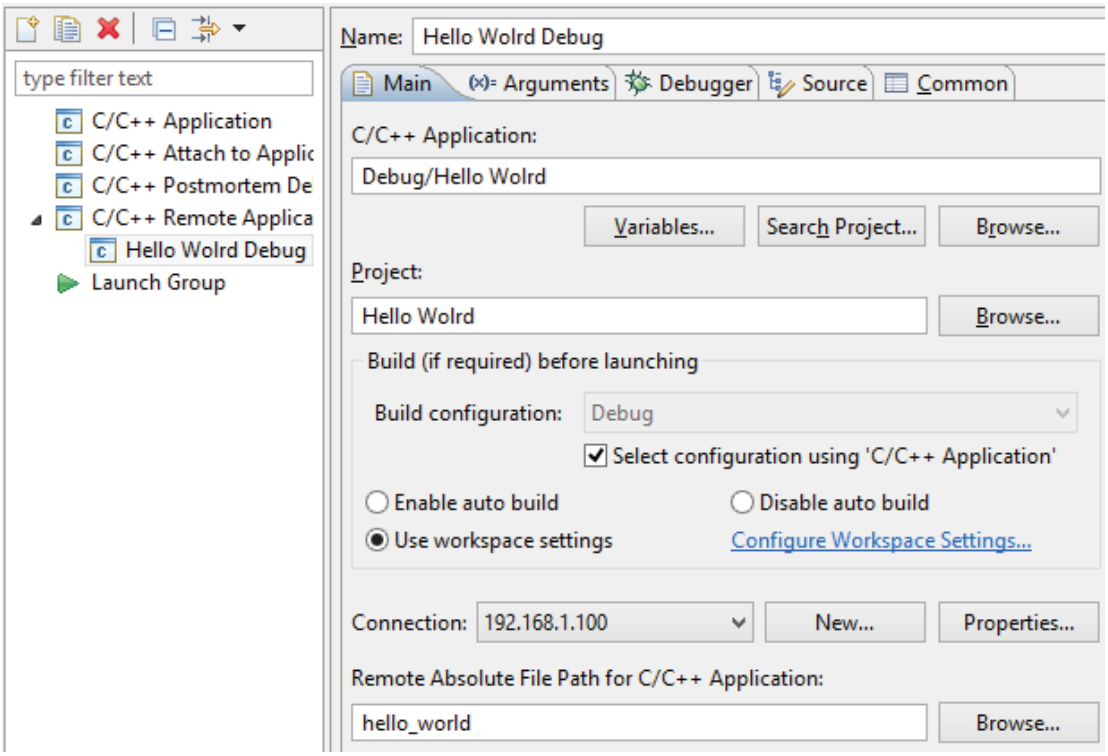

Figure 43 Eclipse - Main remote application debugging configuration for "Hello World" project.

3. Remote debugging is performed by a program specifically compiled by the OpenWRT toolchain for the architecture of the OpenWRT router. This program is called GNU Debugger (GDB) and its location is required in order for eclipse to remotely debug the OpenWRT router. An example of the location of the GDB executable produced by the OpenWRT toolchain is:

“C:\cygwin\home\Sean\openwrt\attitude_adjustment\build_dir\toolchain-mipsel_gcc-4.6-linaro_uClibc-0.9.33.2\gdb-linaro-7.2-2011.03-0\gdb\gdb.exe”

Use the Browse button as shown in Figure 44 to locate and set the location of the GDB executable. Click the “Apply” button at the bottom of the “Debug Configuration” dialog to save the configuration and then close the dialog.

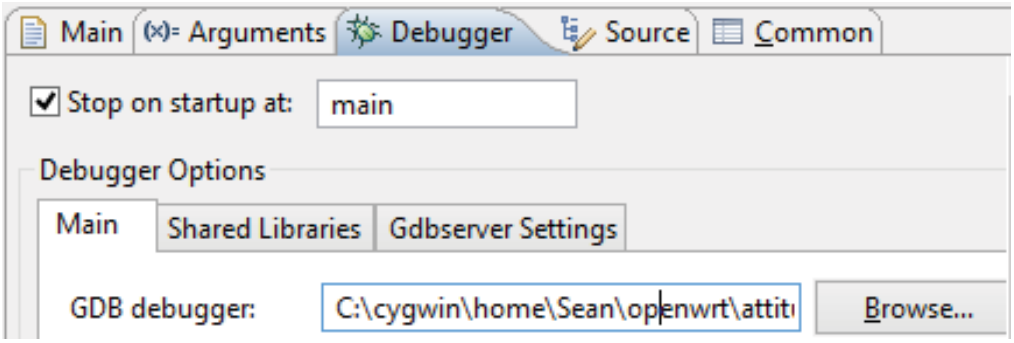

Figure 44 Eclipse - Setting Location of GDB produced by OpenWRT toolchain.

4. The newly created "Hello World Debug" configuration will now appear in the list of debug configurations as shown in Figure 45. The list is accessed by clicking the arrow next to the debug button. Clicking on the "Hello World Debug" configuration will start the debug process.

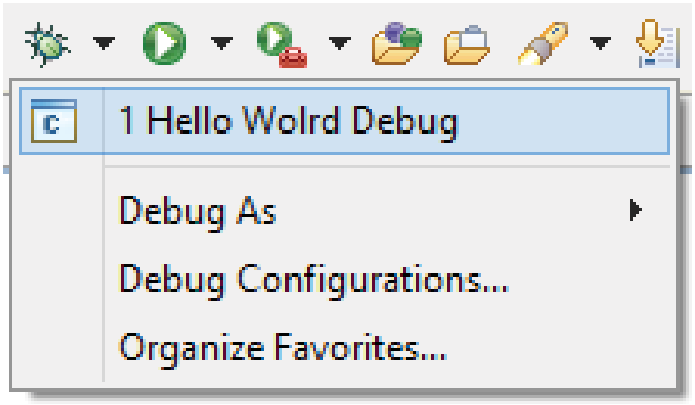

Figure 45 Eclipse - "Hello World Debug" configuration location.

5. If the debug process has been started for the first time a dialog shown in Figure 46 will appear. Clicking "Yes" will allow Eclipse to automatically switch to a different perspective that contains debugging related information.

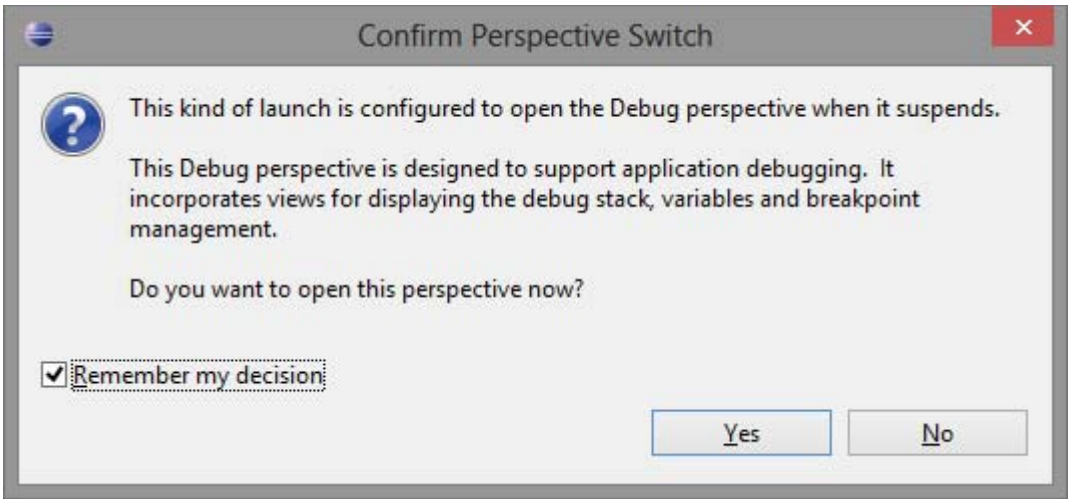

Figure 46 Eclipse - Confirming automatic change to Debug Perspective.

6. The process to start a debugging session includes compiling the project, uploading the executable to the OpenWRT router, and starting debugging services on the OpenWRT router. After completing those tasks Eclipse should be in a debug perspective. The error in Figure 47 occurs due to the difference in file naming schemes between Cygwin and Windows. To correct this error click the “Locate File” button and locate the main.c file contained in the “Hello World” project. For example the location of the main.c file is:
“C:\Users\Sean\workspace\Hello Wolrd\main.c”

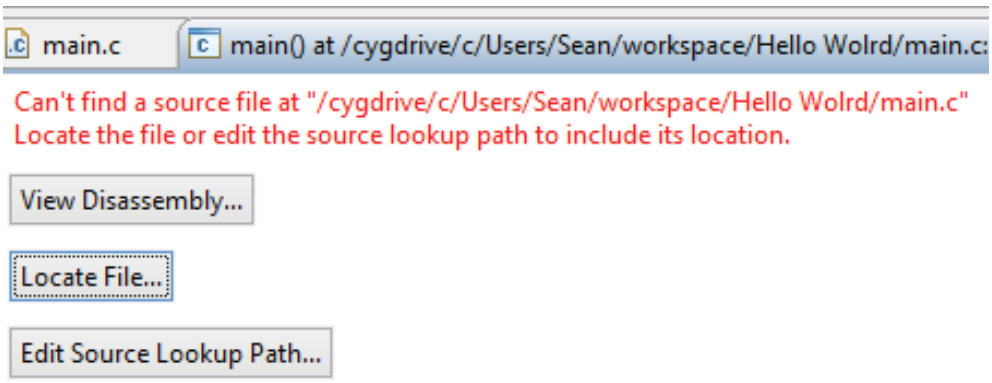

Figure 47 Eclipse - Unable to locate source files error.

7. The debug perspective should appear similar to Figure 48 once the location for the main.c file is known. The location of additional source files in the project do not require the same process as this initial step establishes the location of all sources files in the project. The initial state of the debugger is to break on the first line of code, which is shown by an arrow next to the line of code, and the green highlighting of the line of code as shown in the bottom of Figure 48. Breakpoints can be added by double clicking in the column next to the line. Buttons to control the program flow are located in the toolbar shown in the top of Figure 48 and are as following:
    a. The resume button executes the program until a break point is reached.
    b. The stop button stops execution of the program and the debug session.
    c. The step buttons provide line by line stepping of the code.

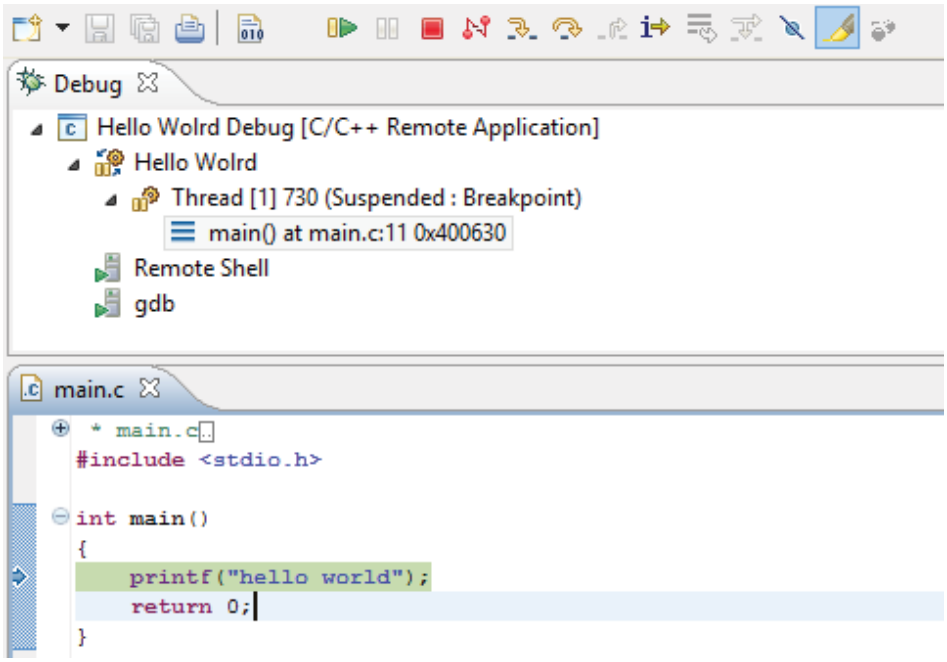

Figure 48 Eclipse - Debug perspective for the debugging of "Hello World" project.

8. A simple console for program output is provided at the button of the eclipse windows as shown in Figure 49. Pressing the step button will execute the line of code that outputs “hello world”, which should appear in the console.

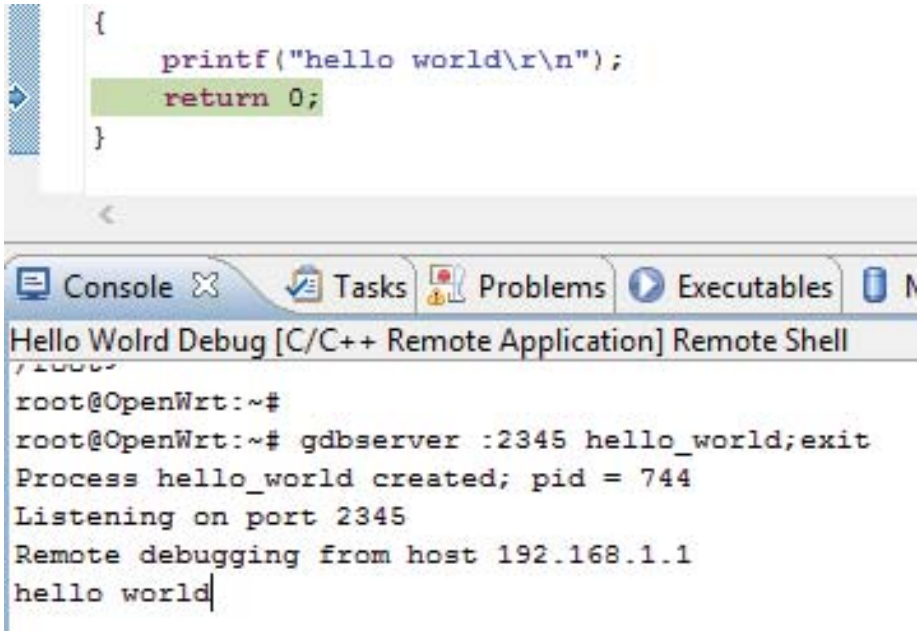

Figure 49 Eclipse - Program output in a debugging session.

This concludes the setup and configuration of the development environment for the OpenWRT router with a working example.

5.1.3 Visual Studio Setup

Visual Studio was selected to develop the services required on the server to receive and store sample data. The following sections give details on installing Visual Studio, installing required libraries and a simple example Visual Studio project. The version of Visual Studio used was the Premium edition, however details on installing the free Express Edition are given. The functionality of each edition is similar for the purpose of developing the required services for the server.

5.1.3.1 Installation

Visual Studio 2012 Express Edition can be obtained from [77]. Install Visual Studio Express Edition with default settings. In order to support MySQL an additional library is needed, this can be obtained from [78]. Install the MySQL library with default settings.

5.1.3.2 Project Creation

The following steps create an example project that uses a MySQL library to connect to a database located on the local machine. This can be used to check that the MySQL database and MySQL library are functioning correctly in order to provide database access. Setup of the MySQL database is given in section 5.2.1.

1. Start the “Create New Project” wizard by click file, new and project as shown in Figure 50.

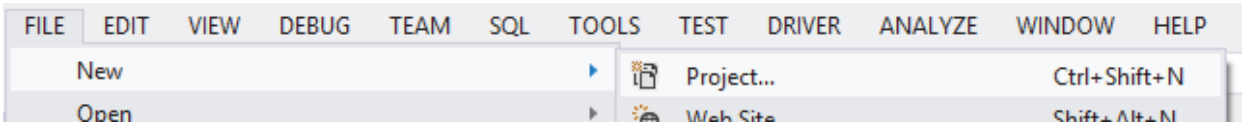

Figure 50 Visual Studio - Creating a new project.

2. From the new project wizard select the Windows category from the Visual C# category in the left tree view as shown in Figure 51. Next select a console Application and give a Name to the Project, in this example it is “Hello World”. Click OK to create the project.

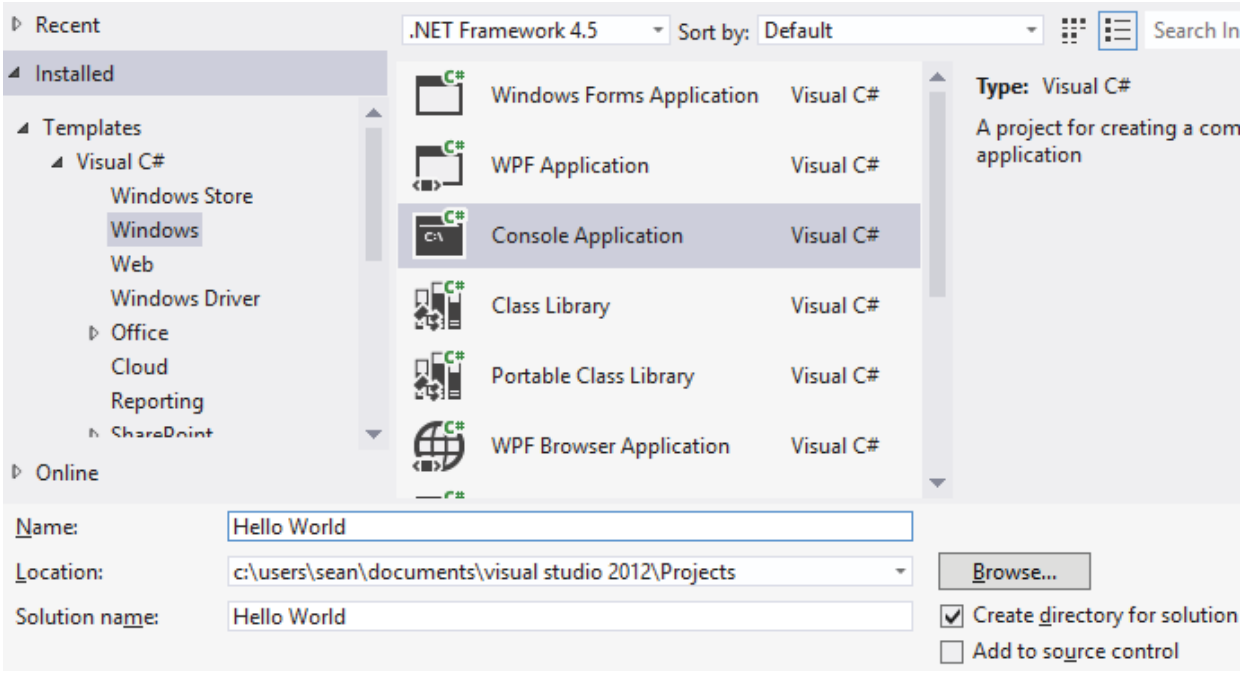

Figure 51 Visual Studio – New project creation dialog for "Hello Word" project.

3. The next two steps are to add the MySQL library to the project in order to communicate with the database. First a reference to the MySQL library needs to be added, this can be done by right clicking on the references section of the project and clicking add as shown in Figure 52. This will show the Reference Manager.

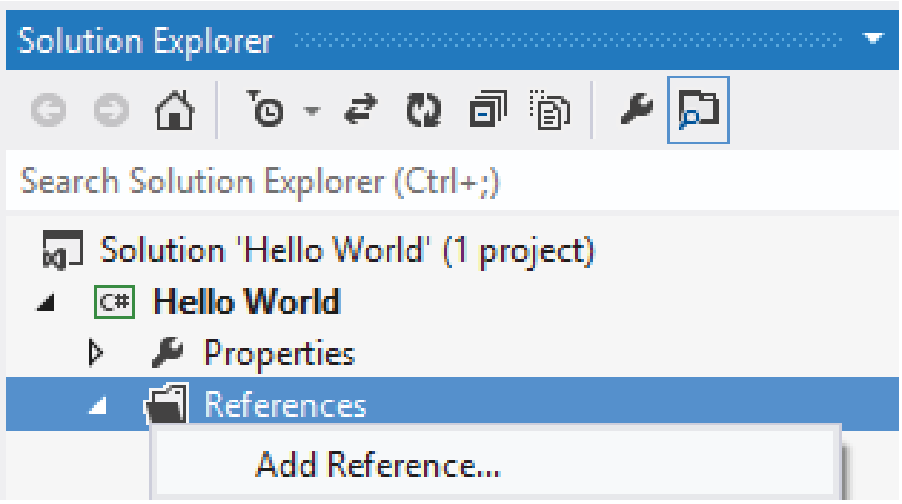

Figure 52 Visual Studio - Opening reference manager for "Hello World" project.

4. In the Reference Manager select Browse on the left hand side, and then click browse at the bottom of the screen, this will bring up a dialogue to select the location of a library. Locate the "MySQL.Data.dll" library which is by default installed in the following directory for 64 bit systems:

   "C:\Program Files (x86)\MySQL\MySQL Connector Net 6.6.5\Assemblies\v4.0"

   Or in this directory for 32 bit systems:

"C:\Program Files\MySQL\MySQL Connector Net 6.6.5\Assemblies\v4.0"

After locating the file the reference manager should appear like Figure 53. Click OK to confirm adding the new reference. The MySQL.Data library should appear in the references section of the project.

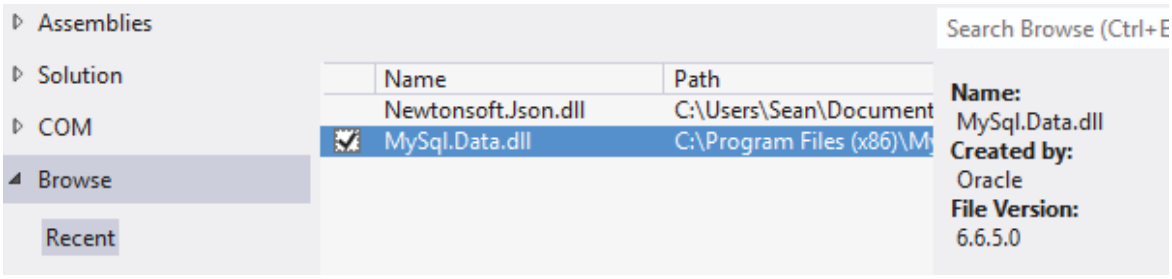

Figure 53 Visual Studio - Adding MySQL.Data reference to "Hello World" project.

5. The final step is to add code to test the MySQL library and MySQL database, the following code was added to program.cs in the project.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MySql.Data.MySqlClient;

namespace Hello_World
{
    class Program
    {
        static void Main(string[] args)
        {
            MySqlConnection conn = null;
            try
            {
                string conns = @"server=localhost;userid=user12;
                                 password=34klq*;database=mydb";
                conn = new MySqlConnection(conns);
                conn.Open();
                Console.WriteLine("MySQL Connection sucess!");
                Console.WriteLine("Sever version: {0}", conn.ServerVersion);
            }
            catch (MySqlException ex)
            {
                Console.WriteLine("Error: {0}", ex.ToString());
            }
            finally
            {
                if (conn != null)
                {
                    conn.Close();
                }
            }
            Console.Read();
        }
    }
}
```

This code uses the MySQL library to create a connection to the database running locally on the computer. The connection parameters are stored in a string called "conns", which are the server to connect to, user id, password, and database to connect to.

5.1.3.3 Project Debugging

1. When a debug session is started in Visual Studio execution of the program continues until a break point is reached. To add a breakpoint click in the column next to the line required as shown in Figure 54.

Figure 54 Visual Studio - Adding a breakpoint to the "Hello World" project.

2. A debug session can be started by clicking the button shown in Figure 55. This will execute the program until a break point is reached. The breakpoint chosen in the previous step will stop the program execution at the line chosen.

Figure 55 Visual Studio - Button for starting debug session.

3. The state of the Visual Studio interface after the breakpoint is reached is shown in Figure 56. The menu bar contains buttons for controlling the program flow such as stepping and continuing. Pressing the Continue button will resume program execution.

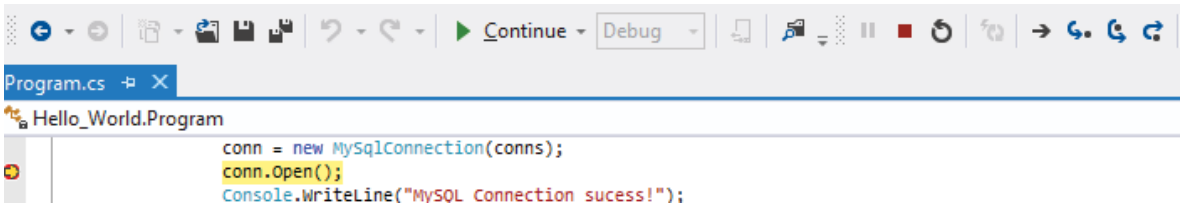

Figure 56 Visual Studio - Debugger stopped at "Hello World" breakpoint.

4. The output of the program is shown in Figure 57, the error produced is due to the absence of a MySQL server which will be setup in section 5.2.1. However the error does indicate that the MySQL library is working correctly as it attempted to create a connection. This program can be used to test the configuration performed in section 5.2.1.

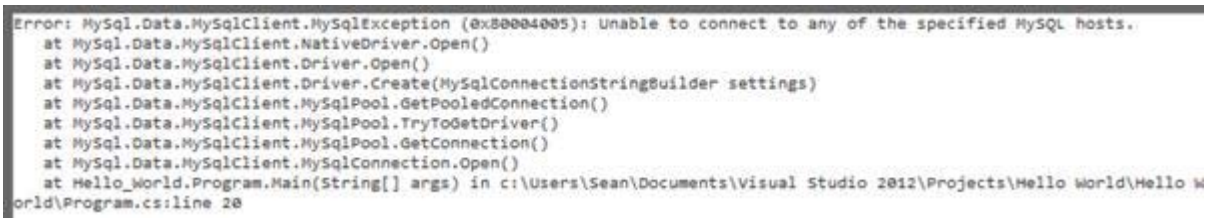

Figure 57 Visual Studio - "Hello World" Executable output.

This concludes the setup of Visual Studio for developing the required services for the IoT platform with a working example.

5.2 Software Configuration

Software packages and operating systems are required for the IoT integrated platform to operate, which require specific configuration. The server requires web hosting capability to display sensor data and a database for storing sensor information, which is provided by the Windows Apache and MySQL and PHP (WAMP) software package. The IoT gateway requires a customised Linux operating for networking functionality. In order to have secure communication between the IoT gateway and the server a secure communication software package called Open Virtual Private Network (OpenVPN) was used. The following sections give details on how to configure the various software components to create the supporting architecture for an IoT integrated platform.

5.2.1 Windows Apache MySQL and PHP (WAMP)

To provide web hosting and database capability is the WAMP software package was chosen, however after a default installation some parameters or settings need to be changed for the IoT integrated platform. A windows based server with a permanent internet connection is required to run the WAMP software package.

5.2.1.1 WAMP Installation

The version of WAMP used was 2.22E and can be obtained from [79]. Install WAMP using the default settings.

WAMP requires the Visual Studio redistributable package to be installed, this can be obtained from [80].

5.2.1.2 WAMP Configuration

Configuration to setup a WAMP service running on a server is given in the following steps.

1. Run the WAMP executable which will place an icon in the system tray as shown in Figure 58.
2. WAMP by default does not allow external IP addresses to connect to the Apache server. To allow external IP addresses to connect right click the WAMP icon and select "Put Online" as shown in Figure 58.

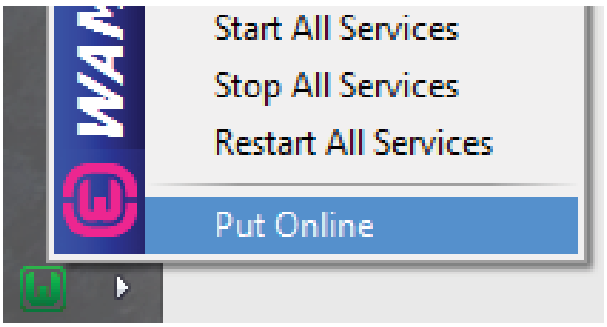

Figure 58 WAMP - "Put Online" to enable access to WAMP from external IP addresses.

3. PHP Sockets are required for the implementation, to enable them right click the WAMP icon then navigate to PHP -> PHP Extensions and click on "php_sockets" as shown in Figure 59.

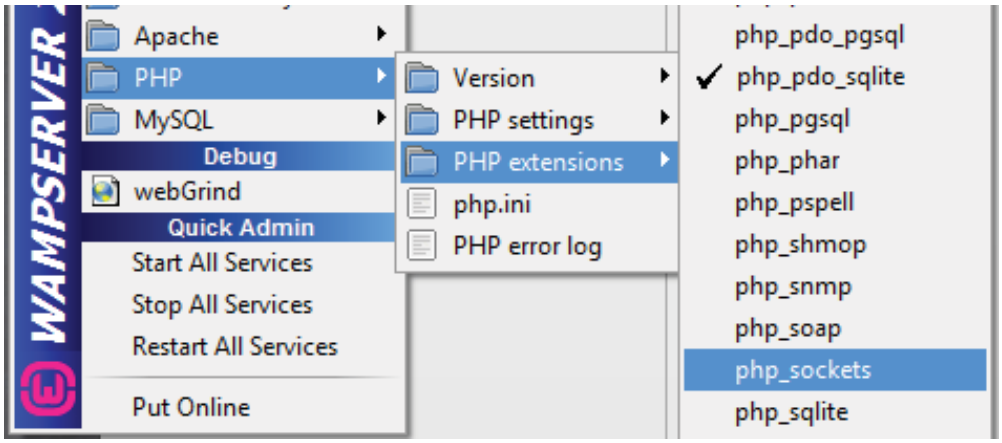

Figure 59 WAMP - Enabling "php_sockets" from the menu.

4. The MySQL database is by default unsecured, and therefore a password for the root user needs to be set. Right click the WAMP icon and select MySQL then MySQL Console to start a MySQL console session as shown in Figure 60.

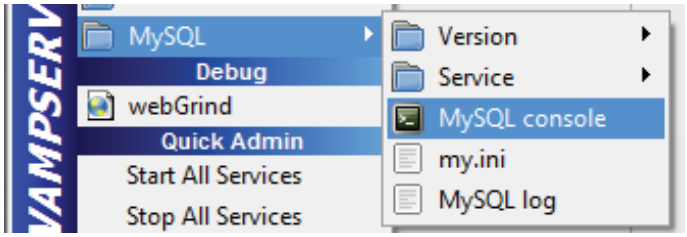

Figure 60 WAMP - opening a MySQL console session.

5. Use the following commands to set the password for the root user.

```
update mysql.user SET password=password("password") where user="root";
flush privileges;
quit
```

The update statement selects the root user from the user table in the database and updates the password field to password. In this case password is an example, it is recommended to use a complex password to improve security. The flush command ensures that all the fields are update and the quit command terminates the MySQL session. An example of successfully setting the root password is shown in Figure 61.

```
mysql> update mysql.user SET password=password("password") where user="root";
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql> quit_
```

Figure 61 MySQL - successfully changing the root users' password.

6. In order for phpMyAdmin to work with the database it needs root access, therefore it needs the root password. Change following line in file
   "C:\wamp\apps\phpmyadmin3.5.1\config.inc.php"
   to reflect changes to the MySQL root password.

```
$cfg['Servers'][$i]['password'] = 'password';
```

   Replace "password" with the password set in the previous step.

7. "phpMyAdmin" is used to administer the database which can be accessed from "http://127.0.0.1/phpmyadmin/". There were problems accessing "phpMyAdmin" from "http://localhost/phpmyadmin/" in Internet Explorer.

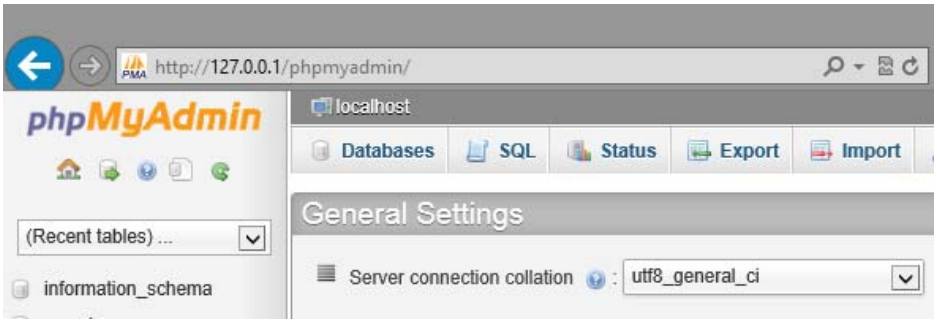

Figure 62 WAMP - phpMyAdmin in Internet Explorer.

This concludes the setup of the WAMP server for the IoT integrated platform. To complete the IoT integrated platform custom software for receiving sensor data, and displaying this data on a website was developed, details of this software is in the section 5.6.2.

5.2.2 Open source embedded Linux installation

OpenWRT is an open source embedded Linux created for routers and can be freely downloaded and modified. It contains a number of useful software packages to create an IoT Application gateway. Remote debugging of customised software can be achieved using a free toolset provided by OpenWRT coupled with the Eclipse IDE. Communication between the IoT application gateway and server needs to be secure, and OpenWRT provides secure VPN connectivity capabilities from its software packages. The next section gives specific details on compiling a customised firmware image, this image is hosted on the google code website for the project so that the lengthy process to create the firmware image does not need to be repeat. The last section elaborates on installing the custom firmware image on the WRT54GL router in order to provide a platform to develop an IoT Application gateway.

5.2.2.1 OpenWRT Compilation

OpenWRT has many features some of which are not needed for the IoT implementation. A custom firmware image was compiled to create a minimal operating system for the IoT application gateway. The following steps are based on the guides available on the OpenWRT website [81] [82]. This process

requires Linux as the windows environment requires significant changes to the toolset source code. The version of Linux used to create the firmware was Ubuntu 12.04 64bit.

1. To create the environment to cross compile and create a firmware image certain software packages are required, these can be installed using the following command.

```
sudo apt-get install build-essential subversion libncurses5-dev zlib1g-dev gawk gcc-multilib flex git-core gettext
```

2. Next a directory for the required files needs to be created this was done in the home directory using the following commands.

```
mkdir openwrt
cd openwrt
```

3. The source code can now be checked out from the OpenWRT subversion using the following command.

```
svn co svn://svn.openwrt.org/openwrt/branches/backfire
```

4. After the source code has been checked out the feeds need to be updated and installed. Feeds are the source code to software packages that will be installed into the firmware image. The following command will update and install the required feeds.

```
./scripts/feeds update
./scripts/feeds install
```

5. The OpenVPN and secure FTP must be installed using the scripts in the previous step using the following.

```
./scripts/feeds install openvpn
./scripts/feeds install openssh-sftp-server
```

6. A menu system is provided to configure the parameters needed to build the firmware as there are many software packages and platforms available. To start the menu use the following command.

```
make menuconfig
```

7. The platform the firmware is to be built for is selected in “Target System”. The WRT54GL router is based on the BCM947xx chip therefore the target system “Broadcom BCM947xx/953xx” must be selected.
8. Enabling the toolchain option allows for debugging options to be selected. It can be enabled by the selecting the “Advanced configuration options” item and then selecting “Toolcahin options”.
9. IPv6 support is added by installed the IPv6 kernel module. This is added to the firmware by selecting “Kernel Modules” then “Network Support”, located “kmod-ipv6” and select it until an

asterisk (“*”) appears next to it. If an “M” appears next to it the package will be compiled but not installed into the firmware. This must be done for all the following steps.

10. Select the OpenVPN software package which is located in “Network” then “VPN”. This is needed to create a secure connection between the IoT application gateway (router) and the server.
11. A secure file transfer protocol (SFTP) service is required for uploading software to be debugged from the Eclipse IDE. The package “openssh-sftp-server” provides a SFTP service, it can selected in “Network”, then “SSH”.
12. To debug software remotely on the router using eclipse the software package “gdbserver” is required. This can be selected in the “Utilities” section.
13. Save the above configuration, which will be used to build a customised firmware for the WRT54GL router.
14. The build process is initiated using the following command.

```
make
```

15. This will take a considerable amount of time as all the components will be downloaded and compiled to create the firmware image. The resulting firmware image is located in the bin directory for example: ~/openwrt/backfire /bin/brcm2.4/openwrt-brcm2.4-squashfs.trx

This process will produce a firmware image, a copy has been placed in the google code repository created for this thesis.

5.2.2.2 OpenWRT installation

The following steps install the firmware image created in the previous section onto the WRT54GL router.

1. Locate the custom OpenWRT firmware image for the WRT54GL router created in the previous section, or download a pre made custom firmware at [83].
2. Connect the power adapter to power the WRT54GL router. Connect an Ethernet cable between port 1 of the WRT54GL router and the computer. Set the IP address for the Ethernet adapter on the computer to 192.168.1.2 with a subnet of 255.255.255.0 in order to communicate with the WRT54GL router. See [84] for details on IP Address configuration in windows. By default the ip address of the WRT54GL router is 192.168.1.1. Using a web browser navigate to http://192.168.1.1/ to access the configuration of the router in order to upload the firmware. The “Firmware upload” page is located in the “Administration” section as shown in Figure 63. Click the “Browse” button and locate the firmware file from the previous

step. Click the Update button to start the upload and flashing of the firmware which is shown in Figure 63.

Figure 63 OpenWRT - Uploading and flashing OpenWRT on WRT54GL router.

3. The lights marked “power” and “dmz” will flash while the router boots into OpenWRT. When the power light is constantly on the boot process is finished. The initial state of the firmware is there is no password set for the root user and a telnet service is running which makes the WRT54GL unsecure. The telnet service gives access to a shell, in which the root users’ password can be set. To set the password open PuTTY and connect to the router (the IP address remains the same – 192.168.1.1) using the “Telnet” connection type as shown in Figure 64. Click the Open button to establish a telnet connection to the WRT54GL router.

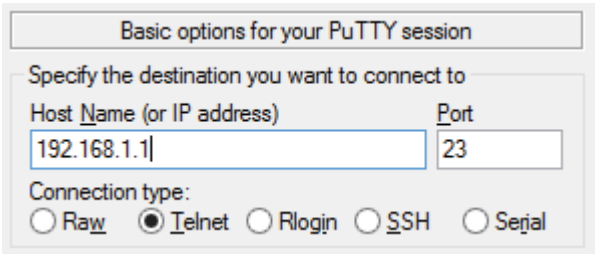

Figure 64 PuTTY settings for connecting to WRT54GL after OpenWRT installation.

4. If the telnet connection is successful a banner and prompt should appear in a new window as shown in Figure 65. To set the password use the “passwd” command, this will prompt the user

to enter a new password twice as shown in the bottom of Figure 65. Once the password is set the session can be ended using the “exit” command. After the session is closed the telnet service will stop and the WRT54GL router will be secure. A Secure Shell (SSH) service will be started in order to provide shell access which will require the password set previously.

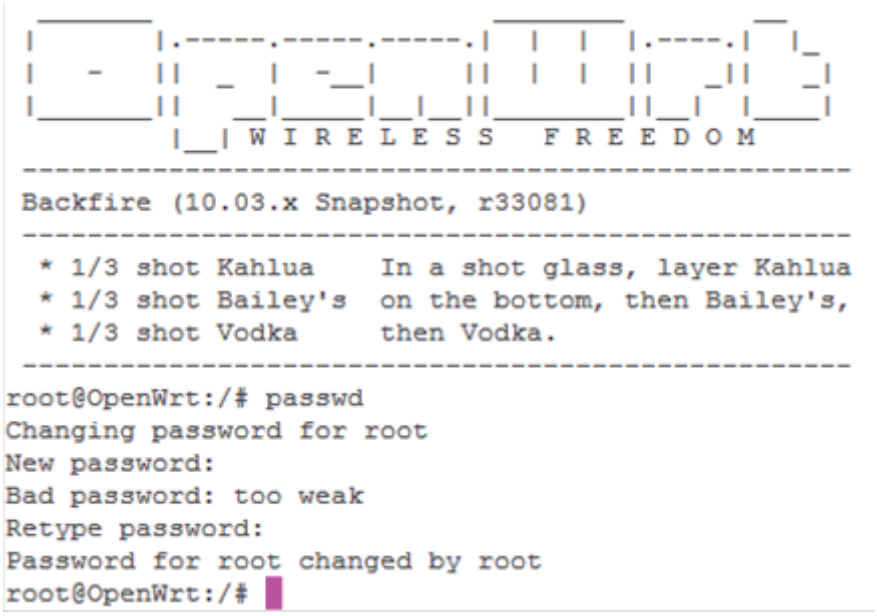

Figure 65 OpenWRT - telnet connection to set root password.

5. To connect to the SSH on the WRT54GL router the settings shown in Figure 66 should be used. This will create a secured shell session with the router. Click Open to start the session.

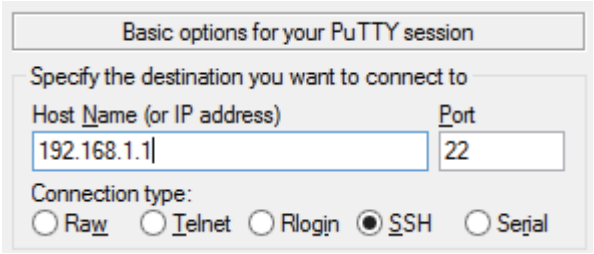

Figure 66 PuTTY settings for SSH access to the WRT54GL router.

6. The first time a secure session is made keys must be generated for the connection. This means the warning shown in Figure 67 will appear, click “Yes” to connect to the router.

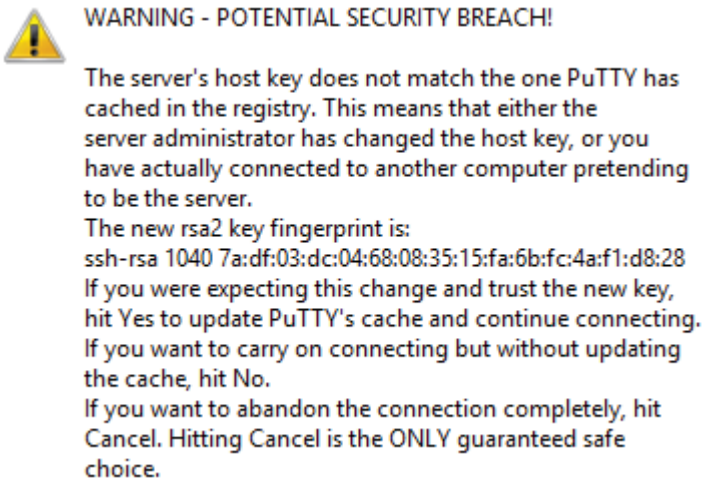

Figure 67 PuTTY key mismatch error.

7. Login to the router with the user "root" and the password set in Step 4. Figure 68 shows the result of a successful login and the shell prompt.

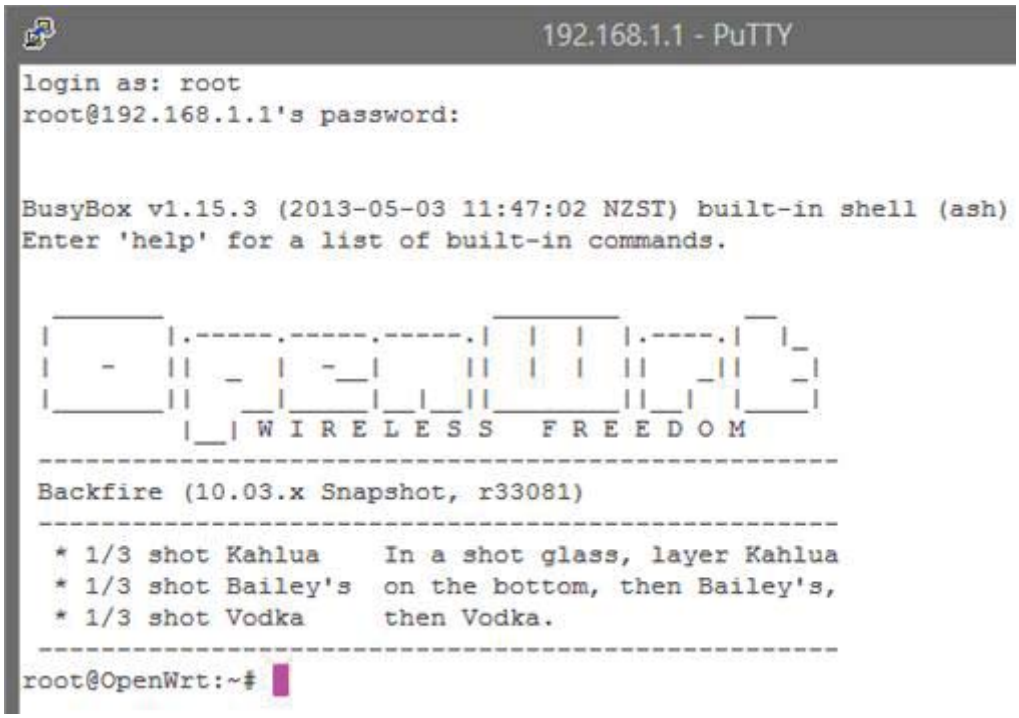

Figure 68 OpenWRT - Secure Shell session login.

8. The next steps are to allow the router to connect to the internet via a local network. An internet connection is required to connect to the server using OpenVPN. The IP configuration of the router needs to be changed in order to exist on the local network and use the internet connection on the local network. A dynamically assigned address is not preferable as the router needs to be at a known address to be accessible. The network settings are in the file "/etc/config/network" which can be edited using the vi command as following.

```
vi /etc/config/network
```

For further information on how to use vi see [85]. Figure 69 shows specific settings for connecting to a home network.

```
#### LAN configuration
config interface lan
        option type     bridge
        option ifname   "eth0.0"
        option proto    static
        option ipaddr   192.168.1.100
        option netmask  255.255.255.0
        option dns      192.168.1.2
        option gateway  192.168.1.2
```

Figure 69 OpenWRT - IP Configuration for joining local network.

9. Once the networking configuration file has been changed the networking interfaces need to be restarted in order for the configuration to take effect. The following command will restart the networking interfaces.

```
/etc/init.d/network restart
```

The PuTTY window can be closed as the new network configuration will end the SSH session. The router will now be at the IP Address assigned in the file. A new SSH session using PuTTY will need to be established using steps 5 and 6.

This concludes the Setup of OpenWRT on the WRT54GL router. In the next section OpenVPN will be configured on the router and server to create the base for the communication between the router and server to create an IoT integrated platform. The customised software developed to send sensor data from the router to the server requires the secure connection created by OpenVPN.

5.2.3 OpenVPN

OpenVPN is open source software that creates a Virtual Private Network, and runs on many platforms such as Linux and windows. OpenVPN has been ported to the OpenWRT, making it suitable to create a secure connection between an OpenWRT based router and a server. Sensor data can be transmitted via this secure connection as it supports IPv6. The following sections give details on installation and configuration of OpenVPN on the server and router.

5.2.3.1 OpenVPN Installation

OpenVPN version 2.2.2 was used and can be obtained from [86]. This is not the most current version of OpenVPN as the most current version was released after the system was developed. Due to changes in the functionality of the new version of OpenVPN it cannot be used, and OpenWRT does not support the new version. OpenVPN must be installed with the default settings in order to follow the steps in the following sections.

5.2.3.2 OpenVPN Certificate and Key Generation

Certificates and keys are required to secure the connection between the IoT application gateways and the server. The following steps give details on creating a Certificate Authority (CA) and issuing certificates and keys for the server and clients (IoT application gateways). The steps must be undertaken on the server to make configuration and distributing keys and certificates a simple process. The steps are based on the guide provided at [87].

1. Open a command prompt by pressing the windows key and “r”. Put “cmd” into the textbox as shown in Figure 70 and click ok, this will open a command prompt.

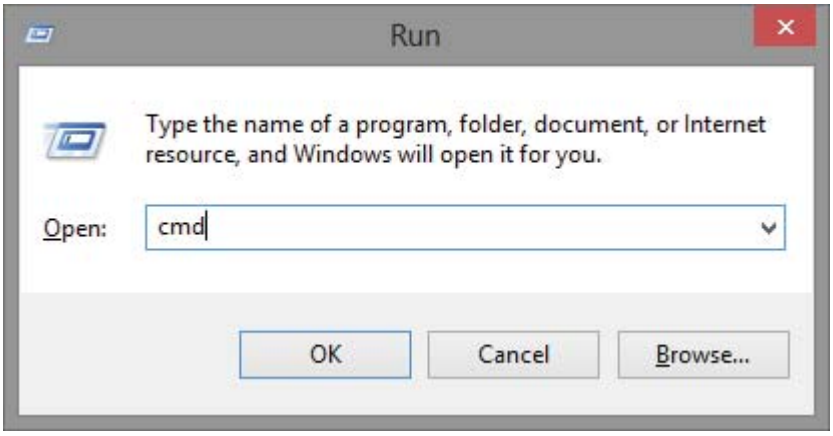

Figure 70 OpenVPN - Starting a command prompt.

2. In the command prompt change the directory to the “easy-rsa” subdirectory and run the batch file “init-config” as shown in Figure 71. This will create a batch file called “vars.bat” which contains variables needed to generate the certificates.

```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>cd "C:\Program Files\OpenVPN\easy-rsa"

C:\Program Files\OpenVPN\easy-rsa>init-config

C:\Program Files\OpenVPN\easy-rsa>copy vars.bat.sample vars.bat
        1 file(s) copied.
```

Figure 71 OpenVPN - Changing to easy-rsa directory and running init-config.

3. The “vars.bat” file needs to be edited to change the variables at the end of the file. The following values are examples:
    a. set KEY_COUNTRY=NZ - this changes the country which the certificate is associated with, and it must be two letters.
    b. set KEY_PROVINCE=Manawatu – this changes the province/area which the certificate is associated with.
    c. set KEY_CITY=Palmerston North – this changes the town/city which the certificated is associated with.

d. set KEY_ORG=Massey University – this changes the organisation with the certificate is associated with.

All other variables can remain the same.

4. Running the vars.bat batch file will initialise the variables needed for the key and certificate creation. Figure 72 shows the result of running “vars.bat” and “clean-all.bat” which sets up the variables and directories for key and certificate creation.

```
C:\Program Files\OpenVPN\easy-rsa>vars

C:\Program Files\OpenVPN\easy-rsa>clean-all
        1 file(s) copied.
        1 file(s) copied.
```

Figure 72 OpenVPN - Running "vars.bat" and "clean-all.bat" to setup the environment.

5. The CA is generated using “build-ca.bat” as shown in Figure 73. The variables in “vars.bat” are used to automatically fill out the required parameters as shown by square brackets around the values. Pressing enter will use the default value in the square brackets. The “Common Name” and “Name” needs to be changed to the name of the server as shown at the bottom of Figure 73. After filling out the required parameters the CA will be generated in the directory “C:\Program Files\OpenVPN\easy-rsa\keys” with the file name being “ca.crt”.

```
C:\Program Files\OpenVPN\easy-rsa>build-ca
WARNING: can't open config file: /etc/ssl/openssl.cnf
Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
..............++++++
..++++++
writing new private key to 'keys\ca.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [NZ]:
State or Province Name (full name) [Manawatu]:
Locality Name (eg, city) [Palmerston North]:
Organization Name (eg, company) [Massey University]:
Organizational Unit Name (eg, section) [changeme]:
Common Name (eg, your name or your server's hostname) [changeme]:iots2is.org
Name [changeme]:iots2is.org
Email Address [mail@host.domain]:
```

Figure 73 OpenVPN - Running "build-ca.bat" to create the main key and certificate for the OpenVPN server.

6. The server certificate and key is generated using “build-key-server.bat” as shown in Figure 74. The parameter “iots2is.org” specifies the name of the key and certificate being generated. The values requested are the same as those in step 5 with two additional parameters which can remain default. After giving the required parameters a request to sign the certificate is presented. Pressing “y” will sign the certificate and pressing “y” again will commit the new

certificate to the CA database as shown in the bottom of Figure 74. The server certificate and key will now be in the keys directory with the file names “iots2is.org.ca” and “iots2is.org.key”.

```
C:\Program Files\OpenVPN\easy-rsa>build-key-server iots2is.org
WARNING: can't open config file: /etc/ssl/openssl.cnf
Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
.........++++++
.....................++++++
writing new private key to 'keys\iots2is.org.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [NZ]:
State or Province Name (full name) [Manawatu]:
Locality Name (eg, city) [Palmerston North]:
Organization Name (eg, company) [Massey University]:
Organizational Unit Name (eg, section) [changeme]:
Common Name (eg, your name or your server's hostname) [changeme]:iots2is.org
Name [changeme]:iots2is.org
Email Address [mail@host.domain]:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
WARNING: can't open config file: /etc/ssl/openssl.cnf
Using configuration from openssl-1.0.0.cnf
Loading 'screen' into random state - done
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'NZ'
stateOrProvinceName   :PRINTABLE:'Manawatu'
localityName          :PRINTABLE:'Palmerston North'
organizationName      :PRINTABLE:'Massey University'
organizationalUnitName:PRINTABLE:'changeme'
commonName            :PRINTABLE:'iots2is.org'
name                  :PRINTABLE:'iots2is.org'
emailAddress          :IA5STRING:'mail@host.domain'
Certificate is to be certified until Apr 21 01:12:47 2023 GMT (3650 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Figure 74 OpenWRT - Generating server certificate and key.

7. The client (IoT Application Gateway) certificate and key is generated with “build-key.bat” as shown in Figure 75. The “router1” parameter specifies the file name of the certificate generated. The procedure is similar to the previous step however the “Common name” and “Name” must be changed to the name of the client, “router1” in this case. The certificate must also be signed and committed to the CA database as shown at the bottom of Figure 75. This step must be repeated for each client that needs to connect to the server with a different file name, “Common Name” and “Name” for each client.

```
C:\Program Files\OpenVPN\easy-rsa>build-key router1
WARNING: can't open config file: /etc/ssl/openssl.cnf
Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
.....................................................++++++
.....................................................++++++
writing new private key to 'keys\router1.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [NZ]:
State or Province Name (full name) [Manawatu]:
Locality Name (eg, city) [Palmerston North]:
Organization Name (eg, company) [Massey University]:
Organizational Unit Name (eg, section) [changeme]:
Common Name (eg, your name or your server's hostname) [changeme]:router1
Name [changeme]:router1
Email Address [mail@host.domain]:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
WARNING: can't open config file: /etc/ssl/openssl.cnf
Using configuration from openssl-1.0.0.cnf
Loading 'screen' into random state - done
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'NZ'
stateOrProvinceName   :PRINTABLE:'Manawatu'
localityName          :PRINTABLE:'Palmerston North'
organizationName      :PRINTABLE:'Massey University'
organizationalUnitName:PRINTABLE:'changeme'
commonName            :PRINTABLE:'router1'
name                  :PRINTABLE:'router1'
emailAddress          :IA5STRING:'mail@host.domain'
Certificate is to be certified until Apr 21 01:37:55 2023 GMT (3650 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Figure 75 OpenVPN - Generating IoT application gateway certificates and keys.

8. The final step is to generate the Diffie Hellman parameters using "build-dh.bat" as shown in Figure 76.

```
C:\Program Files\OpenVPN\easy-rsa>build-dh
WARNING: can't open config file: /etc/ssl/openssl.cnf
Loading 'screen' into random state - done
Generating DH parameters, 1024 bit long safe prime, generator 2
This is going to take a long time
.....................................................................
```

Figure 76 OpenVPN - Generating Diffie Hellman parameters.

If more certificates or keys need to be generated after the command prompt is closed, "vars.bat" must be run in the new command prompt to initialise the environment and then Step 7 can be followed to generate the required certificate and key.

5.2.3.3 OpenVPN Server Configuration

OpenVPN provides a service on the server to allow VPN connections to be made to the server. This service requires configuration which is given in the following steps.

1. A sample configuration for a server is contained in
   "C:\Program Files\OpenVPN\sample-config\server.ovpn"
   copy this file to the directory
   "C:\Program Files\OpenVPN\config"
   This is the location where the OpenVPN graphical user interface (GUI) looks for configurations as multiple configurations can be present in that directory.
2. Edit the server configuration file located in the config directory from the previous step and make the following changes:
   a. In order to allow for forwarding through routers the protocol OpenVPN uses must be changed to TCP. Change the protocol from UDP to TCP by commenting and uncommenting as following:

```
proto tcp
;proto udp
```

   b. For IPv6 to operate with OpenVPN the type of tunnel must be changed from a tunnel that operates on the transport layer (tun) to a tunnel that operates on the data layer (tap). Change the tunnel type from a "tun" to "tap" by commenting and uncommenting as following:

```
dev tap
;dev tun
```

   c. The locations of the CA, certificate and key for the server are required. The location of these files is based on the previous section. Due to OpenVPN being multiplatform the paths must be specified in quotation marks and the slash character must be escaped. Change the locations as following:

```
ca "C:\\Program Files\\OpenVPN\\easy-rsa\\keys\\ca.crt"
cert "C:\\Program Files\\OpenVPN\\easy-rsa\\keys\\iots2is.org.crt"
key "C:\\Program Files\\OpenVPN\\easy-rsa\\keys\\iots2is.org.key"
```

   d. The location of the Diffie Hellman parameters is required for the server. The location of this files is based on the previous section. Change the location as following:

```
dh "C:\\Program Files\\OpenVPN\\easy-rsa\\keys\\dh1024.pem"
```

   e. Due to problems with packet fragmentation in windows the following option should be appended to the bottom of the configuration.

```
mssfix 1000
```

This completes the configuration, save the configuration with the above changes.

3. Run the OpenVPN GUI application, it will appear in the task tray as shown in Figure 77. Right click the icon and “Connect” should appear at the top of the menu as shown in .Click on “Connect” menu item to start the OpenVPN server.

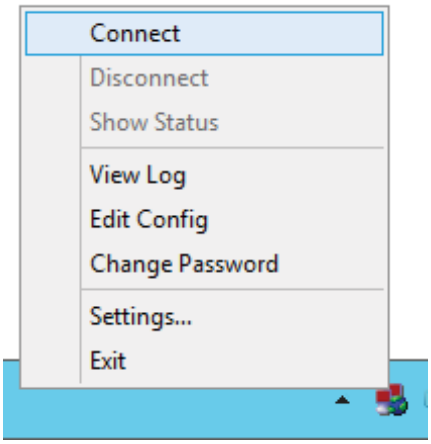

Figure 77 OpenVPN GUI application on the server.

4. A status window will appear showing the connection progress as shown in Figure 78. If the tunnel is successfully created this window will close with no error messages and the icon in the task tray will turn green.

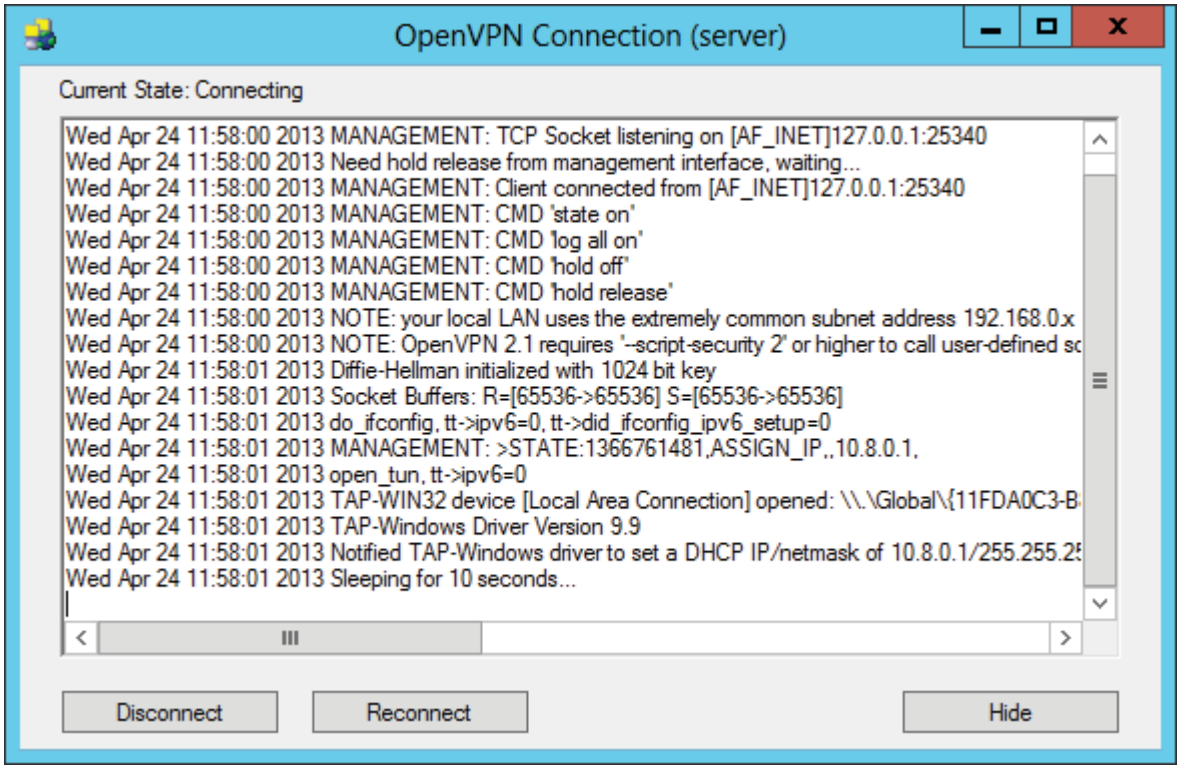

Figure 78 OpenVPN - Successfully creating a tunnel service.

5. The status window is useful for displaying when a client connects and troubleshooting any problems. It can be shown by clicking on the OpenVPN GUI icon in the task tray then clicking “Show Status” as shown in Figure 79, this will display the window shown in Figure 78.

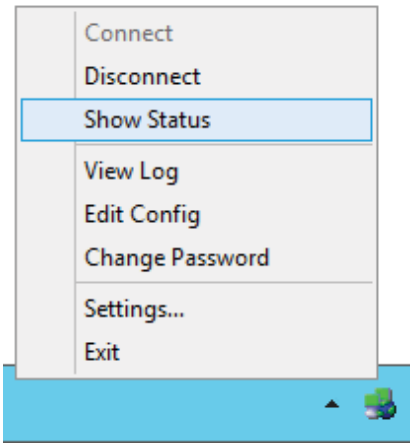

Figure 79 OpenVPN - Method to show status window.

6. OpenVPN version 2.2.2 does not support auto configuration of IPv6 addresses, therefore the IPv6 address for the server must be configured. The following steps show how to configure the address in Windows 7 or Windows 8. First open the “Network and Sharing Center by right clicking on the network icon in the system tray as shown in Figure 80.

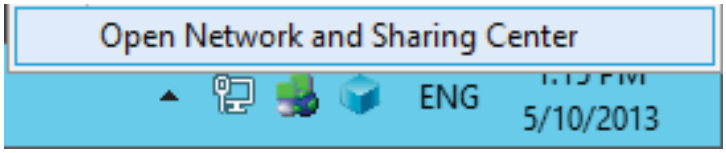

Figure 80 Opening networking and Sharing Center

7. In the “Network and Sharing Center” choose adapter settings in the column on the left side of the screen, this will open a window containing a list of the network adapters installed on the computer. Locate the network adapter for the OpenVPN tap driver as shown in Figure 81 and right click on it then choose properties. This will show the properties dialog for the adapter.

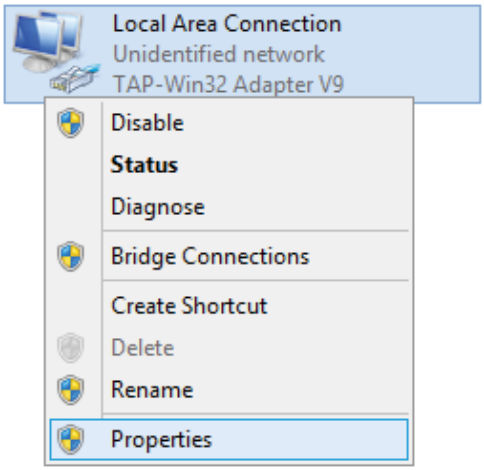

Figure 81 Opening properties for OpenVPN tap driver.

8. In the adapter settings dialog for the OpenVPN tap driver choose “Internet Protocol Version 6 (TCP/IPv6)” from the list and click on the “Properties” button. This will show a window to set the IPv6 address for the adapter.

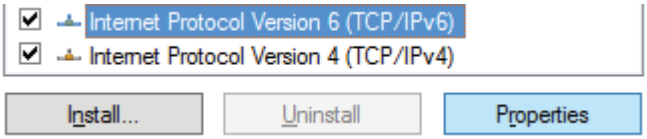

Figure 82 Selecting IPv6 properties.

9. The IPv6 address and subnet for the adapter can be set by clicking “Use the following IPv6 address” and then filling out the appropriate text boxes as shown in Figure 83.

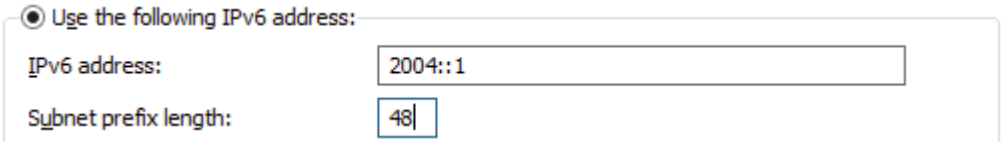

Figure 83 Changing IPv6 address for OpenVPN tap driver.

The OpenVPN GUI application will need to be run and the connection started if the computer is restarted. The next section give details on configuring OpenVPN on the router in order to connect to the OpenVPN service setup on the server.

5.2.3.4 OpenVPN Client Configuration

The configuration of OpenVPN on the router is similar in some aspects to the server, however due to a Unified Configuration Interface (UCI) [88] used in OpenWRT the specification of parameters is different. The following steps provide details on configuring OpenVPN running on OpenWRT to connect to the OpenVPN server setup in the previous section.

1. In order to create a secure connection between the router and server the certificates generated on the server (in the beginning of the previous section) must be available on the

router. The following files are needed, the certificate authority certificate which is “ca.crt” and the certificate and key file for the client. To transfer the certificates to the router the certificate files were placed in a temporary directory “tmp” in the WAMP “www” directory. This made them accessible through the apache http server in WAMP. Figure 84 shows the commands to download the certificates onto the router, which was connected via the LAN to the server. First the directory is changed to the “/etc/openvpn” directory which is used to store the configuration, certificate and key files. Next the “wget” command was used to download the files form the WAMP server. Due to the server being on the local network the IP address of the server was used, this may need to be changed based on the network setup.

```
root@OpenWrt:~# cd /etc/openvpn
root@OpenWrt:/etc/openvpn# wget http://192.168.1.11/tmp/ca.crt
Connecting to 192.168.1.11 (192.168.1.11:80)
ca.crt               100% |*******************************|  1468   0:00:00 ETA
root@OpenWrt:/etc/openvpn# wget http://192.168.1.11/tmp/router1.crt -O client.crt
Connecting to 192.168.1.11 (192.168.1.11:80)
client.crt           100% |*******************************|  4131   0:00:00 ETA
root@OpenWrt:/etc/openvpn# wget http://192.168.1.11/tmp/router1.key -O client.key
Connecting to 192.168.1.11 (192.168.1.11:80)
client.key           100% |*******************************|   916   0:00:00 ETA
root@OpenWrt:/etc/openvpn# ls
ca.crt      client.crt  client.key
```

Figure 84 OpenVPN - Downloading certificates and keys onto WRT54GL router.

2. A configuration file provided by OpenVPN is present in the location “/etc/config/openvpn”, this needs to be edited in order to connect to the server. Editing software called “vi” can be used to edit the configuration using the following command. Details on how to use “vi” can be found at [85].

```
vi /etc/config/openvpn
```

3. The configuration file contains two sample configurations, a server configuration and a client configuration. Locate the client configuration by finding the following.

```
##############################################
# Sample client-side OpenVPN 2.0 uci config  #
# for connecting to multi-client server.     #
##############################################
```

The settings following the lines above need to be altered to configure the client.

   a. Both the sample configurations are disabled by default, therefore to use the client configuration it must be enabled by changing enabled from 0 to 1 as following.

```
option enabled 1
```

   b. The tunnel type needs to be changed to tap from tun to enabled IPv6 support. To do this comment the tun line and uncomment the tap line as shown below.

```
option dev tap
#option dev tun
```

c. To allow forwarding through a router the TCP protocol must be used by commenting the UDP protocol and uncommenting the TCP protocol as shown below.

```
option proto tcp
#option proto udp
```

d. The address of the OpenVPN server is set as following

```
list remote “iots2is.org 1194”
```

Where “iots2is.org” is the host name of the server and 1194 is port. The port must correspond with the port number set in the server configuration (port 1194 is the default port).

e. To fix problems with fragmentation in windows the following option should be appended to the end of the configuration file.

```
option mssfix 1000
```

f. Due to the lack of IPv6 address auto configuration in this version of OpenVPN a script is used to set the IPv6 address of the router. The following line executes a script “up.sh” located in the “/etc/openvpn” directory. This script will be created in future steps.

```
option up /etc/openvpn/up.sh
```

g. The following option should be appended in order for the script to have the permission to change the IPv6 address.

```
option script_secturiy 2
```

4. A script to change the IPv6 address of the router can be created using “vi” with the following command.

```
vi /etc/openvpn/up.sh
```

The following lines can then be added to the script in “vi”

```
#!/bin/sh
ifconfig add %2 2004::2
```

The first line indicates that the file is a script, and the second line changes the IPv6 address of the tunnel adapter. The “ifconfig” command on the second line is given three parameters, the first is the command “add”, the second is the name of the network adapter, and the third is the address to add to the network adapter. The name of the tunnel adapter is passed to the script as a parameter, and this parameter is accessed using “%2”, this ensures that the correct adapter is given the IPv6 address as the name of the adapter may change. The IPv6 address given to the router must be unique to the router meaning that the script file on each router must contain a different IPv6 address.

5. The configuration of OpenVPN for the router is now complete and the OpenVPN client service can be started using the following command.

```
/etc/init.d/openpvn start
```

   The status of the client service can be checked using the “logread” command which will display a log file containing status information from various services including OpenVPN.

6. If the configuration and there is a functional network connection (via the internet or LAN) the router will successfully connect to the server to create a secured VPN connection. Figure 85 shows the output in the status window on the server of a successful connection form OpenVPN running on a router.

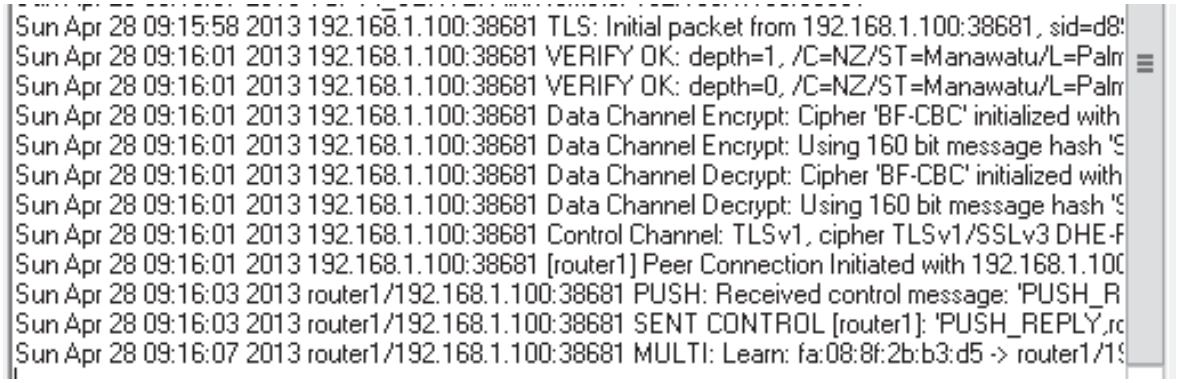

Figure 85 OpenVPN - sever status window showing successful router connection.

A common problem that occurs is that the router will rapidly connect and disconnect, which may be due to the router have an incorrect date and time. The WRT54GL does not have a real time clock that can maintain the time after it has been powered off. This means the time has to be set using the internet, which may fail in some cases. If the time is not correctly set the certificates used in OpenVPN will not be valid as they are only valid after the date they are issued for security reasons.

This concludes the setup required to configure the software packages required by the custom developed software to create and integrated IoT platform.

5.2.4 XBee Configuration

The XBee modules need to be configured in order to establish a ZigBee network. This network is then accessible via the coordinator, which will be connected to the IoT gateway. There are two main types of configuration which are End Device and Coordinator. The End Devices requires configuration for sensor input and to join a ZigBee network. The Coordinator requires configuration for establishing ZigBee network and serial communication parameters. The following sections give details on configuring the End Devices and Coordinators.

5.2.4.1 XBee Configuration Software

Specialised software called “X-CTU” is provided to configure XBee modules, it can be downloaded from [89]. When run the X-CTU program presents a windows that consists of four tabs, of which “PC Settings”, and “Modem Configuration” are used to configure an XBee module. The other tabs “Range Test”, and “Terminal” are for range testing and directly communicating with the modem respectively, these are not used for configuration.

The “PC Settings” tab on the left side of Figure 86 is used to configure the serial connection to the XBee module. The following parameters need to be selected:

- The com port that the XBee Module is connected to which this appears in the “Select Com Port” list box. The Sparkfun XBee Explorer USB appears as “USB Serial Port (COM…)” in the list. The com port number is usually randomly assigned when the Explorer is connected to the computer.
- The baud rate that the XBee Module uses. This is selected using the drop down list next to “Baud”. The default baud rate for an XBee module is 9600. The coordinator that connects to the OpenWRT router will use a baud rate of 115200.
- If API mode is enabled on the XBee module “Enable API” needs to be checked. API mode can also use escaped characters in which case “User escape characters (ATAP = 2)” needs to also be checked. The coordinator that connects to the OpenWRT router uses API mode and escaped characters.
- All other settings do not need to be changed as they remain default.

To test the serial connection click the “Test/Query” button, and a message box will appear with various parameters of the connected module if the communication is successful.

The “Modem Configuration” tab on the right side of Figure 86 is used to configure the XBee module by displaying the modules configuration. The “Read” button reads the current XBee module configuration which is then displayed in the tree view. The “Write” button writes the configuration settings displayed in the tree view. There are several hardware variants of the XBee modules, the series 2 XBee module appears as “XB24-B” in the drop down box under “Modem: …”. For each hardware variant there are various firmware options, this is displayed in the drop down box under “Function Set”. The settings in the tree view are arranged into sections, and each setting is formatted with the value in brackets followed by the name. A setting can be changed by clicking on it which displays a text box where the value can be modified.

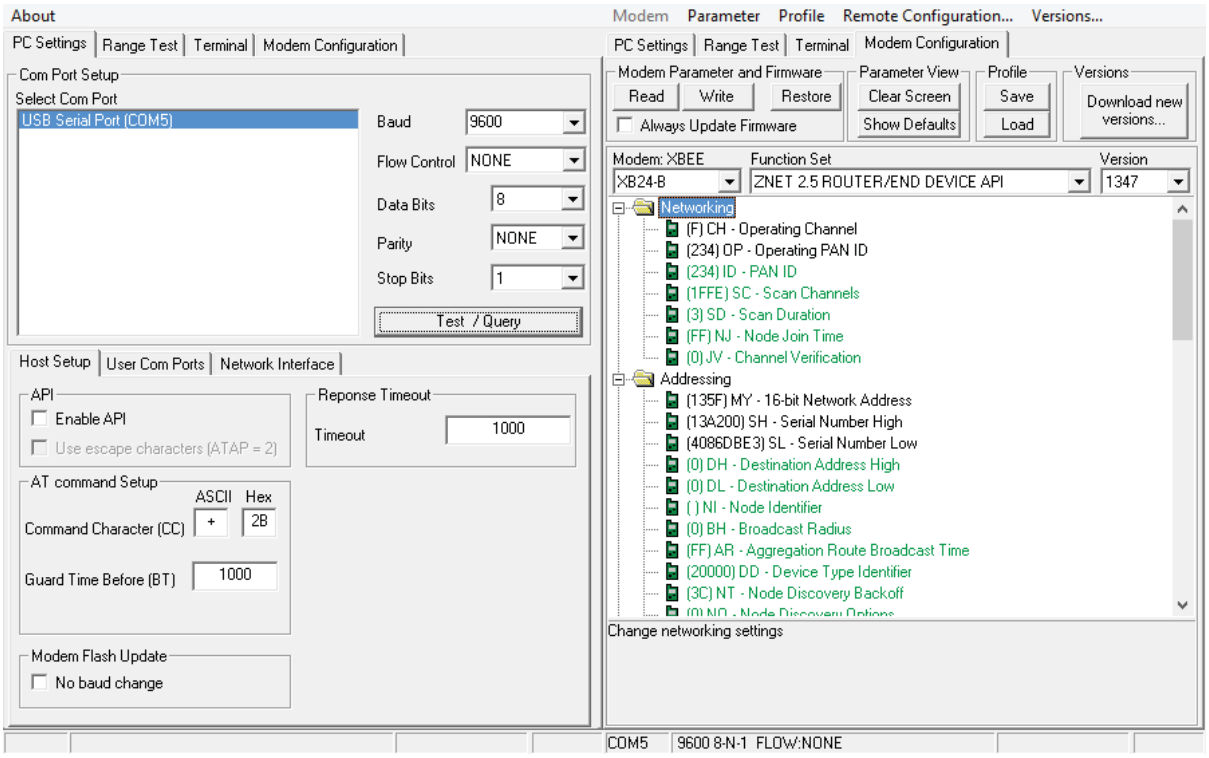

Figure 86 X-CTU - XBee module configuration software showing connection settings and module settings.

5.2.4.2 XBee Coordinator Configuration

A Coordinator is required to create a ZigBee network, and from the coordinator the nodes on the ZigBee network can be accessed. The XBee module provides a serial interface which can be used to configure the XBee module or communicate with the ZigBee network. The following steps show how to establish a ZigBee network and configure the communication over a serial connection to the router.

1. Setup the serial connection in the “PC Settings” by selecting the com port and baud rate. Test the serial connection using the “Test/Query” button to ensure that the XBee module is connected.
2. The coordinator firmware needs to be selected for the XBee module. Click on the “Modem Configuration” tab, select “XB24-B” from the drop box under “Modem: …”, and select “ZNET 2.5 COODINATOR API” from the drop box under “Function Set” as shown in Figure 87. This will populate the tree view with settings for an XBee Coordinator.

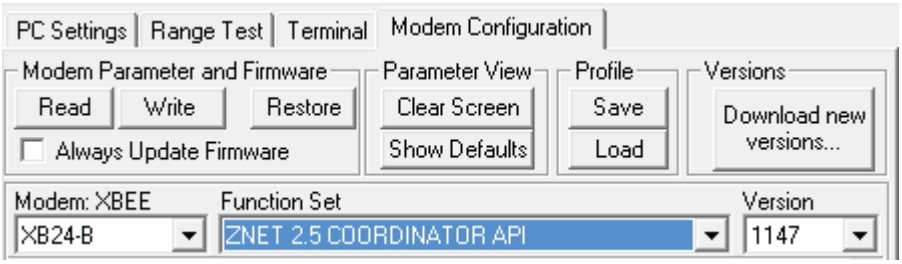

Figure 87 X-CTU - Selecting module type and firmware for coordinator.

3. ZigBee uses a unique 16 bit identifier to determine which network a node belongs to. This 16 bit identifier is called the PAN ID, and is displayed in the tree view under “Networking” as seen in Figure 88, and is displayed in hexadecimal form. The value can be changed by clicking on the PAN ID in the tree view, this displays a text box next to the PAN ID to set it, as shown in Figure 88. The value “1234” is used in this example, each end device must have the same PANID in order to connect to the end device.

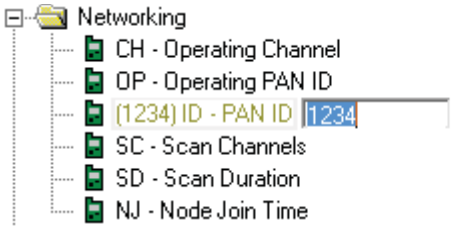

Figure 88 X-CTU - Setting network parameters for coordinator XBee module.

4. The OpenWRT router has a serial port that is configured by the XBee routing software with a baud rate of 115200 bits/second, and uses API mode with escaping. The coordinator needs to be setup in order to connect to the OpenWRT router with these settings, which are in the tree view under “Serial Settings”. The baud rate appears as “BD – Baud Rate” and is set using a drop down box and needs to be set to “7 – 115200” as seen in Figure 89. The API mode appears as “AP – API Enable”, which needs to be set to the value of “2”, as seen in Figure 89. This means API mode is enabled and character escaping is enabled. A value of “1” would mean that only API mode is enabled.

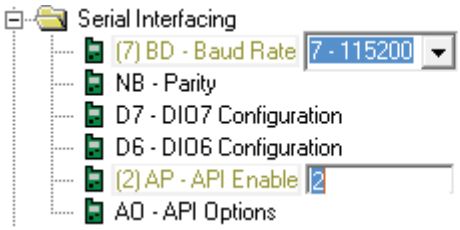

Figure 89 X-CTU - Setting serial connection parameters for coordinator XBee module.

5. The tree view now contains all the settings required to establish a ZigBee network and connect to the OpenWRT router. These settings need to be written to the XBee module, and depending on the firmware present, a firmware upgrade may be required. Clicking on the “Write” button will initiate the process to upgrade the firmware and write the settings to the XBee module. A progress bar will appear as in the top of Figure 90 to indicate the status of a firmware update. This will be followed by text output to indicate the status of writing the settings to the XBee module as seen in the bottom of Figure 90.

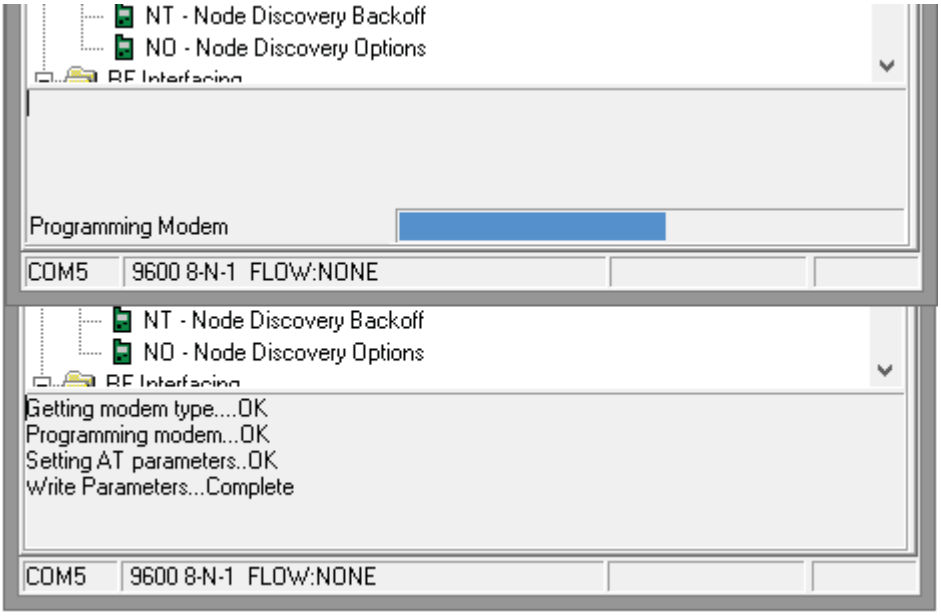

Figure 90 X-CTU - upper is programming coordinator XBee module and lower is configuring coordinator XBee module.

6. The serial settings for the XBee module can be verified by click on the "PC Settings" tab and testing the connection. The baud rate and API settings need to be changed before the connection can be tested. Set the baud rate to 115200 and under the API section check the boxes next to "Enable API" and "Use escape characters (ATAP = 2)" as shown in Figure 91. Click the "Test / Query" button to test the serial connection, this should display a dialog with the XBee module parameters if the connection was successful.

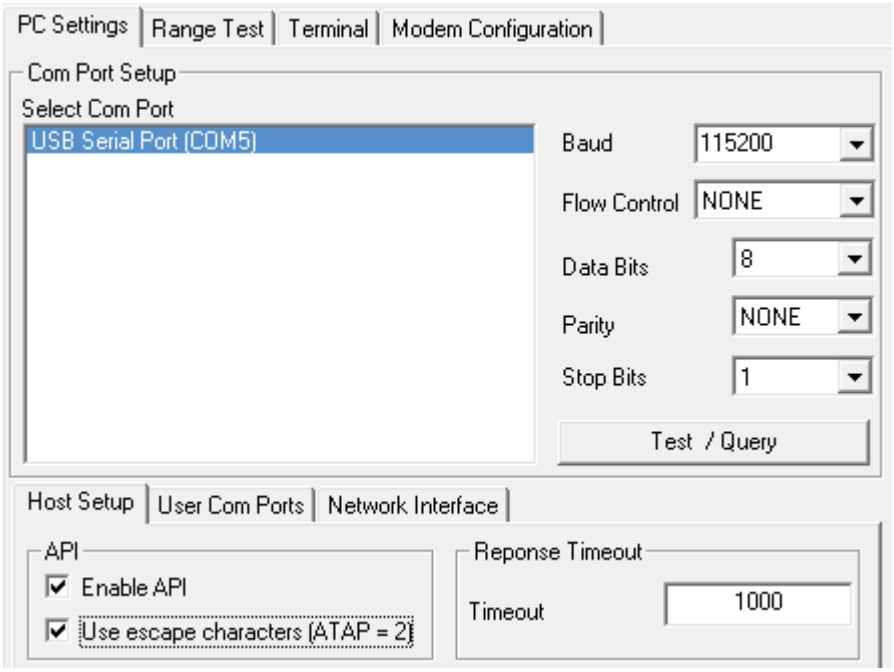

Figure 91 X-CTU - serial configuration for testing connection to coordinator XBee module for the OpenWRT router.

7. The last step is to verify the network settings were written correctly to the XBee module. Navigate back to the "Modem Configuration" tab, and click the "Read" button, which will

populate the tree view with the modules settings as shown in Figure 92. The PAN ID should be identical to the one set in the previous steps.

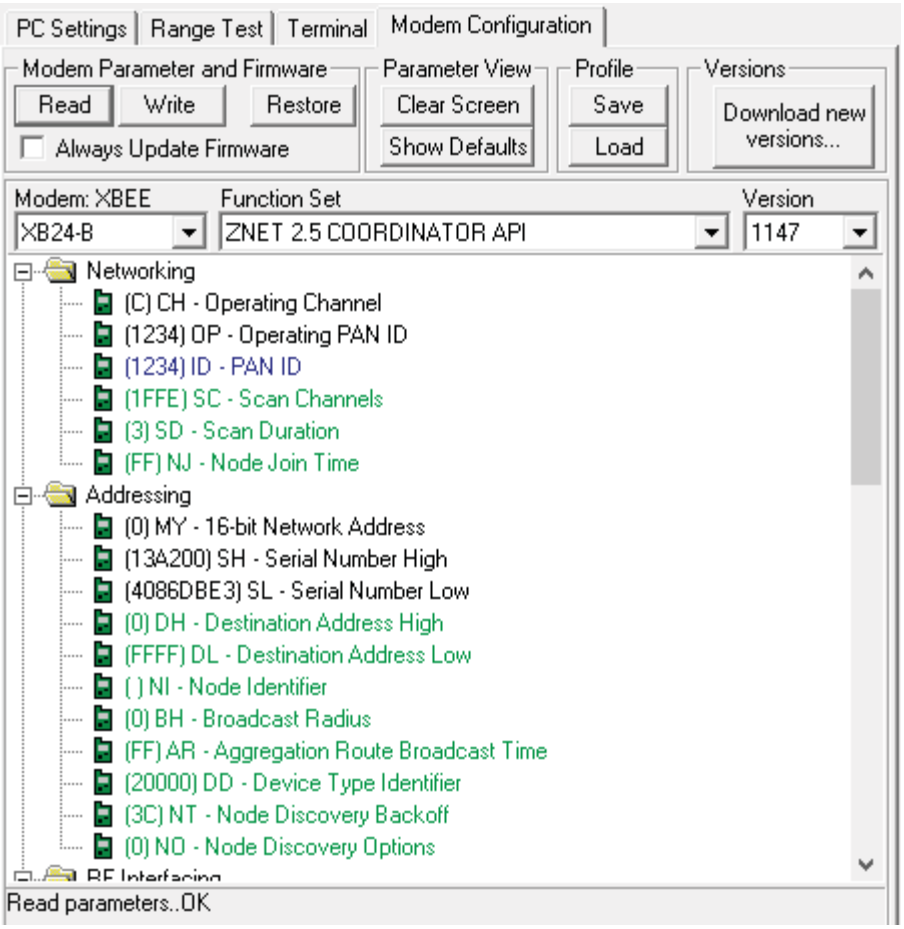

Figure 92 X-CTU – Verifying settings by reading them from coordinator XBee module.

5.2.4.3 XBee End Device Configuration

ZigBee end devices connect to a ZigBee coordinator using a unique identifier (PAN ID) in order to communicate with other end devices and the coordinator belonging to the ZigBee network with the unique identifier (PAN ID). The XBee end device needs to be configured with this unique identifier (PAN ID) so that it can send sensor data to the XBee coordinator and/or the coordinator can send commands to the end device. In order to send sensor data the end device needs to be configured to sample the data by selecting the channel and sampling rate. The XBee module can also go into a low powered mode between sampling in order to save power, this also needs to be configured. The following steps give details on configuring the XBee end device to exist in a network created by a coordinator, sample analogue data, and sleep between sampling analogue data.

1. Setup the serial connection in the “PC Settings” by selecting the com port and baud rate. Test the serial connection using the “Test/Query” button to ensure that the XBee module is connected.

2. The end device software needs to be selected for the XBee module. Click on the “Modem Configuration” tab, select “XB24-B” from the drop box under “Modem: …”, and select “ZNET 2.5 ROUTER/END DEVICE API” from the drop box under “Function Set” as shown in Figure 93. This will populate the tree view with settings for an XBee end device.

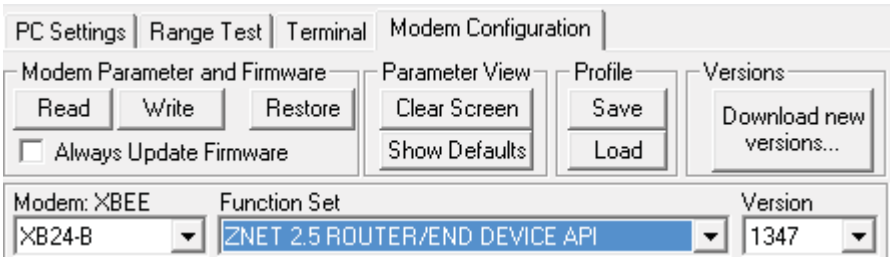

Figure 93 X-CTU Setting module type and firmware for End Device.

3. The 16 bit unique identifier (PANID) determines which network the node belongs to. Set the value to 1234, as shown in Figure 94, this is the same as the PAN ID set on the coordinator.

Figure 94 X-CTU Setting PAN ID for End Device.

4. If the node is required to sleep to conserve power the following settings in the “Sleep Modes” can be set:
    a. “Sleep Mode” enables and selects the type of sleep mode. Set this value to “4 – CYCLIC SLEEP” in the drop down box, as shown in Figure 95. Cyclic sleep mode means that the node will sleep for a given time (“Cyclic Sleep Period”), stay awake for a given time period (“Time before sleep”), and power down the input/out for a given number of cycles (“Number of cycles to power down IO”).
    b. “Time before Sleep” is the amount of time the XBee module remains awake (powered on) before it powers down and goes to sleep. This value is in hexadecimal form and is in the unit milliseconds. Set the value to 32, as shown in Figure 95, this means the XBee module will remain awake for 50 milliseconds before sleeping. During this time the radio will be active and the XBee module will request any queued messages from the coordinator.
    c. “Cyclic Sleep Period” is the amount of time the XBee Module remains powered down or in the sleep state. The value is in hexadecimal for in is the unit 10s of milliseconds. Set the value to 7D0, as shown in Figure 95, this means the module will sleep for 20

seconds. During this time the radio and microcontroller of the XBee module will be powered down, and hence not able to receive data from the coordinator or serial.

d. "Number of Cycles to power down IO" is the number of cycles the XBee module will not power the input/output when in the power on state. This allows an XBee module to power up and check for messages from the coordinator without sending sample packets. Set the value to 3, as shown in Figure 95, this means that the XBee module will only send a sample packet after every 3rd time the XBee module wakes up.

Further information on the sleep cycle of XBee modules can be found in the XBee Product Manual [89].

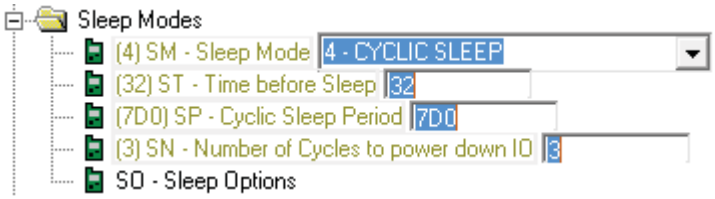

Figure 95 X-CTU Setting Sleep Mode settings for End Device.

5. An XBee module has a number of digital inputs/outputs, and ADCs (analogue to digital converters). These are multiplexed to various pins which need to be configured based on where each sensor or actuator is connected. The configuration shown in Figure 96 shows the first 3 pins are configured as ADC inputs. The rate at which these pins are sampled ("IO Sampling rate") is given in terms of milliseconds in a hexadecimal format. The value of the sampling rate depends on the sleep mode used. If no sleep mode is used the module will continuously send samples with the period between samples being the value of the "IO Sampling rate". If a sleep mode is used the module will send samples at the given rate until the module sleeps. In Figure 96 the sampling rate is configured so that it is greater than the time the module will be awake for (specified in the previous step), therefore only one sample will be sent each time the module wakes up.

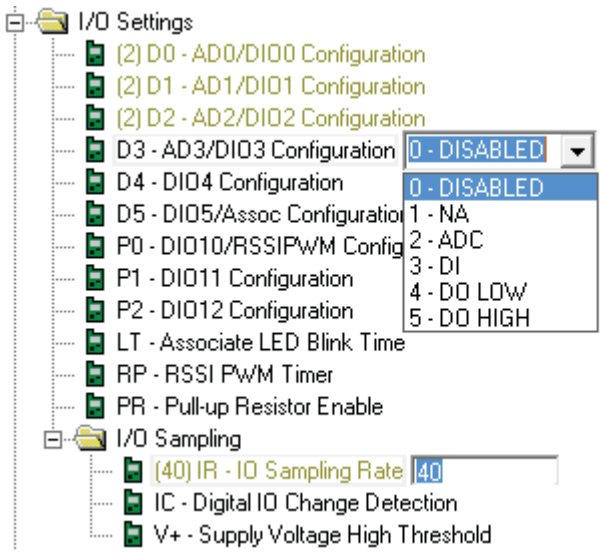

Figure 96 X-CTU Setting Input/Output settings for End Device.

6. The tree view now contains all the settings required for the End Device, clicking the “Write” button will write the settings to the XBee module, and if necessary update the firmware. After the process completes the settings can be confirmed by clicking the “Read” button to populate the tree view with the current settings of the module. These current settings should reflect the changes made. If a sleep mode has been set the module may fail to respond when reading the settings, this requires the module to be reset. The module can be reset by shorting the reset pin to ground with a wire for a second.

5.3 Network Interface Implementation Techniques

Techniques for receiving, parsing, sending, and creating network packets are given in this section. The first section 5.3.1 shows how to send and receive raw packets from a network interface in Linux. The second section 5.3.2 demonstrates a technique to parse and create network packets.

5.3.1 Tunnel Driver Interface

Network connectivity (and hence internet connectivity) is provided by an embedded Linux operating system and networking hardware. The connection to the internet is provided by Ethernet or WiFi in most cases, these hardware interfaces have a driver that provides a network interface. In order for a WSN to interconnect with the internet it must have a network interface present in the embedded Linux operating system. Routing between the WSN network interface and internet network interface is what gives the WSN internet connectivity. The network interface can be provided by drivers developed for the embedded Linux operating system, however driver development is a difficult time consuming task. The alternative to developing a driver is to use user space software to interface with the hardware and provide a network interface. In this case the network interface still requires a driver, which is provided by a virtual network interface known as the tun/tap driver in Linux.

Normally a Linux kernel network driver is attached to a hardware interface such as an Ethernet card or Wi-Fi adapter. However the tun/tap driver is a virtual network device driver that can attach to software running in the user space providing a raw network interface. When software interacts with the Linux kernel networking functionality it is commonly done via an abstracted interface called sockets where the raw data sent via the network interfaces cannot be accessed or modified. The tun/tap driver provides direct access to the raw data sent and received by the interface. There are two levels at which the network interface can be used, the tun (tunnel) level or the tap level. The tun operates at the network layer, and the tap operates at the link layer. For the WSN network only the network layer is required so the tun driver was used.

The tun driver provides a method to directly send and receive IPv6 UDP packets for a virtual subnet. The wireless sensor network can be mapped to the subnet defined by the tun interface allowing the nodes in the wireless sensor network to receive an IPv6 address.

5.3.2 Parsing raw data from buffers

The data formats used in network packets require parsing in order to interpret the data in code. The following sections give details on the XStruct technique to extract data from network packets and to create network packets.

5.3.2.1 Host and network byte order

In most networking standards the order of the bytes contained in packets is big endian, this was instigated in the first definitions for networking standards such as [90], and is known as network order. However most CPU architectures, such as x86, use a little endian format (or host byte order) when storing data in memory. This presents a problem when parsing or creating network packets as the data is stored in a different order.

The order of bytes in a data type is also known as the endianness, which can be either little endian or big endian. Big endian means that the byte order is from least significant byte to most significant byte as the memory addresses increase. Little endian is the opposite with the most significant byte appearing first and the least significant byte appearing last as the memory addresses increase. The hexadecimal value 0x12345678 is shown in Table 6 with different endianness in memory.

Table 6 Memory storage locations of the value 0x12345678 with different endianness

Memory address	1	2	3	4
Little Endian	0x78	0x56	0x34	0x12
Big Endian	0x12	0x34	0x56	0x78

From the table it can be seen that in order to produce a network order representation of the host order data (or vice-versa) a conversion technique is needed if the CPU architecture is little endian.

Table 7 shows the structure of an IPv6 header in a packet in network byte order and Table 8 shows the structure of an IPv6 header in memory. It can clearly be seen that the host byte order structure and network byte order structure appear differently, which means conversion of the structure representing the IPv6 header is required.

Table 7 IPv6 header in network byte order

Bytes	1		2		3	4
0	Version		Traffic Class		Flow Label	
	0x6	1	0x2	3	0x45	0x67
4	Payload Length				Next Header	Hop Limit
	0x00		0x20		0x11	0x40
8	Source Address					
	0x20		0x04		0x00	0x00
12	Source Address					
	0x00		0x00		0x00	0x00
16	Source Address					
	0x00		0x00		0x00	0x00
20	Source Address					
	0x00		0x00		0x00	0x01
24	Destination Address					
	0x20		0x04		0x00	0x00
28	Destination Address					
	0x00		0x00		0x12	0x34
32	Destination Address					
	0x00		0x13		0xa2	0x00
36	Destination Address					
	0x26		0xe4		0x7e	0xf4

The colours of each section of the IPv6 header shown in Table 7 correlate with the colours in Table 8. It can be seen that the Version parameter takes up the entire byte in host byte order (Table 8), where as in the network byte order (Table 7) it takes up 4 bits and the lower 4 bits are allocated to the Traffic Class. The Traffic Class is located at the next byte boundary in the host byte order (Table 8), where as in the network byte order (Table 7) it is between the byte boundaries.

Table 8 IPv6 header structure in memory, or host byte order

Bytes	1	2	3	4
0	Version	Traffic Class	Padding	
	0x06	0x12	0x00	0x00
4	Flow Label			
	0x67	0x45	0x03	0x00
8	Payload Length		Next Header	Hop Limit
	0x20	0x00	0x11	0x40
12	Source Address			
	0x20	0x04	0x00	0x00
16	Source Address			
	0x00	0x00	0x00	0x00
20	Source Address			
	0x00	0x00	0x00	0x00
24	Source Address			
	0x00	0x00	0x00	0x01
28	Destination Address			
	0x20	0x04	0x00	0x00
32	Destination Address			
	0x00	0x00	0x12	0x34
36	Destination Address			
	0x00	0x13	0xa2	0x00
40	Destination Address			
	0x26	0xe4	0x7e	0xf4

The following sections introduce the techniques required for converting between network byte order and host byte order, in order to parse and create packets.

5.3.2.2 XMacro technique

XMacros are a technique that uses macros to create initialisers for variables at compile time [91]. This means that when code is compiled a macro defines the variable and type instead of implicitly defining. The following code defines a variable called “an_integer” with the type “int”.

```
1   int an_integer;
```

Using the XMacro technique, the same variable can be defined using the following code.

```
1   #define X(type, name) type      name;
2   X(int,an_integer)
```

The “#define” statement (line 1) is used to define the macro “X” which takes the parameters type and name. When these parameters are passed to the at compile time macro they simply get placed in the order that they appear after the macro definition which is the type, a space, and the name. This means the result at compile time is a variable definition specified by the parameters given to the macro, therefore the code above performs a variable definition identical to the single line of code used to define the variable “an_integer”.

Macros can be undefined, which allows the “X” macro defined above to be repurposed to perform additional functions. This means a variable can be defined using a definition macro, and then operated upon using a functional macro. The following macro code gives an example of repurposing the “X” macro to perform definition, initialisation, and display of variables defined in the macro “VARIABLE_LIST”

```
1   #define VARIABLE_LIST \
2               X(int,integer_1,1)\
3               X(int,integer_2,2)\
4               X(int,integer_3,3)
5   #define X(type, name, value) type name;
6   VARIABLE_LIST
7   #undef X
8   #define X(type, name, value) name = value;
9   VARIABLE_LIST
10  #undef X
11  #define X(type, name, value) printf("name:%s type:%s value:%i",#name,#type,name);
12  VARIABLE_LIST
```

At compile time the following code will be produced from the above macro code.

```
1   int  integer_1;
2   int  integer_2;
3   int  integer_3;
4   integer_1 = 1;
5   integer_2 = 2;
6   integer_3 = 3;
7   printf("name:%s type:%s value:%i","integer_1","int",integer_1);
8   printf("name:%s type:%s value:%i","integer_2","int",integer_2);
9   printf("name:%s type:%s value:%i","integer_3","int",integer_3);
```

Lines 1 to 3 are generated by the first “X” Macro defined on line 5 in the macro code, which define the variables. Lines 4 to 6 are generated by the second “X” Macro defined at line 8 in the macro code section, these lines set the variables. The last three lines (7 to 9) then print out the variables and are generated by the macro defined on line 11 in the macro code.

This means that a single macro (for example “VARIABLE_LIST”) can be repurposed in order to define, initialise and perform and function on a variable.

5.3.2.3 XStruct technique

The XStruct technique uses the XMacro technique with structures for conversion of data between host and network byte order. This involves using an XMacro to define the structures then redefining the macro to call appropriate conversion functions. The advantage of this technique is that code for conversion of each structure type is not required, reducing the amount of effort required in writing code to parse and create network packets. The disadvantage of this technique is that it will make the code generated inefficient and difficult to understand and debug.

Each data type used in a structure requires a conversion function that converts from and to network byte order. In order to do this an XMacro is used to define the variable of a given type, and then repurposed to define a function call to a conversion function for the given type with reference to the variable in the structure. The conversion function operates on memory addresses of the structure and a buffer which are specified as parameters. The structure memory address will contain the host byte order and the buffer memory address will contain the network byte order. To simplify the XMacros only a single conversion function is used, therefore the type of conversion (network to host, or host to network) is also specified as a parameter.

The implementation of the XStruct technique works by defining a structure using macros in headers. In the source code files the macros are repurposed to call conversion functions defined in the source code. This allows for the definition and conversion of a structure without writing specific conversion code for that structure.

An example of XStruct structure definition is shown in the code example below. The structure defined is the XBee API header which has 3 members, the start delimiter (“SD”), the Length, and the command ID (“cmdID”).

```
1   #define XSTRUCT_DEF
2   #include "xstruct.h"
3
4   #define _XBeeAPIHeader\
5       X(Uint8, SD)\
6       X(Uint16,         Length)\
7       X(Uint8,          cmdID)
8
9   typedef struct
10  {
11      _XBeeAPIHeader
12  }XBeeAPIHeader;
```

The first part is the definition of “XSTRUCT_DEF” on line 1 and the inclusion of “xstruct.h” on line 2. The “xstruct.h” header creates the XMacro definitions for defining variables in structures when “XSTRUCT_DEF” has been defined.

The second part is the definition of the macro to create the variables in the structure which is on lines 4 to 7. These lines create a macro that uses the XMacro to define the variable name and type. Multi-

line macros are made possible by escaping the return character at the end of each line with the backslash character.

The third part is the definition of the structure type which is on lines 9 to 12. Line 11 uses the macro in the second part to populate the members of the structure.

The result at compile time of the code above is the following structure definition in the header.

```
1   typedef struct
2   {
3       Uint8 SD;
4       Uint16 Length;
5       Uint8 cmdID;
6   }XBeeAPIHeader;
```

To perform a conversion of the structure, conversion functions of the types “Uint8” and “Uint16” must be defined in the code. The following code gives an example of the conversion function for the “Uint16” type.

```
1   void memop_Uint16(Uint16 * structP,
2                     Uint8 * bufP,
3                     Uint16 * bufPos,
4                     Uint16 bufSz,
5                     Uint8 op)
6   {
7       switch(op)
8       {
9       case NtoH:
10              *structP = ((Uint16)bufP[*bufPos] << 8) + bufP[*bufPos + 1];
11              break;
12      case HtoN:
13              bufP[*bufPos] = (Uint8)(*structP >> 8);
14              bufP[*bufPos + 1] = (Uint8)(*structP);
15              break;
16      }
17      *bufPos+=2;
18  }
```

The function is defined using the following convention.

- The function name must start with “memop_” with the type name after (the type name is case sensitive) this is shown on line 1.
- The first parameter must be a pointer of the type the conversion function is converting. This pointer locates the variable in the structure in memory, which is in host byte order. Line 1 also contains the definition for this parameter.
- The second parameter must be a pointer to buffer (shown one line 2), which is used to contain network order data.
- The third parameter must be a pointer to an unsigned integer (shown on line 3) which contains the current offset in the buffer. This is used to keep track of the current location in the buffer and must be incremented the size of the type in the conversion function. A pointer is required as the value is changed in the function.

- The fourth parameter must be an unsigned integer (shown on line 4) that contains the length of the buffer. This prevents buffer overruns from occurring (the third parameter, offset, must not be greater than the fourth, length).
- The last parameter (shown on line 5) is used to determine which conversion is taking place (network to host or host to network).

The function contains a switch statement on line 7 which is used to determine which conversion operation is required. Line 10 performs the network to host conversion by copying data from the buffer at the required offset to the structures member given by the pointer. Line 13 and 14 performs the host to network conversion by copying the individual bytes from the structures member to the buffer. Line 17 moves the offset in the buffer to the position of the next data item, this must occur for both operation types therefore it is not in the switch statement.

Conversion functions in “xstruct.h” called ntoh (network to host) and hton (host to network) then use the XMacros to convert the structure defined using the XStruct method. The following lines of code demonstrate the conversion from a network order buffer to a host order structure. This code is located in the source file with the “.c” extension.

```
1  #undef XSTRUCT_DEF
2  #include "xstruct.h"
3  ...
4  Uint8 HeaderBuf[] = {0x7e, 0x00, 0x18, 0x92};
5  XBeeAPIHeader APIheader = {0};
6  ntoh(XBeeAPIHeader,APIheader, &HeaderBuf[0],sizeof(HeaderBuf));
```

The first two lines of code are used to repurpose the XMacros to call the definition functions. The “xstruct.h” header defines the XMacros as function calls to the conversion functions if no definition of “XSTRUCT_DEF” is found, line 1 ensures that it is undefined. These lines appear at the beginning of the source file where the headers are included.

Line 4 is a declaration and initialisation of a variable that contains the network order data of an XBee header. The first byte is the frame delimiter which is the hexadecimal value 0x73. The next two bytes are the frame length which is the value 0x0018 (24 in decimal). The last byte, 0x92, is the command id.

Lines 5 and 6 will appear in function, line 4 declares a variable with the XStruct type “XBeeAPIHeaer”, and line 5 is a macro to perform the network to host conversion from the buffer. The parameters for the “ntoh” function are the name of the type of structure, the variable name of the structure, a pointer to the buffer, and the length of the buffer. The “ntoh” macro in the previous code section will become the follow code at compile time.

The following code sections show the stages of the macros produced by the “ntoh” macro. The first stage is the expansion of the “ntoh” macro.

```
1   ntoh(XBeeAPIHeader,APIheader, &HeaderBuf[0],sizeof(HeaderBuf));
```

Expands to:

```
1   Uint8 op = NtoH;
2   XBeeAPIHeader * s = &APIheader;
3   Uint8 * pbuf = (Uint8 *)&HeaderBuf[0];
4   Uint16 bufPos = 0;
5   Uint16 bufSz = sizeof(HeaderBuf);
6   memset(&APIheader,0,sizeof(XBeeAPIHeader));
7   _XBeeAPIHeader
```

Line 2 to 7 contain variables required for the conversion function. Details of these are given later.

Line 9 contains the macro for defining the members of the XBeeAPIHeader. Expanding the macro gives the following code.

```
1   Uint8 op = NtoH;
2   XBeeAPIHeader * s = &APIheader;
3   Uint8 * pbuf = (Uint8 *)&HeaderBuf[0];
4   Uint16 bufPos = 0;
5   Uint16 bufSz = sizeof(HeaderBuf);
6   memset(&APIheader,0,sizeof(XBeeAPIHeader));
7   X(Uint8, SD)
8   X(Uint16, Length)
9   X(Uint8, cmdID)
```

This gives multiple X macros for each member of the structure which are on lines 8 to 10. The original purpose of the X macro was to provide a declaration of these members, but when repurposed the macro then creates a function call to the conversion function. The definition for repurposing the “X” macro function to call a conversion function is as follows.

```
1   #undef X
2   #define X(type, name)\
3       if(bufPos < bufSz)\
4       memop_ ## type(&s->name, pbuf, &bufPos, bufSz, op);\
```

Line 1 removes the previous definition of the macro “X”, which was to declare the variable.

Line 2 to 4 redefines the “X” macro to first check for a buffer overrun, and call the conversion function if there is no buffer overrun. The conversion functions naming convention is given by the definition of the “X” macro. It can be seen on line 4 that the function name is created by concatenating “memop_” with the macro variable “type” to create. The parameter convention is also given by the “X” macro. The parameters are generated using the macro variables to create specific function names

The final result of all the macros is the following code.

```
1   Uint8 op = NtoH;
2   XBeeAPIHeader * s = &APIheader;
3   Uint8 * pbuf = (Uint8 *)&HeaderBuf[0];
4   Uint16 bufPos = 0;
5   Uint16 bufSz = sizeof(HeaderBuf);
6   memset(&APIheader,0,sizeof(XBeeAPIHeader));
7   if(bufPos < bufSz)
8       memop_Uint8(&s->SD, pbuf, &bufPos, bufSz, op);
9   if(bufPos < bufSz)
10      memop_Uint16(&s->Length, pbuf, &bufPos, bufSz, op);
11  if(bufPos < bufSz)
```

```
12      memop_Uint8(&s->cmdID, pbuf, &bufPos, bufSz, op);
```

Lines 2 to 7 declare and set the variables required by the conversion functions.

Line 2 declares and set the operation variable used to tell the conversion function which operation to perform.

Line 3 creates a pointer variable that contains a pointer to the memory address of the structure variable given to the “ntoh” macro. This uses the first and second parameters given to the “ntoh” macro.

Line 4 creates a pointer variable that contains a pointer to the memory address of the buffer. This uses the third parameter given to the “ntoh” macro.

Line 5 creates a variable to store the current offset in the buffer.

Line 6 creates a variable that contains the size of the buffer.

Line 7 sets the all the memory of the structure to a value of 0.

Lines 8 to 13 are generated form the XMacros defined in the structures definition macro. These are the repurposed macros to call the conversion functions. Before calling the conversion function a simple check to make sure there isn’t a buffer overrun is performed.

The Xstruct technique demonstrated provides a technique to convert between data formats used on the host CPU architecture and network packets. This technique makes the conversion between structures and buffers possible with a single line of code, making writing code quicker and easier to read. However if the macro malfunctions debugging the cause of the malfunction is difficult as the macros are expanded at compile time, meaning the expanded code is not visible.

5.4 ZigBee IoT Platform Implementation

The first IoT platform developed consisted of a ZigBee based WSN connected to a gateway to provide access to the WSN from an internet server. The following sections provide the implementation details of connecting the ZigBee WSN to the gateway, and software running on the gateway to allow the server to communicate with the WSN.

5.4.1 ZigBee Hardware Implementation

XBee modules are used to create a ZigBee network, and the WRT54GL router to create a gateway to the ZigBee network. A Raspberry PI was also used to create a gateway, however it was not used as extensively as the WRT54GL as it is a new product. The following sections give details of the hardware interface, and sensor modules. The hardware interface is the hardware created to connect the XBee

module to the WRT54GL, and the sensor modules are various hardware interfaces for connecting sensors to the XBee modules.

5.4.1.1 XBee Gateway

Two hardware platforms were considered for the IoT gateway, the WRT54GL router and the Raspberry PI. The WRT54GL was primarily used as it was available before the Raspberry PI and has a more robust framework. A hardware interface was required for both platforms in order to connect the XBee coordinator module via serial to the platform.

The primary hardware used to create an IoT gateway was a Linksys WRT54GL router running an embedded version of Linux called OpenWRT [92]. This router is based upon the WRT54G router which became popular as it ran embedded Linux which could be modified. The WRT54GL was released specifically for modifying the firmware. There are several versions of Linux that were developed for the hardware such as DD-WRT [93], Tomato [94], and OpenWRT. DD-WRT does provide source code and development tools, but OpenWRT provides a more complete and comprehensive toolset for configuration and compilation of the Linux operating system which is why it was chosen.

The WRT54GL provides two serial ports [74], one of which is available to connect to the XBee coordinator. The serial pins provided are shown in Figure 97. Serial port 2 was used to connect to the XBee coordinator because serial port 1 is used by OpenWRT to provide a terminal to access the Linux shell from hardware. A power pin is also provided that can supply the 3.3V required by the XBee coordinator module.

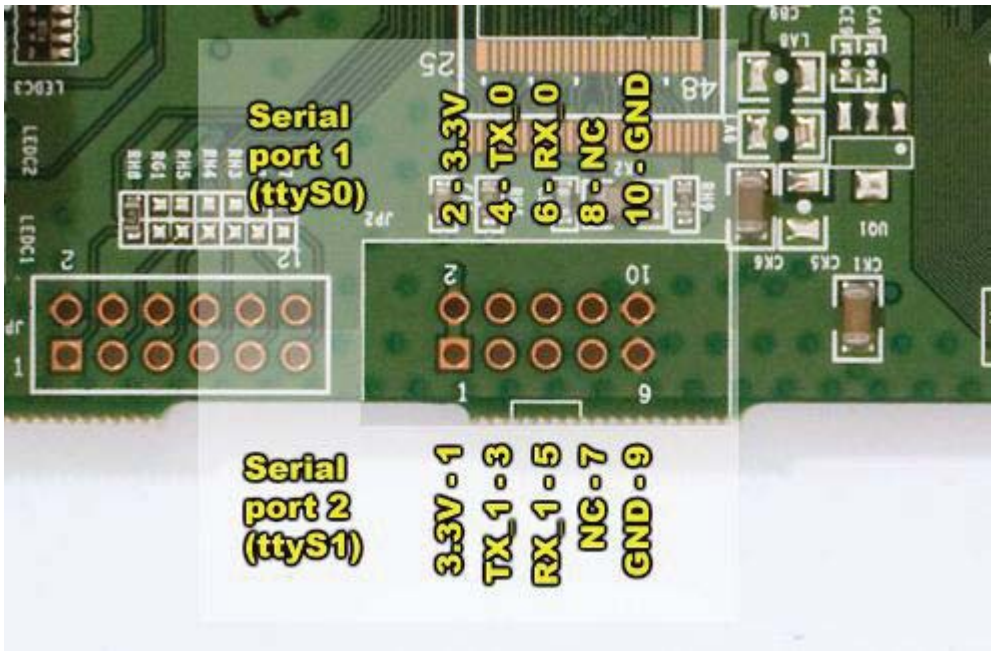

Figure 97 WRT54GL - serial port pins provided on the board obtained from [74]

A PCB was designed to attach the XBee coordinator module to the WRT54GL board. Figure 98 shows the PCB attached to the WRT54GL board.

Figure 98 WRT54GL router with XBee Coordinator attached to serial port using fabricated PCB

Figure 99 shows the schematic and PCB design for the PCB shown in Figure 98. This design conforms to the antenna keep-out requirements in [89]. The requirement is that no PCB traces should be in a zone around the antenna, which is located at the top of the XBee module. In the schematic the part named WRT is the header to connect to the header on WRT54GL board shown in Figure 97. The pin numbers do not match the pin numbers on the board because only a single row header was required.

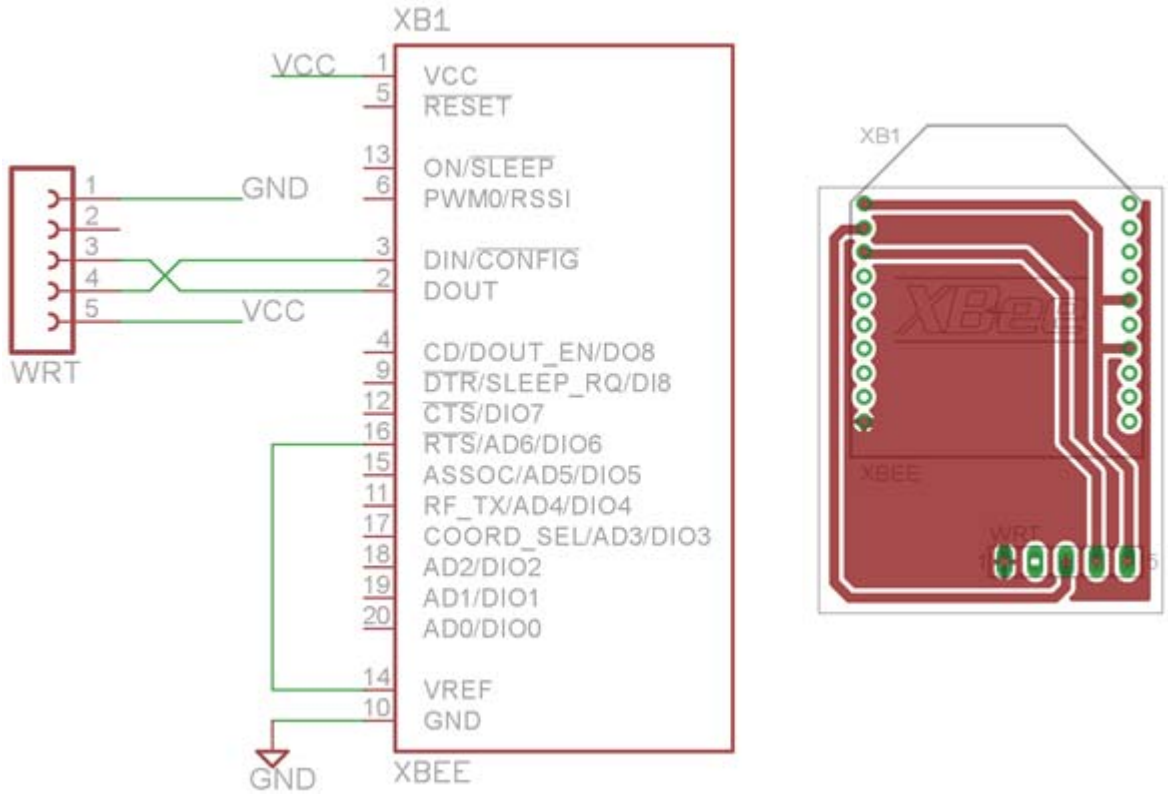

Figure 99 Schematic and PCB design for hardware interface between XBee S2 coordinator module and WRT54GL board

In addition to the WRT54GL hardware, the Raspberry PI was also used. A PCB was fabricated to interface with the Raspberry PI and is shown in Figure 100.

Figure 100 Raspberry PI with fabricated PCB for XBee coordinator to connect to serial port

The raspberry pi provides 3.3V and 5V on the header shown in Figure 101. The 5V pin is on the same row and is in close proximity to the ground, transmit and receive pins. However the 3.3V pin is on a separate row, meaning a separate header would be required to connect to the 3.3V pin. Due to the

requirement for an additional header, and the current required from the 3.3V line the 5V pin was used instead. Using the 5V pin requires a voltage regulator to provide 3.3V to the XBee coordinator module. The 5V pin is directly connected to the USB header where power is supplied to the Raspberry PI, therefore current consumption on this pin depends on the power supplied to the Raspberry PI and not regulators on the Raspberry PI.

Figure 101 Raspberry PI header pins from [95]

The PCB shown in Figure 100 to connect the Raspberry PI and XBee module has the schematic and PCB design in Figure 102. The component Q1 is a voltage regulator (LE33CZ) which takes 5 volts from the 5V pin and outputs 3.3 volts for the XBee coordinator. The capacitors C1 and C2 are recommended in the datasheet for the LE33CZ [96]. SV1 is the header that connects to the Raspberry PI header providing power and communication. The first pin is the 5V pin on the Raspberry PI, the third and fourth pins are for the serial transmission lines and the fifth pin is the ground. The resistor R0 is used a jumper over the ground plane and may be a length of wire.

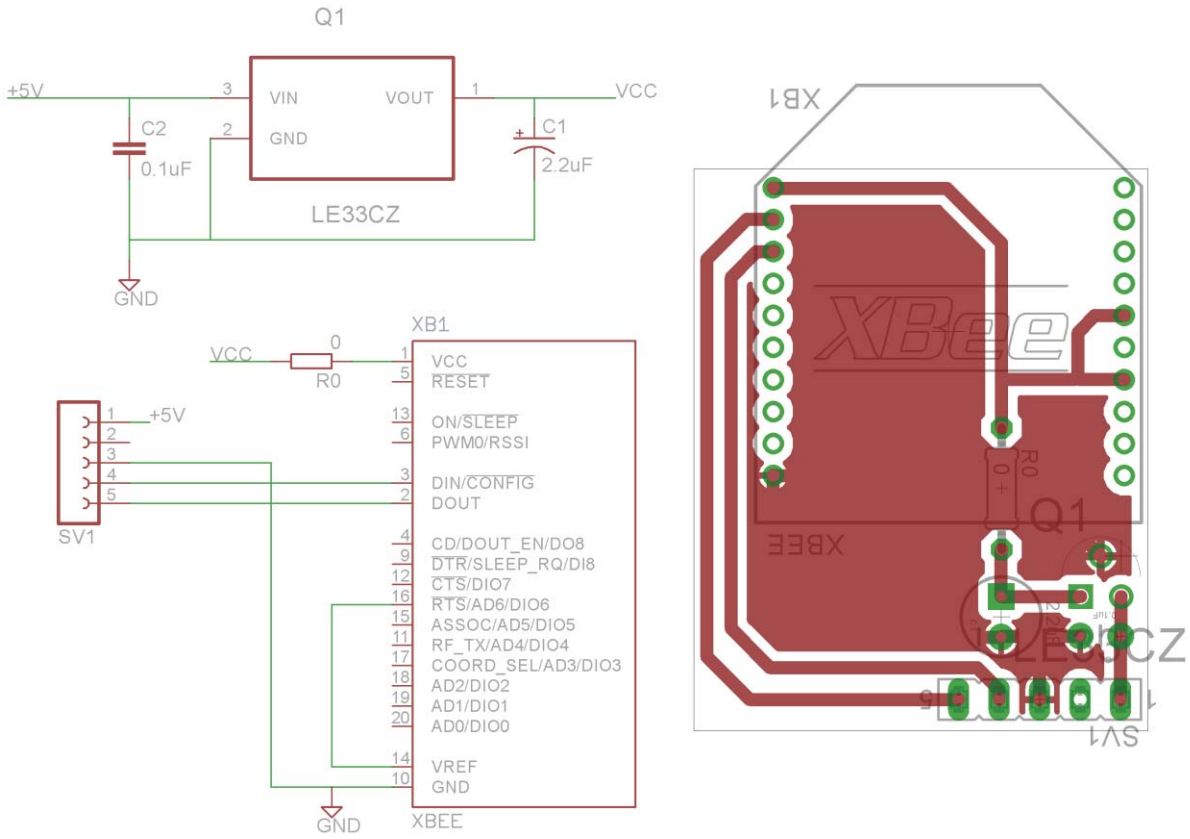

Figure 102 Schematic and PCB design for hardware interface between XBee S2 coordinator module and Raspberry PI

5.4.1.2 XBee Temperature Sensor Module

The XBee temperature sensor module uses the TMP3x temperature sensor, which is connected to the XBee modules' ADC pin. The TMP3x is a low power temperature sensor that draws 50 µA [97]. Table 9 shows the parameters of the three variants of the temperature sensors available. All the variants use the same physical package and can therefore be interchanged easily.

Table 9 TMP3x parameters from [97]

Temperature Sensor	TMP35	TMP36	TMP37
Output Voltage Range	100 mV to 2000 mV	100 mV to 2000 mV	100 mV to 2000 mV
Output Voltage Scale (m_{tmp3x})	10 mV/°C	10 mV/°C	20 mV/°C
Output Voltage Offset (c_{tmp3x})	0 mV	500 mV	0 mV
Input Temperature Range	10°C to 200°C	-40°C to 150°C	5°C to 100°C
Accuracy over Range	±2°C	±2°C	±2°C

The input range for the XBee Module is from 0V to 1.2V which is represented by a 10 bit integer. The following formulae show how the temperature (t_{tmp3x}) was derived for any of the TMP3x sensors, given the output voltage scale (m_{tmp3x}), the output voltage offset (c_{tmp3x}), output voltage (V_{tmp3x}), and the ADC value (ADC).

$$ADC = \frac{1024 \cdot V_{tmp3x}}{1.2} \Rightarrow V_{tmp3x} = \frac{1.2 \cdot ADC}{1024}$$

$$V_{tmp3x} = m_{tmp3x} \cdot t_{tmp3x} + c_{tmp3x} \Rightarrow t_{tmp3x} = \frac{V_{tmp3x} - c_{tmp3x}}{m_{tmp3x}}$$

$$\Rightarrow t_{tmp3x} = \left(\frac{1.2 \cdot ADC}{1024 \cdot m_{tmp3x}}\right) - \left(\frac{c_{tmp3x}}{m_{tmp3x}}\right)$$

Table 10 was created using the TMP3x characteristics in Table 9 and the formulae above for the ADC values produced from an XBee module. The minimum and maximum values for each sensor was determined based upon the minimum and maximum ADC value produced by the XBee module. The formulae for converting an XBee ADC value to a temperature for each sensor is also given.

Table 10 Temperature range for TMP3x sensor when using XBee module

	TMP35	**TMP36**	**TMP37**
Temperature (t_{tmp3x})	$\frac{1.2}{10.24} \cdot ADC$	$\frac{1.2}{10.24} \cdot ADC - 50$	$\frac{1.2}{20.48} \cdot ADC$
Minimum V_{tmp3x}	0.1 V	0.1 V	0.1 V
Minimum ADC	85	85	85
Minimum t_{tmp3x}	10°C	-40°C	5°C
Maximum V_{tmp3x}	1.2 V	1.2 V	1.2 V
Maximum ADC	1024	1024	1024
Maximum t_{tmp3x}	120°C	70°C	60°C
°C/ADC division	0.117°C	0.117°C	0.059°C

Figure 103 shows the schematic for connecting three TMP36 temperature sensors to the analogue inputs of an XBee Module. Any of the TMP3x sensor module could be used in the schematic as they have the same footprint. The input pins used are DIO0, DIO1, and DIO2 which need to be configured as analogue input pins. A 3.3V low-dropout voltage regulator (Q1 in the schematic) provides the power supply to the TMP36 temperature sensors and XBee module. The input range of the LE33CZ used is from 3.5V to 18V meaning it can be used with a large range of power supplies. An LED (LED1 in the schematic) is attached to the association indication pin of the XBee module in order to give visual status information. The resistor R0 is a current limiting resistor for the LED.

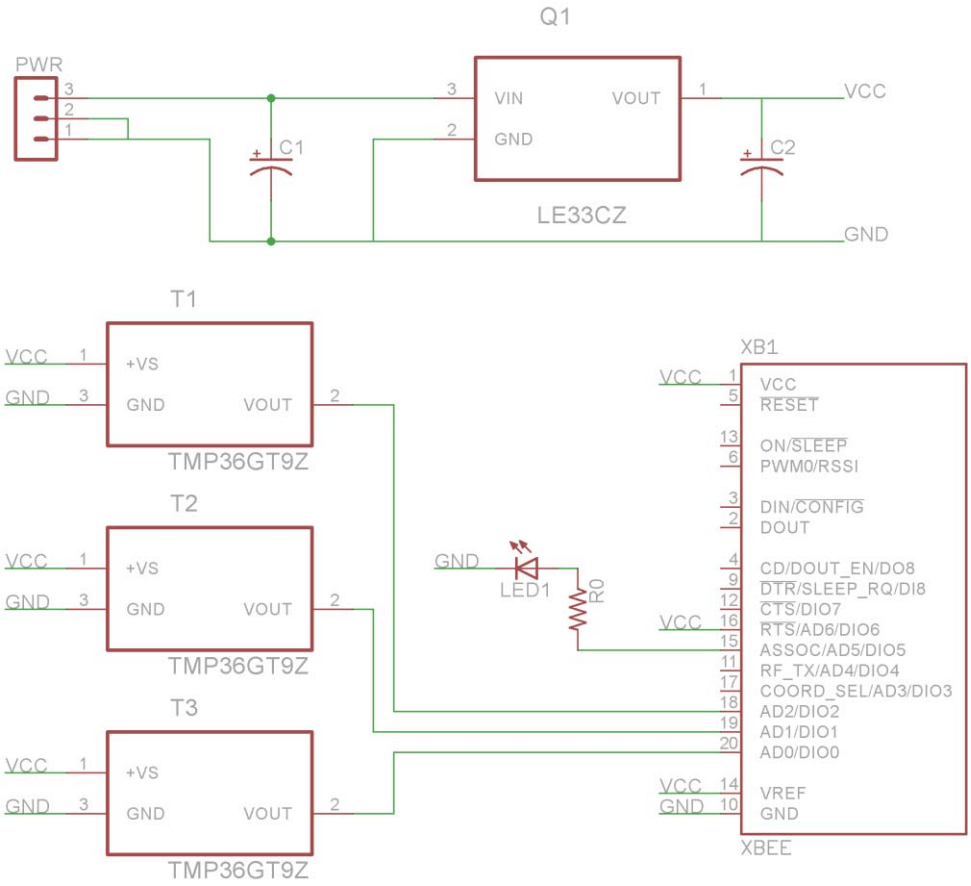

Figure 103 XBee temperature sensor hardware interface schematic

The PCB designed from the schematic in Figure 103 is shown in Figure 104. The PCB was designed to be as minimal as possible and the keep-out zone for the XBee modules antenna was also observed.

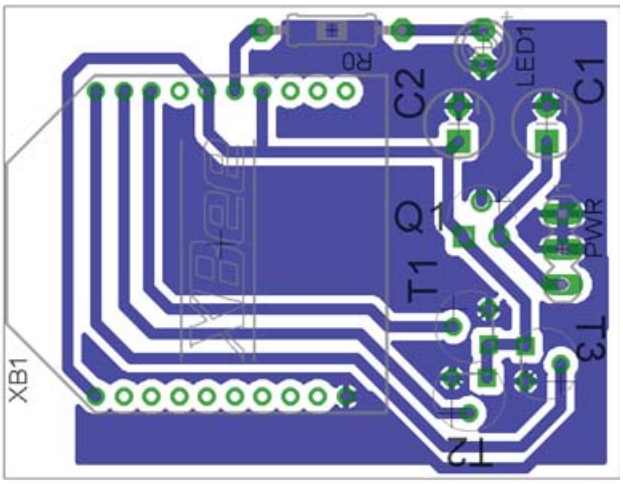

Figure 104 XBee temperature sensor hardware interface PCB

5.4.1.3 XBee Hot Water System Monitor

Hot water is supplied by a hot water cylinder in a typical household, and the hot water cylinder is heated either by electricity or gas. Solar heating systems can be used to heat the water during the day reducing the gas or power consumption provided there is sufficient solar energy. The system shown in

Figure 105 shows a solar water heater attached to an electric water cylinder. The solar heater system consists of the following.

- Solar heating panels, which heat the water using solar energy.
- A pump to circulate the water in the system. The water heated in the solar heater panels needs to be pumped back to the hot water cylinder, and replaced with cooler water to be heated.
- Temperature sensors in the Solar water heater, and hot water cylinder. These provide information to the controller to determine when to turn on the pump.
- The solar heat controller controls the pump based on the temperature sensors. When there is a predetermined temperature difference between the solar heat panel temperature and water cylinder temperature the pump circulates the hot water into the cylinder and cool water to the solar heater panels. The temperatures of each temperature probe are displayed on the solar heater controller.

The temperature sensors provided by the solar heating system are monitored by an XBee module.

The electric water heater uses a coil inside the cylinder to heat the water, this is controller using a thermostat that turns the coil on or off depending on the temperature. To monitor the current consumption of the water cylinders' coil a current transformer is used. The current transformer is monitored by the XBee module.

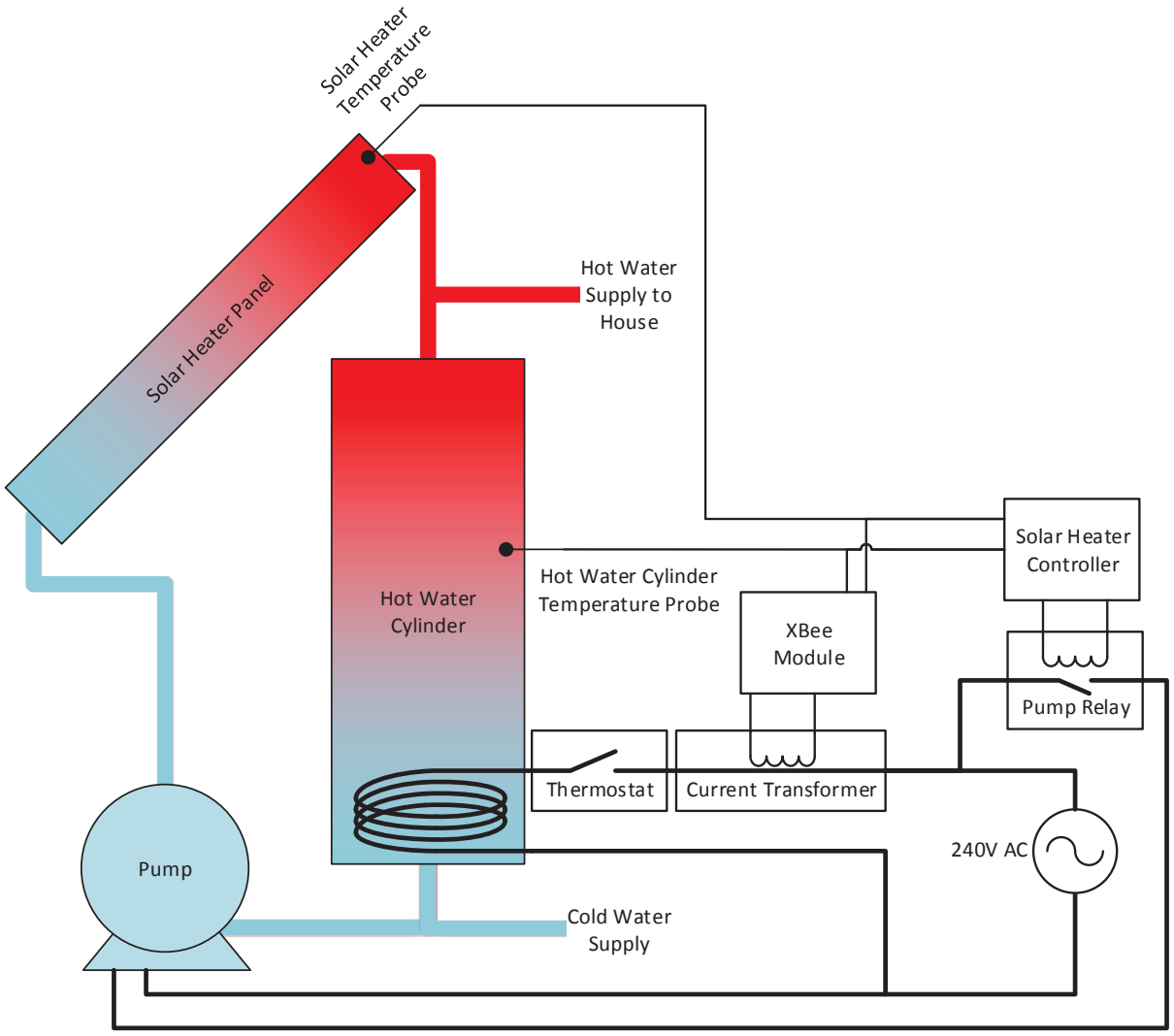

Figure 105 Hot water system and solar heating system

The inputs from the temperature sensor and current transformer required signal conditioning in order to be monitored by the XBee module. Measurements of the voltage output from the temperature probes were taken for various temperatures indicated on the solar heater controller in order to determine the signal conditioning required. The temperature probe signals must be scaled and offset to meet the input requirements for the analogue pins of the XBee module.

Figure 106 shows the input voltage versus the indicated temperature on the solar heater controller for the hot water cylinder. The trend line fitted to the data shows a good linear relationship between the output voltage and temperature. The formulae from the trend line in Figure 106 to give the input voltage (V_{inw}) of the probe for a given temperature (t) is as following.

$$V_{inw} = 0.02376 \cdot t_w + 0.11602$$

The range of temperatures that need to be measured is from 20 °C to 80 °C, substituting these values into the formulae above gives the following.

$$V_{inw} = 0.02376 \times 20\,°C + 0.11602 = 0.59\,V\ (2\ d.p.)$$

$$V_{inw} = 0.02376 \times 80\,°C + 0.11602 = 2.02\,V\ (2\ d.p.)$$

This gives the output range 0.59 V to 2.02 V which must be scaled to the input range 0 V to 1.2 V of the XBee Module. This means the signal needs to be attenuated by 83% and negatively offset by 0.59 V.

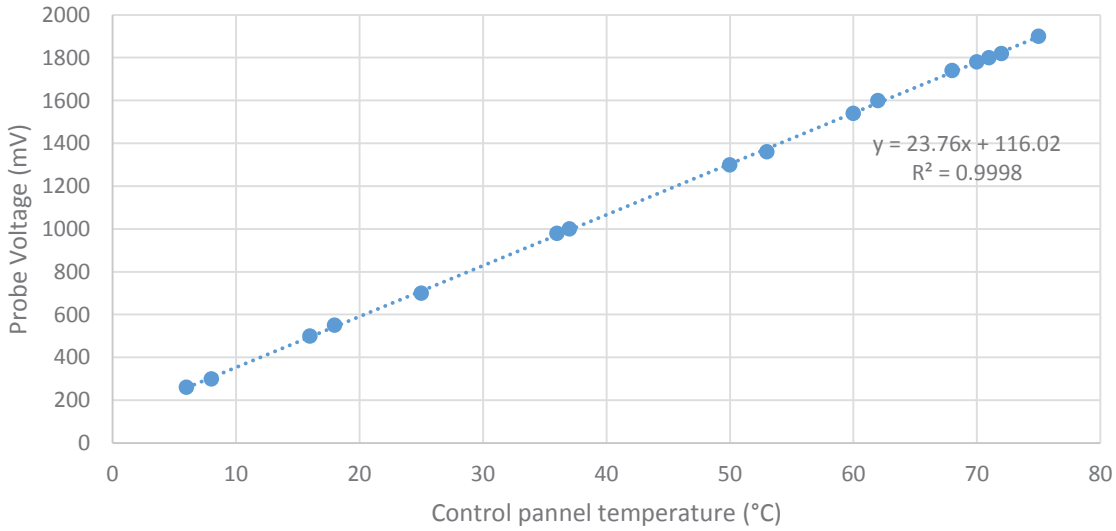

Figure 106 Graph of temperature Vs. Voltage for hot water cylinder probe

Figure 107 is a circuit designed to scale and offset the output voltage from the hot water cylinder temperature probe (V_{inw}) to the input range required by the XBee module (V_{outw}).

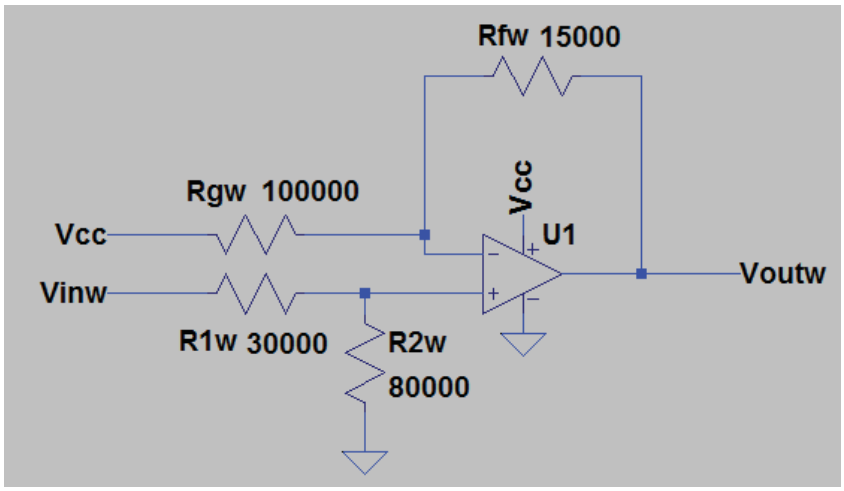

Figure 107 Schematic for circuit to condition signal from hot water cylinder probe

The values of the resistors were determined using the circuit in chapter 5.4 of [98], which gives the following formulae for the circuit in Figure 107.

$$V_{outw} = m \cdot V_{inw} - b$$

$$m = \left(\frac{R_{2W}}{R_{2W} + R_{1W}}\right) \cdot \left(1 + \frac{R_{fW}}{R_{gW}}\right)$$

$$b = V_{cc} \cdot \frac{R_{fW}}{R_{gW}}$$

Where m is the gain, b is offset, and the rest of the variables are shown in Figure 107. The following assumptions have been made; the gain m is 0.833, the offset b is 0.58, and the reference voltage V_{cc} is 3.3V. From these assumptions the values for R_{1W}, R_{2W}, R_{gW}, and R_{fW} can be determined as following.

$$b = V_{cc} \cdot \frac{R_{fW}}{R_{gW}} \Rightarrow R_{fW} = \frac{b \cdot R_{gW}}{V_{cc}}$$

$$m = \left(\frac{R_{2W}}{R_{2W} + R_{1W}}\right) \cdot \left(1 + \frac{R_{fW}}{R_{gW}}\right) = \left(\frac{R_{2W}}{R_{2W} + R_{1W}}\right) \cdot \left(1 + \frac{b \cdot R_{gW}}{R_{gW} \cdot V_{cc}}\right) = \left(\frac{R_{2W}}{R_{2W} + R_{1W}}\right) \cdot \left(1 + \frac{b}{V_{cc}}\right)$$

$$\Rightarrow \left(\frac{R_{2W}}{R_{2W} + R_{1W}}\right) = \frac{m}{\left(1 + \frac{b}{V_{cc}}\right)} = \frac{0.833}{\left(1 + \frac{0.58}{3.3}\right)} \approx \frac{0.8}{1.1}$$

$$\therefore R_{2W} \approx 0.8, R_{1W} \approx 0.3$$

$$m = \left(\frac{R_{2W}}{R_{2W} + R_{1W}}\right) \cdot \left(1 + \frac{R_{fW}}{R_{gW}}\right)$$

$$\Rightarrow \frac{R_{fW}}{R_{gW}} = \left(\frac{m \cdot (R_{2W} + R_{1W})}{R_{2W}}\right) - 1 = \left(\frac{0.833 \times 1.1}{0.8}\right) - 1 \approx 0.15$$

$$\therefore R_{fW} \approx 0.15, R_{gW} \approx 1$$

The resistor values were rounded so that they would correspond with the e24 resistor scale when scaled. To produce achievable resistor values and reduce the input impedance the calculated values must be scaled. The values for R_{1W}, and R_{2W} are scaled by 10^5. The values for R_{gW}, and R_{fW} are scaled by $2x10^5$. The final values are given in the circuit shown in Figure 107.

Due to the rounding of the resistor values the range of the temperatures measured will changed the following calculations determine this range. Substituting the values into the formulae above for obtaining b and m gives the following.

$$b = V_{cc} \cdot \frac{R_{fW}}{R_{gW}} = 3.3 \times \frac{0.15}{1} = 0.495$$

$$m = \left(\frac{R_{2W}}{R_{2W} + R_{1W}}\right) \cdot \left(1 + \frac{R_{fW}}{R_{gW}}\right) = \left(\frac{0.8}{0.8 + 0.3}\right) \times \left(1 + \frac{0.15}{1}\right) = 0.766\ (3\ d.p.)$$

The values for the gain and offset are different from those specified for the range 20 °C to 80 °C. The temperature range is determined by the substituting the minimum and maximum value for V_{outw} into following formulae using the calculated gain and offset values. The minimum and maximum value for V_{outw} is 0 and 1.2 respectively.

$$V_{outw} = m \cdot V_{inw} + b$$

$$\Rightarrow V_{inw} = \frac{V_{outw} + b}{m}$$

$$V_{inw} = 0.02376 \cdot t_w + 0.11602$$

$$\Rightarrow\ t_w = \frac{V_{inw} - 0.11602}{0.02376}$$

$$\Rightarrow t_w = \left(\frac{V_{outw} + b}{0.02376 \cdot m}\right) - \left(\frac{0.11602}{0.02376}\right) = \left(\frac{V_{outw}}{0.02376 \times 0.766}\right) + \left(\frac{0.495}{0.02376 \times 0.766}\right) - \left(\frac{0.11602}{0.02376}\right)$$

$$\Rightarrow t_w = \frac{V_{outw}}{0.0182} + 22.315$$

$$t_w = \frac{0}{0.0182} + 22.315 = 22.3\ ℃\ \ (1\ d.p.)$$

$$t_w = \frac{1.2}{0.0182} + 22.315 = 88.2\ ℃\ \ (1\ d.p.)$$

This gives the range 22 °C to 88 °C that the signal conditioning circuit will produce using the calculated resistor values.

The XBee modules ADC is of 10 bit resolution meaning it produces values between 0 and 1024. To convert an ADC value to a temperature a conversation formulae is required. The formulae given above specifies the output voltage V_{outw} for the temperature sensor, the following formulae describes the relationship between the output voltage V_{outw} given to the XBee modules ADC and the numerical value obtained from the ADC. From this a relationship between the temperature of the solar heater and the XBee Modules ADC value.

$$ADC = \frac{1024 \cdot V_{outw}}{1.2} \Rightarrow V_{outw} = \frac{1.2 \cdot ADC}{1024}$$

$$t_w = \frac{V_{outw}}{0.0182} + 22.315 = \frac{1.2 \cdot ADC}{1024 \times 0.0182} + 22.315$$

$$\Rightarrow t_w = \frac{1.2}{18.6368} \cdot ADC + 22.315$$

The formulae above can be used to obtain a temperature of the solar heater for a given ADC value.

The circuit was simulated in LTSpice to check the output was correct for the input range, the result of the simulation is shown in Figure 108.

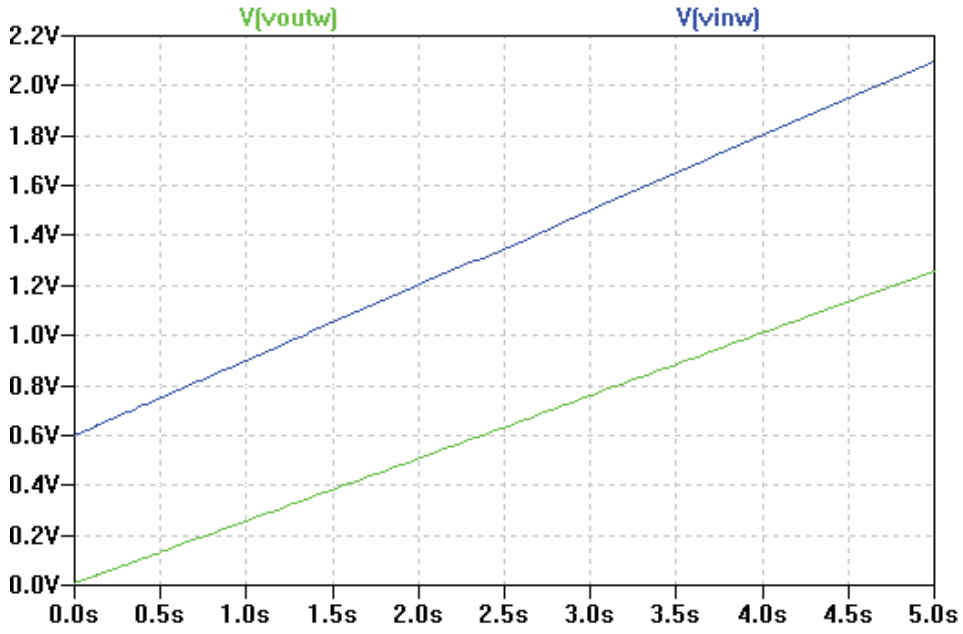

Figure 108 LTSpice simulation results for water cylinder temperature probe voltage signal conditioning

Figure 109 shows the output voltage versus the indicated temperature on the solar heater controller for the solar heater. The trend line fitted to the data shows a good linear relationship between the output voltage and temperature. The formulae from the trend line in Figure 109 to give the output voltage (v) of the probe for a given temperature (T) is as following.

$$V_{ins} = 0.02 \cdot t_s + 2.8$$

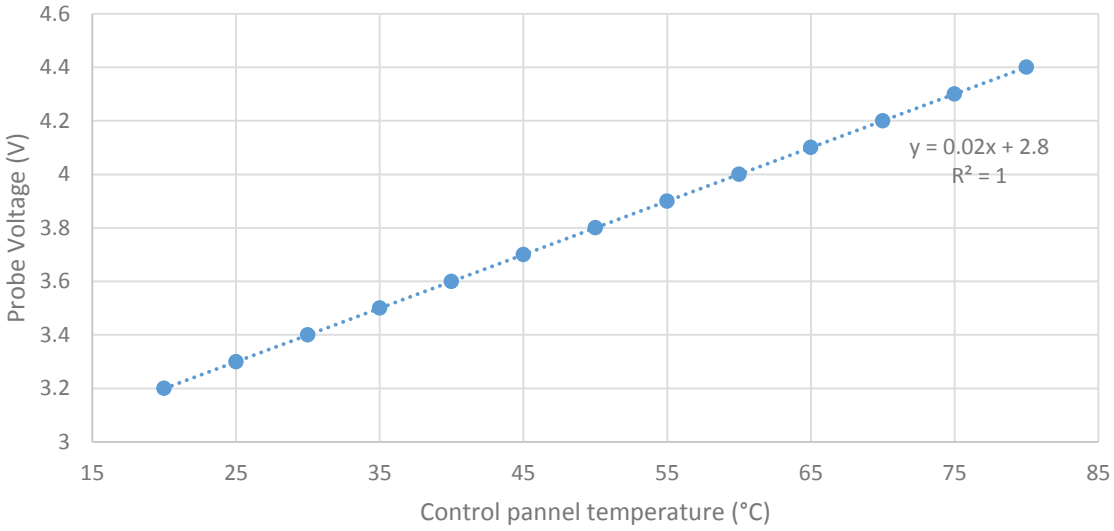

Figure 109 Graph of temperature Vs. voltage for solar heater probe

The range of temperatures that need to be measured is from 20 °C to 80 °C, substituting these values into the formulae above gives the following.

$$v = 0.02 \times 20\,°C + 2.8 = 3.2\,V$$

$$v = 0.02 \times 80\,°C + 2.8 = 4.4\,V$$

This gives the output range 3.2 V to 4.4 V which must be scaled to the input range 0 V to 1.2 V of the XBee Module. This means the signal needs to be negatively offset by 3.2 V. To achieve this the circuit and formulae used above are used with the following assumptions; offset b is 3.2, the gain m is 1 and the reference voltage V_{ref} is 5.0. The variables used in the following formulae are shown in Figure 110.

$$\left(\frac{R_{2S}}{R_{2S}+R_{1S}}\right) = \frac{m}{\left(1+\frac{b}{V_{ref}}\right)} = \frac{1}{\left(1+\frac{3.2}{5.0}\right)} = \frac{1}{1.64} \approx \frac{2}{3.3}$$

$$\therefore R_{2S} \approx 1, R_{1S} \approx 0.64$$

$$\frac{R_{fW}}{R_{gW}} = \left(\frac{m \cdot (R_{2W}+R_{1W})}{R_{2W}}\right) - 1 = \left(\frac{1 \times 1.64}{1}\right) - 1 = 0.64 \approx \frac{1.3}{2}$$

$$\therefore R_{fS} \approx 1.3, R_{gS} \approx 2$$

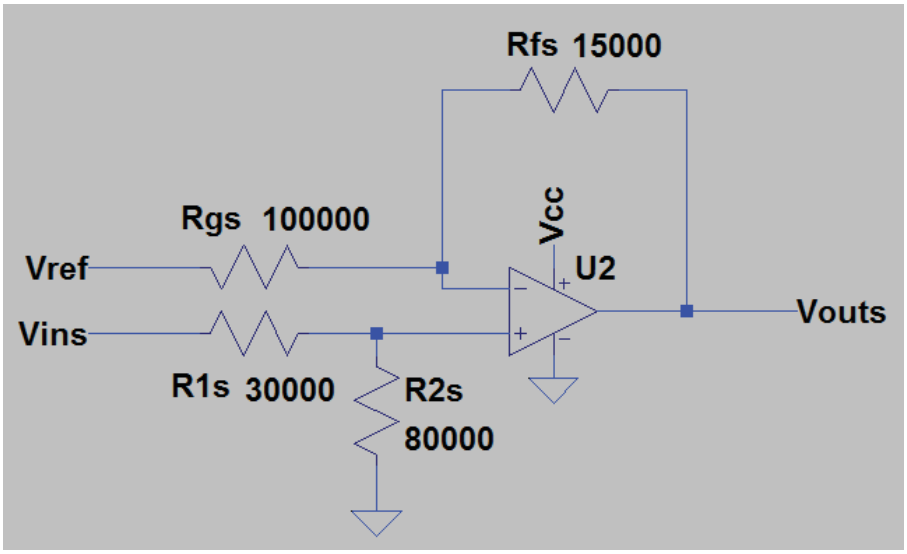

Figure 110 Schematic for circuit to condition signal from solar heater probe

The resistor values need to be scaled and rounded to produce values that correspond with the e24 resistor scale. This is why the resulting resistor values were multiplied by 2 and rounded to one decimal place. Multiplying by 10^5 the resistor values will produce a value that will be on the e24 resistor scale.

The resistor values are not equal to the calculated resistor values meaning the gain and offset are different creating a different temperature range. The new temperature range is determined by obtaining the gain and offset from the resistor values, then using the minimum (0V) and maximum (1.2v) input voltages. The following determines the gain and offset from the resistor values.

$$b = V_{ref} \cdot \frac{R_{fS}}{R_{gS}} = 5 \times \frac{1.3}{2} = 3.25$$

$$m = \left(\frac{R_{2S}}{R_{2S} + R_{1S}}\right) \cdot \left(1 + \frac{R_{fS}}{R_{gS}}\right) = \left(\frac{1}{1 + 0.64}\right) \times \left(1 + \frac{1.3}{2}\right) = 1.006\ (3\ d.p.)$$

Using this gain and offset in the equation for the output voltage, combined with the equation for the input voltage in relation to the temperature gives the following.

$$V_{outs} = m \cdot V_{inw} + b$$

$$\Rightarrow V_{ins} = \frac{V_{outs} + b}{m}$$

$$V_{ins} = 0.02 \cdot t_s + 2.8$$

$$\Rightarrow t_s = \frac{V_{ins} - 2.8}{0.02}$$

$$\Rightarrow t_s = \left(\frac{V_{outs} + b}{0.02 \cdot m}\right) - \left(\frac{2.8}{0.02}\right) = \left(\frac{V_{outs}}{0.02 \times 1.006}\right) + \left(\frac{3.25}{0.02 \times 1.006}\right) - \left(\frac{2.8}{0.02}\right)$$

$$\Rightarrow t_s = \frac{V_{outs}}{2.012 \times 10^{-2}} + 21.531$$

$$t_s = \frac{0}{2.012 \times 10^{-2}} + 21.531 = 21.5\ °\text{C}\ \ (1\ d.p.)$$

$$t_s = \frac{1.2}{2.012 \times 10^{-2}} + 21.531 = 81.2\ °\text{C}\ \ (1\ d.p.)$$

This gives the range 21.5 °C to 81.2 °C for the signal conditioning circuit when the calculated resistor values are used.

A conversion function between the ADC value produced by the XBee Module and the temperature is given by the following.

$$ADC = \frac{1024 \cdot V_{outs}}{1.2} \Rightarrow V_{outw} = \frac{1.2 \cdot ADC}{1024}$$

$$t_s = \frac{V_{outs}}{2.012 \times 10^{-2}} + 21.531 = \frac{1.2 \cdot ADC}{1024 \times 2.012 \times 10^{-2}} + 21.531$$

$$\Rightarrow t_s = \frac{1.2}{2.0603} \cdot ADC + 22.315$$

The current transformer requires signal conditioning as the output is an AC waveform representing the AC current flowing to the hot water cylinders heating coil. The AC waveform has peak values less than 100 mV and is sinusoidal in nature. The approximate RMS value of this waveform needs to be measured meaning the waveform needs to be scaled and then filtered to obtain a DC value.

Half wave rectification of the wave form produced by the current transformer was achieved using an op amp with a single rail power supply. The half wave was then filtered to obtain an average using a low pass filter. Assuming the waveform is sinusoidal the average value obtained from the low pass filter can be determined as following.

As the wave is half rectified we are only measuring the positive part of the sinusoidal wave, therefore we are obtaining the average between 0 and π of the sine function. To find the normalised average, the area under the sine function between 0 and π is divided by 2π as following. Multiplying the normalised average by the peak voltage (V_p) then gives the average voltage for half wave rectified sine wave.

$$V_{avg} = \frac{\left(V_{pk} \cdot \int_0^{\pi} \sin(x)\, dx\right)}{2\pi} = \frac{\left(V_{pk} \cdot [-\cos(x)]_0^{\pi}\right)}{2\pi} = \frac{\left(V_{pk} \cdot (-\cos(\pi) + \cos(0))\right)}{2\pi} = \frac{V_{pk}}{\pi}$$

The current transformer used was an ASM 010 [99] which outputs approximately 3 mV per Ampere. The following equation was derived from the data sheet, where V_{rms} is the rms output voltage of the

current transformer, V_{pk} is the peak output voltage of the current transformer, and I_{rms} is the RMS current supplied to the cylinder heater coil. A scaling factor (m_{pk}) is used to scale the peak voltage for the input of the XBee Module (V_{pks})

$$V_{rms} = \frac{I_{rms}}{3000}, V_{pk} = \sqrt{2} \cdot V_{rms}, V_{pks} = m_{pk} \cdot V_{pk}$$

$$V_{pk} = \frac{\sqrt{2} \cdot I_{rms}}{300}, V_{pks} = \frac{\sqrt{2} \cdot I_{rms} \cdot m_{pk}}{300}$$

Substituting the formulae for V_{pks} into the formulae for V_{ave} gives the relationship between the average scaled voltage to be output to the XBee module and the current seen by the current transformer.

$$V_{avg} = \frac{V_{pk}}{\pi} = \frac{\sqrt{2} \cdot I_{rms} \cdot m_{pk}}{300\pi}$$

The hot water cylinder coil is rated at 3 kW meaning that it uses 12.5 Amperes RMS at 240 Volts RMS. To ensure the maximum value of 12.5 Amperes RMS is included in the measurement range, 13 Amperes RMS was chosen as the maximum value. Substituting the maximum current of 13 Amperes (I_{rms}), and the expected average output voltage (V_{ave}) of 1.2 Volts gives the required gain m_{pk} to scale the input signal of the current transformer.

$$V_{avg} = \frac{\sqrt{2} \cdot I_{rms} \cdot m_{pk}}{300\pi} \Rightarrow m_{pk} = \frac{V_{avg} \cdot 300\pi}{\sqrt{2} \cdot I_{rms}}$$

$$\Rightarrow m_{pk} = \frac{1.2 \times 300 \times \pi}{\sqrt{2} \times 13} = 61.52\ (2\ d.p.) \approx 62$$

The average voltage is fed into the XBee modules ADC converter, therefore the following conversion function was derived to give the RMS current (I_{rms}) for a given ADC value.

$$ADC = \frac{1024 \cdot V_{avg}}{1.2} \Rightarrow V_{avg} = \frac{1.2 \cdot ADC}{1024}$$

$$V_{avg} = \frac{\sqrt{2} \cdot I_{rms} \cdot m_{pk}}{300\pi} \Rightarrow I_{rms} = \frac{V_{avg} \cdot 300\pi}{\sqrt{2} \cdot m_{pk}}$$

$$I_{rms} = \left(\frac{1.2 \cdot ADC}{1024}\right) \cdot \left(\frac{300\pi}{\sqrt{2} \cdot m_{pk}}\right) = \frac{1.2 \times 300 \times \pi \times ADC}{1024 \times \sqrt{2} \times 62}$$

$$I_{rms} = 1.26 \times 10^{-2} \times ADC$$

An inverting op-amp circuit was used to achieve the gain of 62 as determined by the previous equation. The following determines the resistors (R_f, and R_{in}) needed to create an inverting op-amp with a gain of 62.

$$V_{out} = V_{in} \cdot m_{pk} = V_{in} \cdot \frac{R_f}{R_{in}}$$

$$\Rightarrow \frac{R_f}{R_{in}} = m_{pk} = 62$$

$$\therefore R_f = 62, R_{in} = 1$$

The low pass filter was designed with an arbitrary cut-off frequency of 0.5 Hz in order to obtain an average voltage of the half sinusoidal wave without a significant amount of ripple or delay. The following shows the calculation of the capacitor for the filter assuming the frequency (f_c) is 0.5, and the resistor (R) is 12000 ohms.

$$f_c = \frac{1}{2\pi RC}$$

$$\Rightarrow C = \frac{1}{2\pi R f_c} = \frac{1}{2 \times \pi \times 12000 \times 0.5} \approx 27\mu F$$

5.4.2 Address Translation

An XBee module needs to be addressable from the internet so that it can communicate with the internet. In order for an XBee node to addressable from the internet it requires an internet address. The most commonly used addressing format is IPv4, however the IPv4 address space is not large enough to contain XBee addresses. The address space is given by the number of bits used to create an address, IPv4 uses 32 bits to address a node. XBee uses 64 bits to address a unique node in a network and 16 bits to identify the network (PAN ID), therefore 80 bits are required to address an XBee node. The XBee nodes can also be addressed in a shorter 16 bit format, however it is a dynamically assigned address and is not unique to that XBee node. Due to the limited number of addresses in IPv4 a new standard was created called IPv6, which uses 128 bits to address a node. This means that the unique 80 bit address for an XBee node can be contained within a 128 bit IPv6 address, which can be used to give the XBee node an IPv6 address.

There are 4 elements required to assign an XBee node an IPv6 address, these elements are shown in Figure 111. The first element identifies the gateway, which is the first 48 bits of the IPv6 address assigned to the gateway. The PAN ID given by the coordinator of the XBee network is the second

element, which is 16 bits long. The last two elements are the serial high and low of the XBee node, these are 32 bits in length each. The serial high and low are a unique number assigned to an XBee module when it is manufactured [89]. All of these elements are combined to produce an IPv6 address that identifies an XBee module connected to a Coordinator with a given PANID, which is connected to a Gateway with a given Network ID.

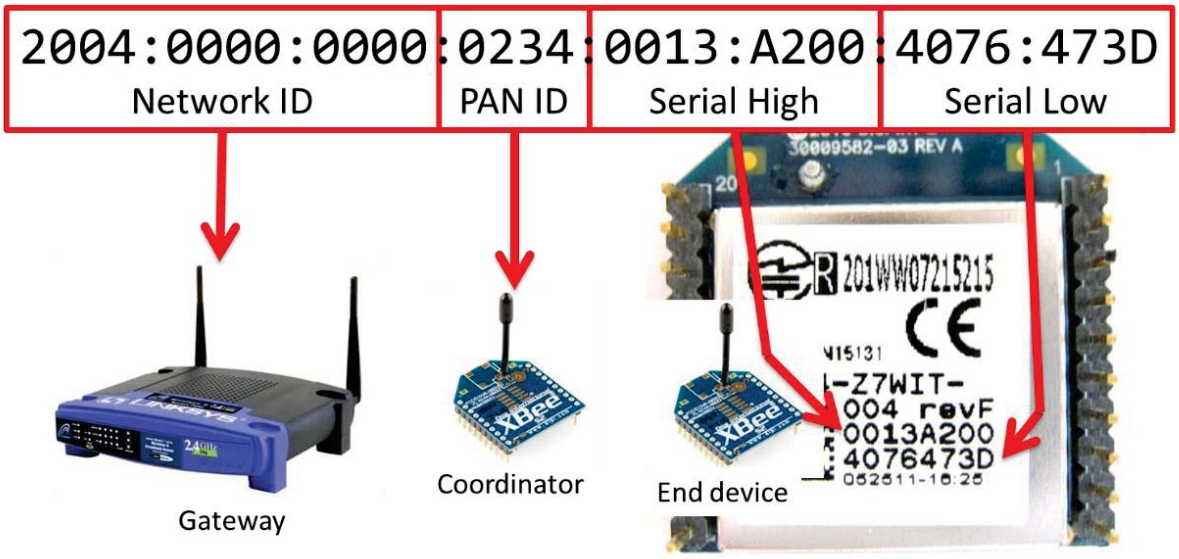

Figure 111 XBee to IPv6 Address translation technique

5.4.3 IPv6 UDP Encapsulation

Packets originating and destined for an XBee network can be encapsulated in an IPv6 UDP packet. In order to send and receive XBee packets encapsulated in IPv6 UDP packet various elements must be placed into the packet, such as the Address of the XBee module, the type of data contained the in the payload and the payload itself.

In an XBee network, IEEE 802.15.4 packets are used to configure and maintain the wireless network and to transport ZigBee packets between XBee modules. Configuration and maintenance of the wireless network is performed by the XBee modules based on their parameters, such as PAN ID, operating channel and role in the network. ZigBee packets are used to exchange information between modules which are commands, and notifications. Notifications can either be generated automatically by the module or in response to a command. Access to notifications and commands is provided by a structured packet format used with the serial interface, known as the XBee API.

There are two basic types of API packets that are produced by an XBee coordinator, which are notification and command. Notification packets consist of network status messages, command response messages, and messages containing digital and analogue samples. Command packets consist of commands to get information from an XBee module or set a modules' parameters. Command

packets therefore generate a response from the XBee module in the form of a notification message. Notification and command messages can be encapsulated into an IPv6 UDP packet.

An IPv6 UDP packet contains a number of parameters to facilitate the transport of a payload from application to application over a network. Table 11 shows the size and organisation of parameters contained in an IPv6 header and UDP header. The first parameter indicates which version of the IP protocol is used. The Traffic Class and Flow Label parameters are not used as the deal with the quality of service which is not needed for XBee API packet encapsulation. The payload length is the number of bytes in the UDP header and payload. The next header parameter dictates the type of header following the IP header, which is a UDP header. The Hop limit parameter determines the number of hops the packet can pass through. The source address contains the 128 bit IPv6 address of the source of the UDP packet, and the destination contains the 128 bit IPv6 address of the destination. The next four parameters belong to the UDP header. The source and destination port are used to multiplex the payload to the required application. The length field is the length of the payload in bytes. The checksum parameter is a 2's complement sum of the payload and pseudo header. The payload of a UDP packet follows after the checksum.

Table 11 IPv6 and UDP header structure

<table>
<tr><th>Section</th><th>Bytes</th><th colspan="2">1</th><th>2</th><th>3</th><th>4</th></tr>
<tr><td rowspan="10">IPv6</td><td>0</td><td>Version</td><td colspan="2">Traffic Class</td><td colspan="3">Flow Label</td></tr>
<tr><td>4</td><td colspan="4">Payload Length</td><td>Next Header</td><td>Hop Limit</td></tr>
<tr><td>8</td><td colspan="5" rowspan="4">Source Address</td></tr>
<tr><td>12</td></tr>
<tr><td>16</td></tr>
<tr><td>20</td></tr>
<tr><td>24</td><td colspan="5" rowspan="4">Destination Address</td></tr>
<tr><td>28</td></tr>
<tr><td>32</td></tr>
<tr><td>36</td></tr>
<tr><td rowspan="2">UDP</td><td>40</td><td colspan="4">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td>44</td><td colspan="4">Length</td><td colspan="2">Checksum</td></tr>
<tr><td>Payload</td><td>49 to1280</td><td colspan="6">UDP Payload</td></tr>
</table>

The IPv6 source and destination addresses in the IPv6 UDP headers are determined by the type of API packet. For an XBee command API packet the IPv6 source address is the IPv6 address of the network device issuing the command, and the IPv6 destination address is the IPv6 address of the XBee module (determined using the method described in the previous section). For an XBee notification API packet the source address is the XBee modules IPv6 address (using the method described in the previous section) and the destination address is the predetermined IPv6 address of a server. The source and

destination port of the UDP header are used to store the type of XBee packet, and the UDP payload contains the XBee packet payload.

When an XBee end device takes a sample it sends the sample to the coordinator, and the coordinator then produces an XBee S2 API packet that contains the sample information and the source address of the module that produced the sample. This API packet is then received on the serial port of the IoT gateway (WRT54GL) and is translated to an IPv6 UDP packet. The IoT gateway then sends this IPv6 UDP packet to the server, where a program listening for the packets decodes and stores the sample information in a database. The flow of information from the XBee end device to the Server is show in Figure 112.

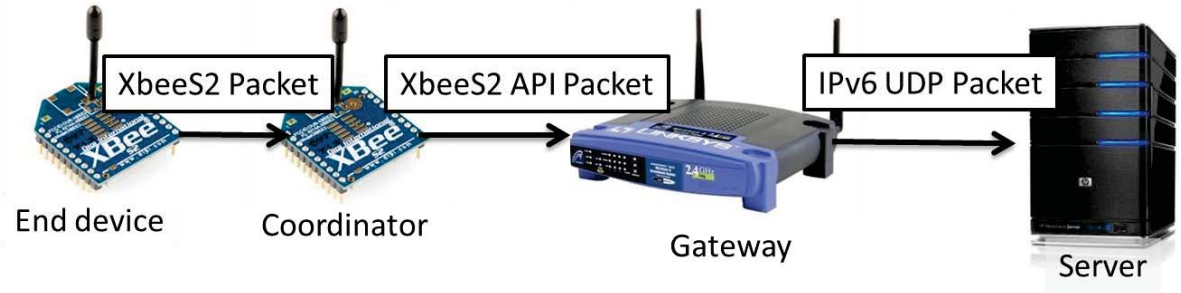

Figure 112 Flow of sample information from XBee end device to Server

To achieve this flow of information the IoT gateway is required to convert XBee S2 API packets to IPv6 UDP packets. The placement of information from the XBee S2 packet into the IPv6 UDP packet is shown in Figure 113.

Xbee S2 packet	
Packet type	Sample
Source Address	Serial number
Data	Sample data

IPv6 UDP packet	
Source Address	Network ID, PAN ID, and Serial number
Destination Address	Server address
Source Port	Packet type
Destination Port	Packet type
Payload	Sample data

Figure 113 Converting an XBee S2 API packet to an IPv6 UDP packet

Control of XBee modules can be achieved from the server, and this requires data to flow from the server to the XBee module as shown in Figure 114. XBee S2 API command messages are used to

change parameters of the XBee module such as pin assignments and sampling rates. These commands can be encapsulated into an IPV6 UDP packet that is sent from the server to the IoT gateway. The UDP packet is then translated to an XBee API command packet by the IoT gateway and sent to the XBee coordinator attached to the IoT Gateway.

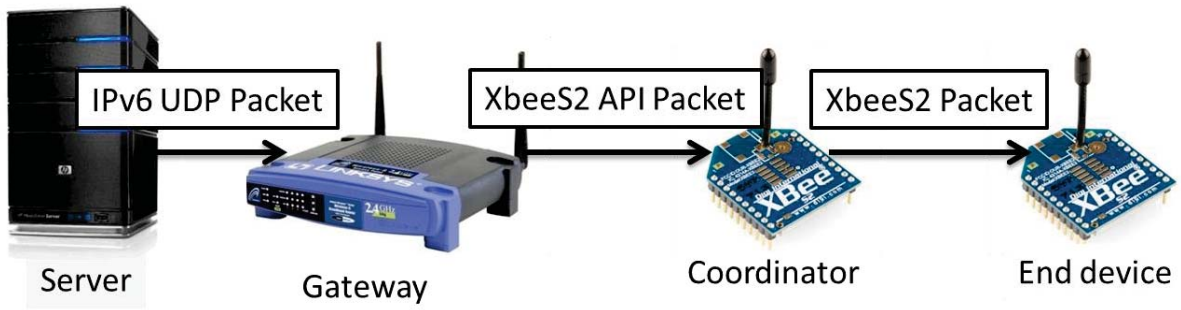

Figure 114 Flow of information from Server to XBee Module

The IoT Gateway must convert IPv6 UDP packets to XBee API command packets in order for the information to flow from the server to the XBee module. The placement of information in an IPv6 UDP packet to contain an XBee API command packet is shown in Figure 115.

IPv6 UDP packet		Xbee S2 packet	
Source Address	Server Address	Packet type	Command request
Destination Address	Network ID, PAN ID, and Serial number	Frame ID	NAT ID
		Destination Address	Network ID, PAN ID, and Serial number
Source Port	Packet type		
Destination Port	Packet type	Data	Control data
Palyoad	Command data		

Figure 115 Converting an IPv6 UDP command packet to an XBee API command packet

The software components involved in the data flow between the XBee module and server are shown in Figure 116. The blue coloured blocks are software elements that were custom developed for the IoT integrated platform.

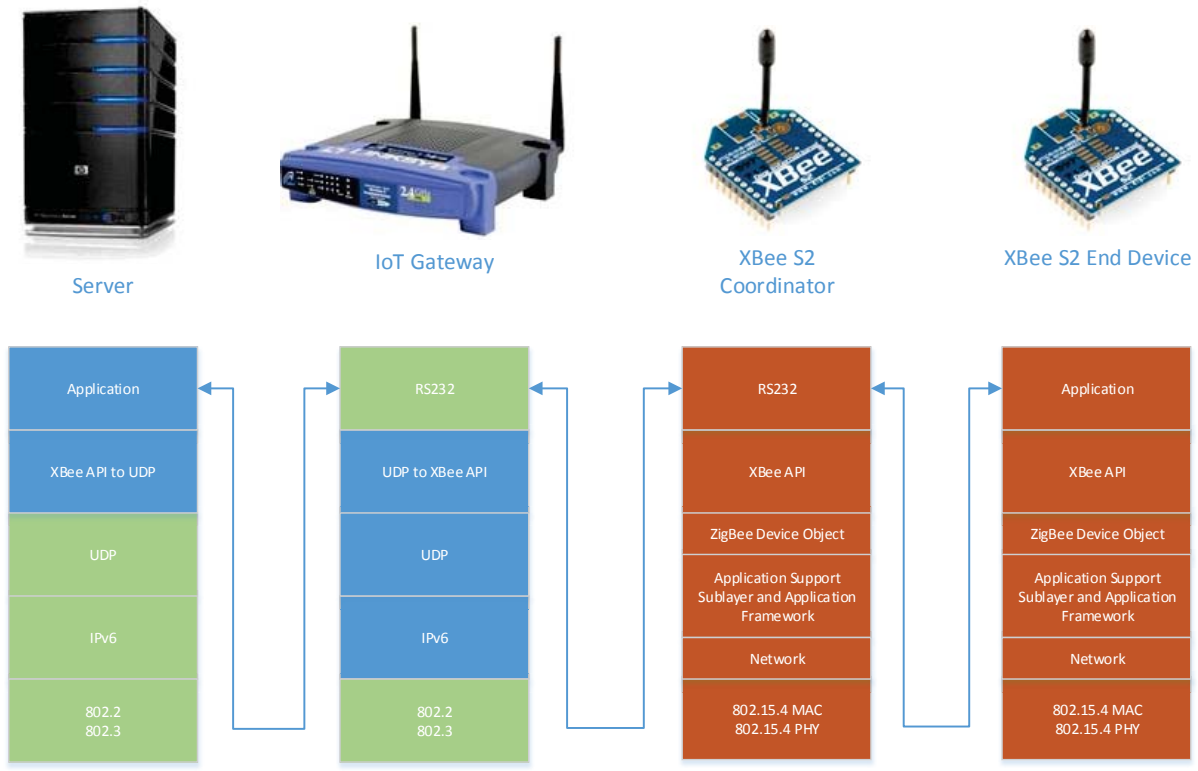

Figure 116 Flow of information through software and hardware elements between XBee module and Server

The techniques to obtain information from XBee sensor modules, and set parameters of the XBee modules from a server over the internet have been defined in this section. The next section shows how these techniques are used in the software implementation on the IoT gateway to allow information to flow between the server and XBee modules.

5.4.4 Software implementation on IoT Gateway for XBee

Customised software for the IoT Gateway was created to translate between the XBee network and the IPv6 network. The software was designed to run in Linux meaning it can run in OpenWRT on the WRT54GL and requires the hardware interface previously described. The Raspberry PI also runs Linux meaning that the same software also worked on the Raspberry PI. The software was created using the toolchain and IDE described in section 5.1.2.

The following flowchart in Figure 117 gives and overview of the internal processes in the software that give the functionality required to create an IoT gateway. Each process is in a block, which are linked together with arrows. Decisions are in diamonds with the outcomes labelled on the arrows exiting the diamond.

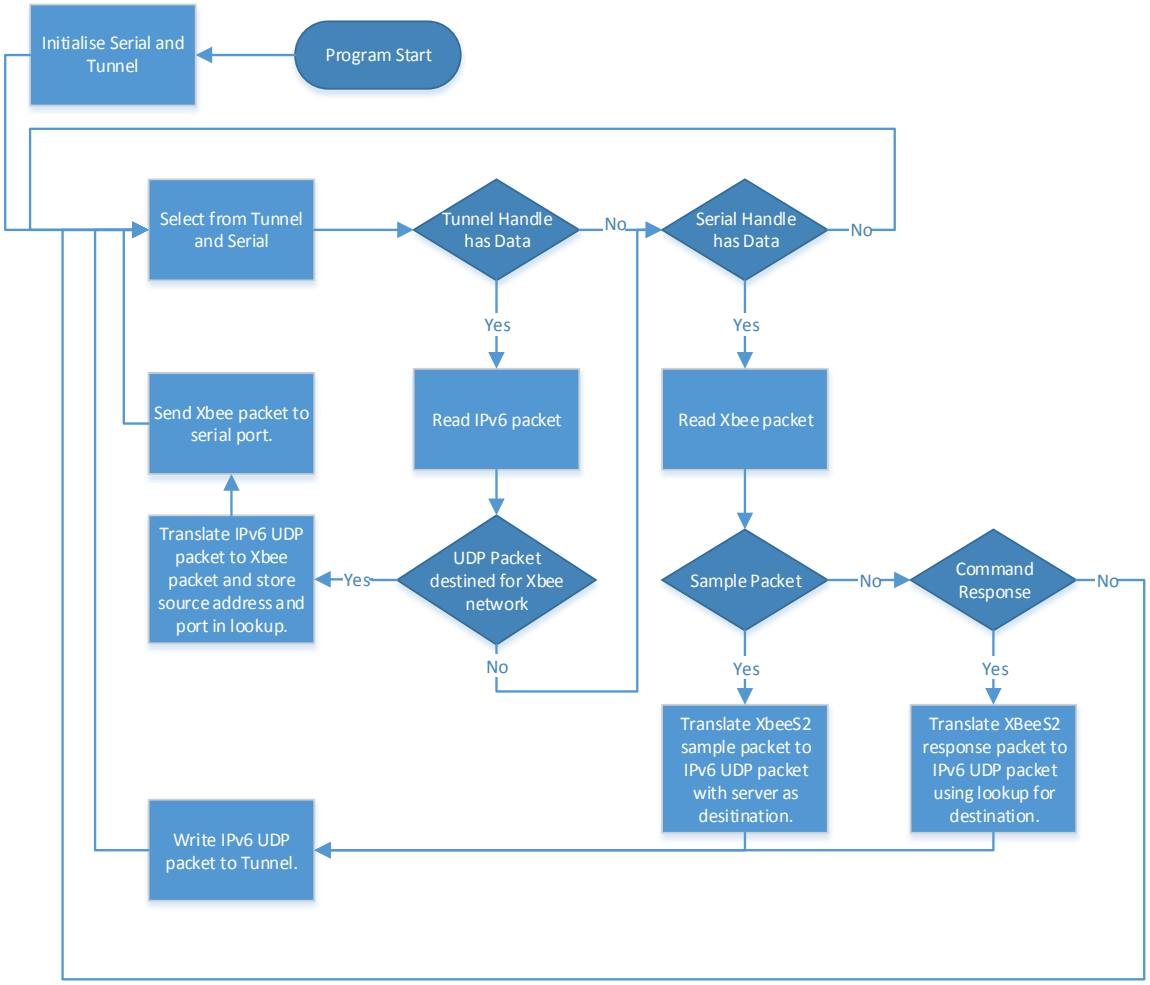

Figure 117 Flowchart of XBee IoT gateway custom software implementation

5.4.4.1 Initialise Serial and Tunnel

The Initialise serial and Tunnel block performs the initial setup required to interface with the tunnel driver and serial port. The following steps are executed in this block:

1. Initialisation of the serial port with the correct baud rate, and options. The location of the serial port is provided as the first parameter to the program in the command line.
2. Test the serial connection by reading parameters from the XBee S2 coordinator attached to the serial port. The parameters read are the PAN ID, serial high and serial low (ID, SH, SL) and these are used to setup the tunnel driver.
3. Initialise the tunnel driver by performing the following.

a. Set the IPv6 address of the tunnel driver to a translated XBee address of the coordinator attached to the serial port. To do set the IPv6 address the following command is executed by the program.

   ifconfig [tun] add [ipv6]/[subnet]

   Where [tun] is the name of the tunnel adapter, [ipv6] is the ipv6 address of the coordinator, and [subnet] is the subnet mast of the network.

b. Add a route to the subnet which represents the Xbee network created by the XBee coordinator attached to the serial port. To add the route the following command is executed.

   route -A inet6 add [ipv6]/[subnet] dev [tun]

5.4.4.2 Select from Tunnel and Serial

Linux provides multiplexing capabilities for from reading multiple file devices from a single process. This functionality is accessed the select function. The select function is given the file handles of the tunnel driver and the serial port, and then blocks the process execution until data is available on any of the file handles. This allows a single process to service multiple file handles without the need for complex multithreading.

The flowchart in Figure 118 elaborates on the process to select from the serial and tunnel file handles. The “Zero Select Structure” and “Add tunnel and serial file handle…” blocks initialise a structure used by the select function. This structure is then passed to a select function (this is the block “Pass select structure to select function”) which blocks until there is data available to read on either of the file handles.

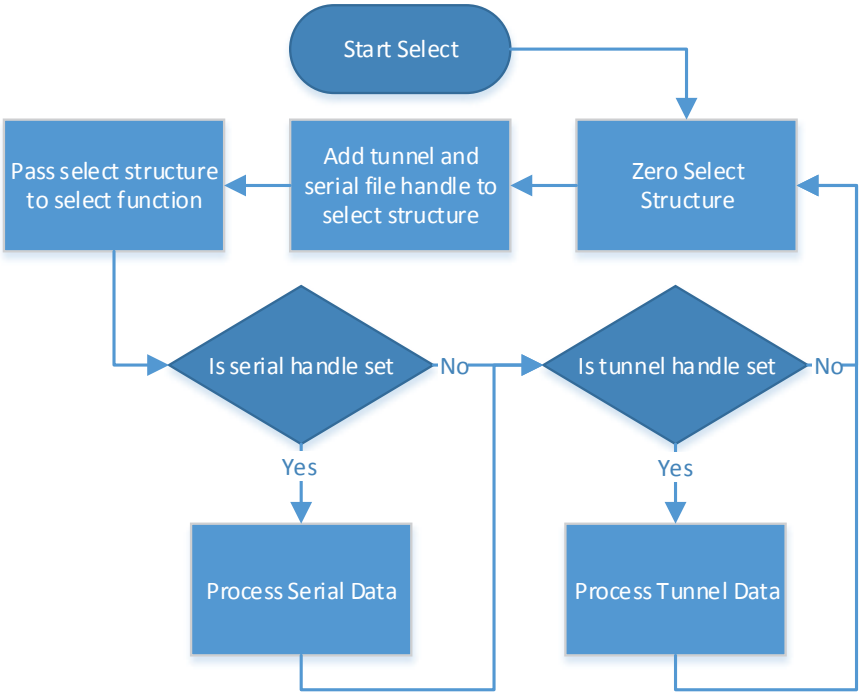

Figure 118 Flowchart of select process in XBee IoT gateway software

5.4.4.3 Processing Tunnel Data

When the select function determines there is data available to be read from the tunnel file handle, the process in Figure 119 is used. The process extracts IPv6, and UDP headers from the data and creates an XBee API command from this information and the UDP payload. In order for a response message to be sent back to source of the command request the address and port of the source are stored in a lookup table.

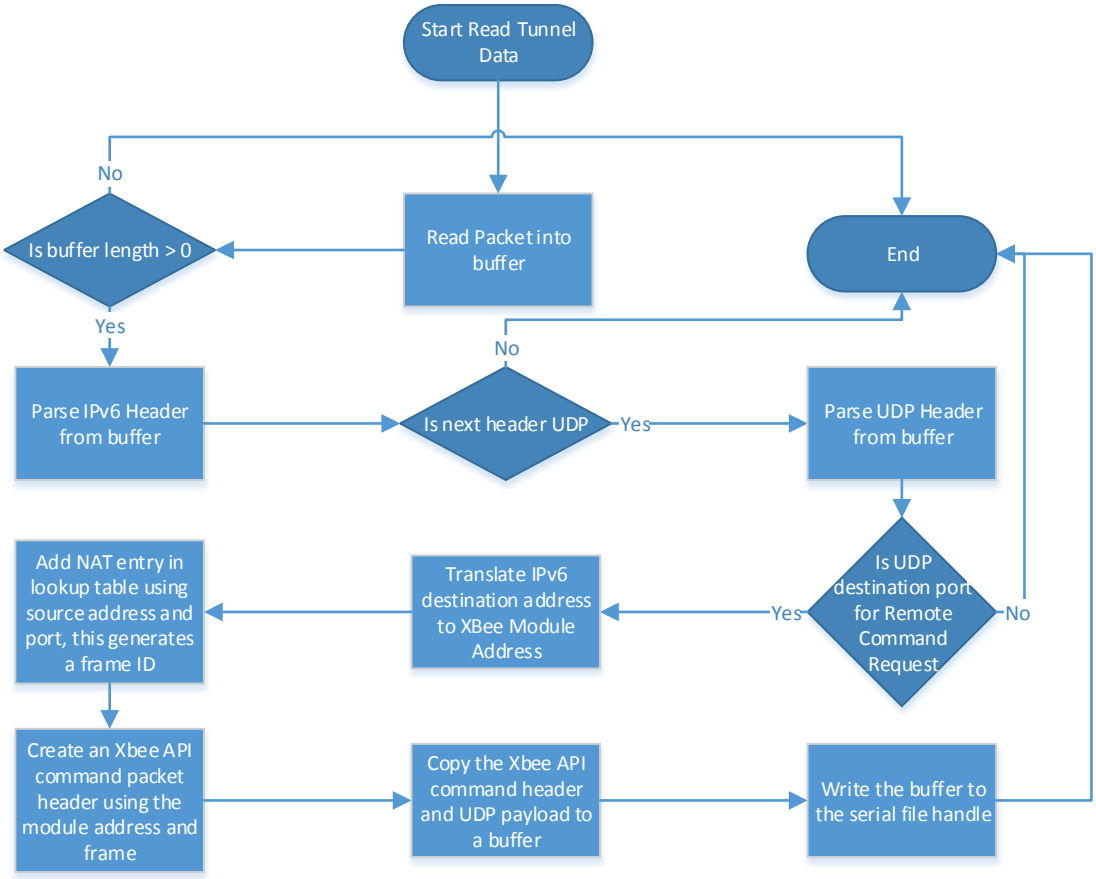

Figure 119 Flowchart of reading tunnel data in XBee IoT Gateway software

The details of each element in Figure 119 are as following, they are in the order of execution in the software.

- "Read Packet into buffer" performs the read operation from the tunnel file handle into a buffer. When reading from a tunnel the amount of data read is not the size of the buffer specified, but the amount of data available to be read from the handle. This means that the buffer used is the maximum size that an IPv6 packet can be on the interface (1500 bytes). The read function returns the actual number of bytes read, and hence the length of the IPv6 packet.
- "Is buffer length > 0" block checks that data was written into the buffer. If there was no data written an error has occurred and the process is stopped.

- “Parse IPv6 Header from buffer” extracts the IPv6 header into a structure from the buffer read from the tunnel file handle. Parsing the header uses the technique described in section 5.3.2. The IPv6 header should contain the IPv6 address of the XBee module, which will be used later to create the XBee API packet.
- “Is next header UDP” checks if the next header field in the IPv6 header contains the value 17 which means the next header is a UDP header. If the next header is not a UDP header the process ends.
- “Parse UDP header from buffer” extracts the UDP header into a structure from the buffer using the technique described in section 5.3.2. The UDP header contains the source and destination port, and the length of the payload. The destination port is used to determine what type of API message to send to the XBee coordinator module. At present only the remote command request API message is supported, additional commands can be added in this process.
- “Is UDP destination port for Remote Command Request” checks the UDP destination port. If the destination port is not a predetermined port for a Remote Command Request then the process ends.
- “Translate IPv6 destination address to XBee Module Address” uses the technique in section 5.4.2 for translation the IPv6 address. This uses the destination address contained in the IPv6 structure parsed from the buffer.
- “Add NAT entry in lookup table using source address and port…” allows for the response to the command to be sent back to the source of the command. To create a response IPv6 UDP packet the destination address and destination port must be known, these are taken from the lookup table entry generated in this step. To know which entry to use the frame ID is used as a key for each entry. The frame ID is a field in the XBee API Remote Command Request header.
- “Create an XBee API command packet header…” combines information from the IPv6 header, UDP header and UDP payload in order to create a remote command request XBee API packet. A general XBee API header is created with the length of the payload and the command id of a remote command request, and is placed in a buffer. The remote command request is created using the IPv6 header and UDP payload, and then placed in the buffer after the general header. Lastly the payload of the remote command request is taken form the UDP payload. The result is a buffer that contains the raw XBee API packet for a remote command request with parameters specified using the IPv6 UDP packet.
- “Write the buffer to the serial file handle” is described in the next paragraph. Writing the buffer to the serial file involves escaping some of the characters and appending a checksum.

To write an XBee API packet to the serial port the process in Figure 120 is used. The process consists of creating the XBee general API header, putting the header at the beginning of buffer, appending a checksum to the buffer, and then writing the buffer to the serial port. When writing the buffer certain characters need to be escaped because the coordinator is in escaped API mode. The escaping of characters ensures that the start character 0x7e for XBee API packets only occurs at the start of a packet, and not inside the packet. This makes detection of the start of the packets more reliable.

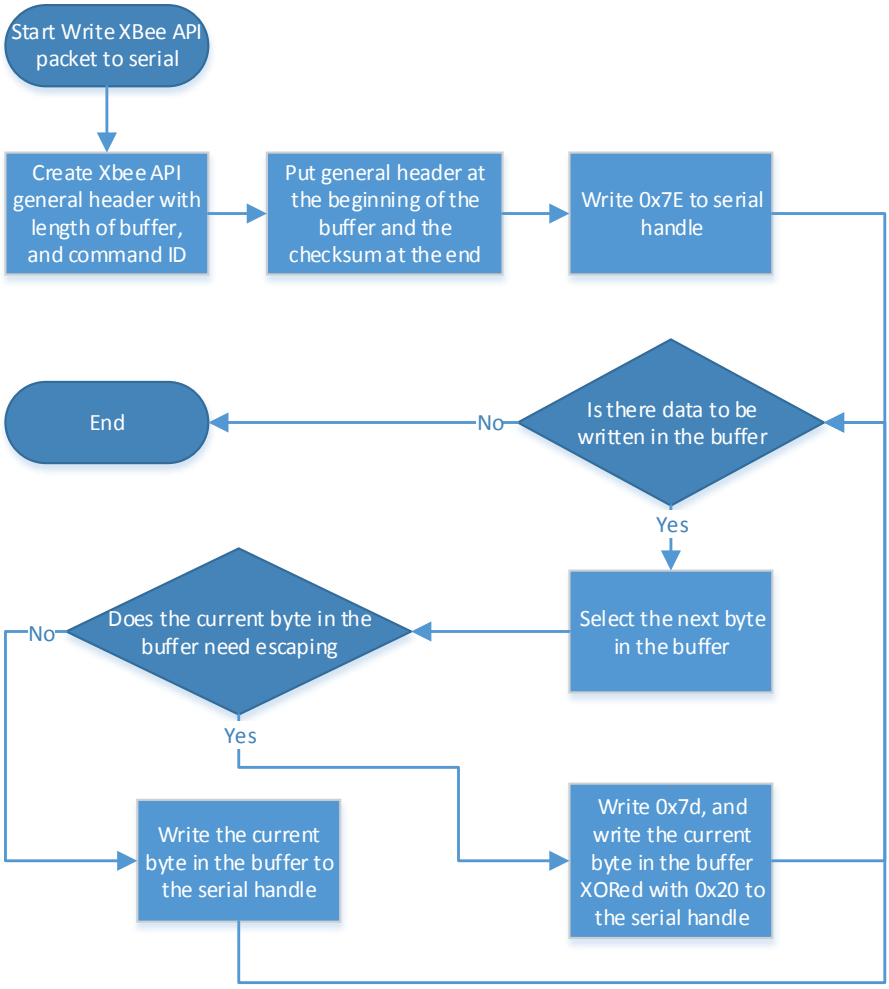

Figure 120 Flowchart for writing an XBee API packet to the serial port

5.4.4.4 Serial Has Data

When data is available from the serial file handle it will be a notification XBee API packet sent via serial by the XBee coordinator. Every XBee API packet contains a general header which is used to identify the start of an XBee API packet, the length of the packet, and the type. This must be read from the serial buffer in order to determine how many bytes to read to obtain the full XBee API packet and how to parse the contents of the packet. Once the header is read, the payload of the XBee API packet can be read into a buffer.

The flowchart in Figure 121 shows the process of reading the XBee API packet from the serial port. The details of each element shown in Figure 121 are as following, they are in order of executing in the software.

- The “Read byte…” and “Is byte 0x7E” detect the start of an XBee API packet by continually reading a byte from the serial handle until the 0x7E character is read. The byte 0x7E will only occur at the beginning of an XBee API packet when in API mode and escaping is enabled.
- The “Create a Buffer…” block allocates a buffer with the length of a general header, which is 4 bytes.
- The next five elements after the “Create a buffer…” block create a loop that reads the rest of the header. In the loop the decision “Is byte 0x7D” checks if an escape character has been read, if so the next character read must be escaped, otherwise the current character is put into the buffer. The loop continues until the buffer is full.
- “Parse XBee API General Header” parses the header into a structure using the technique in section 5.3.2. The general header contains the length of the payload which is needed in order to determine how much data must be read form the serial handle.
- “Extend buffer…” allocates more memory to the buffer so that it can contain the payload of the packet. The amount of memory to allocate is determined by the length field in the general header.
- The next five elements after “Extend buffer…” perform a loop to read the payload of the XBee API packet. This is the same process as the next five elements after “Create a buffer…” described above.
- “Calculate XBee API…” determines the checksum from the buffer. The checksum is calculated using the method described in the XBee Product Manual [89].
- “Checksum OK” compares the calculated checksum with the checksum in the buffer. If they are similar the process completes.
- “Clear buffer…” occurs if the checksum is not correct. Clearing the buffer is required as the contents of the buffer is not correct. An error is generated and is only displayed when the software is started with a debugging parameter. Additional error handling is required and depends on the amount of reliability required.

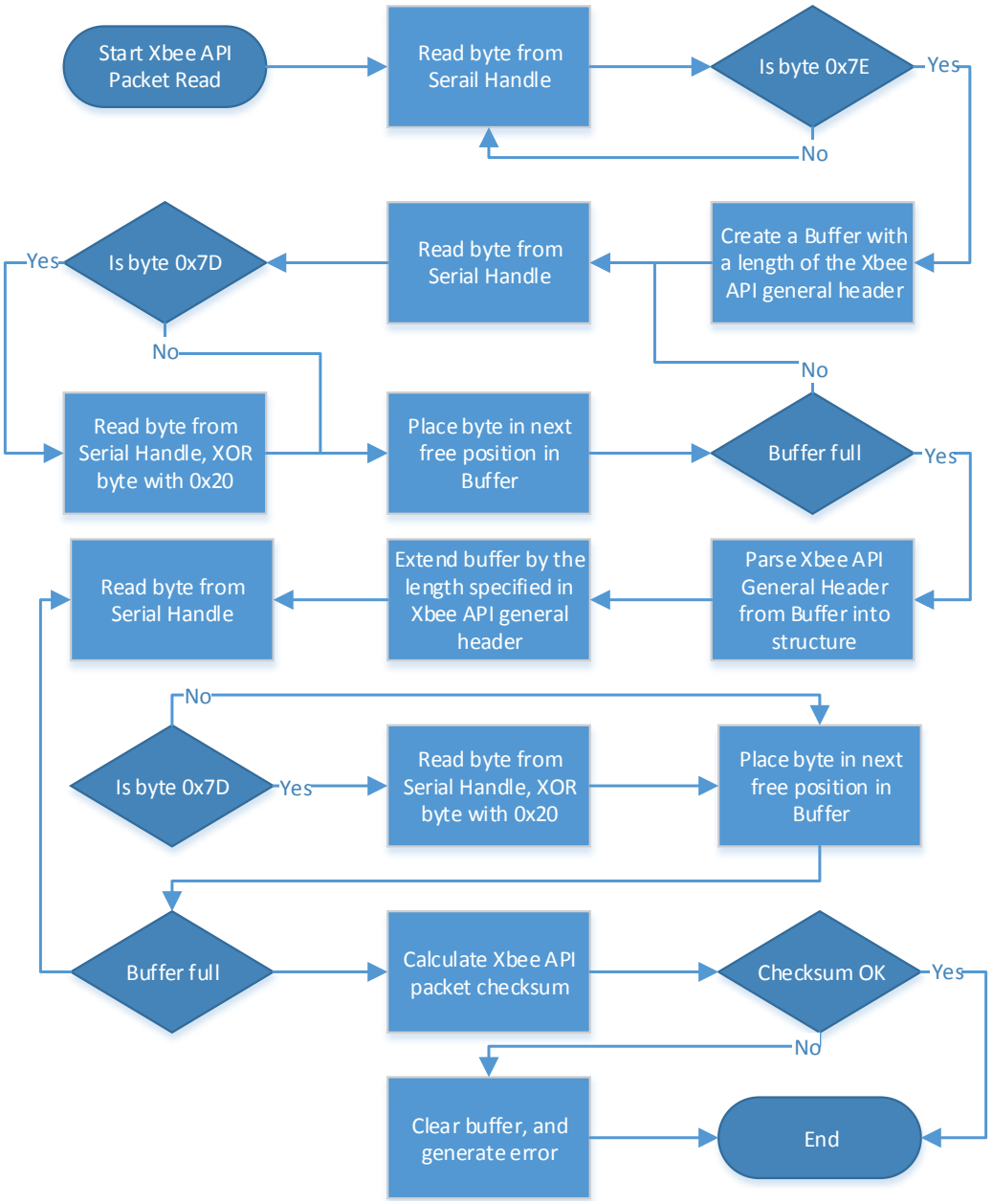

Figure 121 Flowchart for reading an XBee API packet form the serial file handle in the XBee IoT gateway software

The result of the reading process is a structure containing parameters from the general XBee header, and a buffer containing the rest of the API packet. The next step is to determine what type of packet is present in the buffer, and then parse it into an appropriate structure, which is shown in the flowchart in Figure 122.

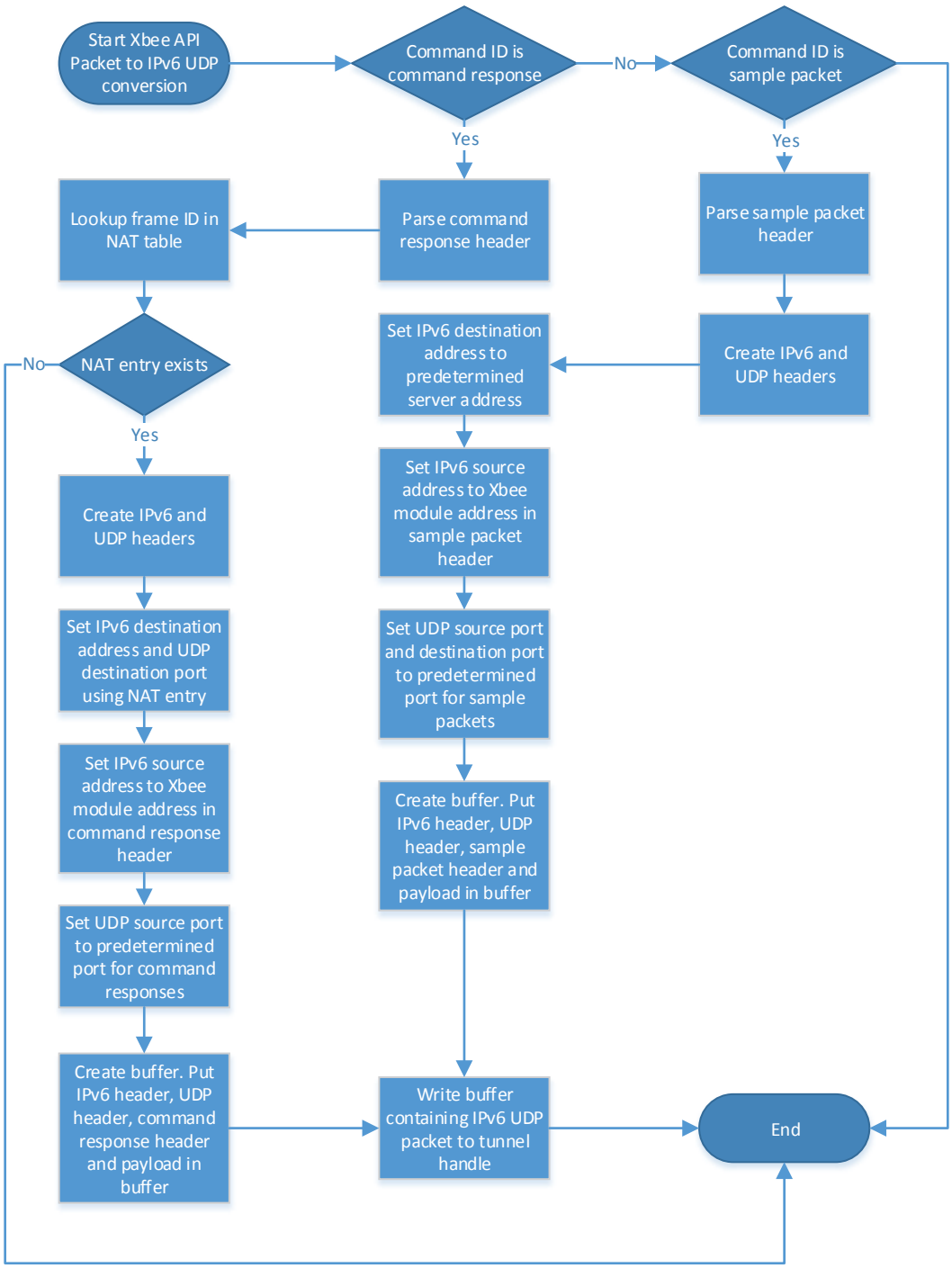

Figure 122 Flowchart for the process to parse different types of XBee API Packets

The details of the elements shown in the flowchart in Figure 122 are as following, they are in the order of execution in the software.

The first two decisions “Command ID is command response” and “Command ID is sample packet” determine the type of packet based upon the command ID in the general header. The following is the details of the process for the command response. If the command id is a sample packet then this process is skipped.

- “Parse command response header” uses the technique in section 5.3.2 to extract the header from the buffer into a structure. The command response header contains information about the XBee module that sent the response.
- “Lookup frame ID in NAT table” uses a table containing the IPv6 address and UDP port that command requests have originated from. The frame ID is used as a key to find the correct IPv6 address and UDP port in order to create an IPv6 UDP packet to contain the command response.
- “NAT entry exists” checks if there is an IPv6 address and UDP port to send the command response to. If there is none then the process ends.
- “Create IPv6 and UDP headers” creates the structures needed to contain the IPv6 header and UPD header.
- “Set IPv6 destination and UDP destination…” uses the NAT table to populate the IPv6 destination address and UDP destination port. The IPv6 source address is the IPv6 address of the XBee module that the response command originated from.
- “Set IPv6 source address…” uses the technique in section 5.4.2 to translate the XBee module address to an IPv6 address.
- “Set UDP source port…” uses a predetermined port number which is the same as the command request port in order for the response to go back to the application that sent the request.
- “Create buffer. Put IPv6 header, UDP header, command response header and payload in buffer” creates the raw IPv6 UDP packet containing the command response to be written to the tunnel file handle. The technique for generating the packet from the structures is given in section 5.3.2. Some of the command response header is not placed in the buffer such as the source XBee addresses as they are used for the IPv6 source address.

The second decision “Command ID is a sample packet” has the following process, and is similar to the above process.

- “Parse sample packet header” uses the technique in section 5.3.2 to extract the header from the buffer into a structure. The sample packet header contains information about the XBee module that sent the sample packet and the number and type of samples.

- “Set IPv6 destination address…” uses the IPv6 address of the server given to the software as a parameter at start-up.
- “Set UDP source port and destination port…” ensures that the UDP packet is delivered to the correct application for the given IPv6 address. The application running on the server to receive the samples will need to listen to this port in order to receive the UDP packets containing the samples.
- “Create buffer. Put IPv6 header, UDP header, sample packet header and payload in buffer” creates the raw IPv6 UDP packet containing sensor samples to be written to the tunnel file handle. The technique for generating the packet from the header structures is given in section 5.3.2. Parts of the sample packet header are not used as they contain redundant information, such as the XBee module address.

Both of these processes produce a buffer that contains a raw IPv6 UDP packet, “Write buffer…” then writes the buffer to the tunnel file handle. The tunnel driver can then send the IPv6 UDP packet to the required network interface in order to be sent to the required IPv6 destination.

The processes detailed in this section demonstrated the software required to allow communication from an internet server with an XBee WSN via a gateway. This allows the internet server to collect sensor information from the XBee WSN, and provides mechanisms to control the XBee WSN.

5.4.5 ZigBee IoT Platform Summary

The customised software presented in this section demonstrates a method of connecting an XBee WSN to and internet server with a gateway. IPv6 UDP packets are used to transport data to and from the WSN via the gateway. The gateway translates these IPv6 UDP packets to corresponding XBee API packets. To complete the IoT platform software on the server is required to communicate with the XBee WSN through the gateway. The implementation of the software on the server to collect data from the WSN, and control the WSN is given in section 5.6.

5.5 6LoWPAN IoT Platform Implementation

The implementation of 6LoWPAN involved the development of the firmware for the sensor nodes and the software for the gateway. The firmware development of the sensor nodes used hardware development kits coupled with an IDE. The development of the gateway software was similar to the development of the XBee gateway.

The firmware for the sensor nodes contains a 6LoWPAN stack, an IEEE 802.15.4 stack and a radio driver. The 6LoWPAN stack implements the 6LoWPAN communication by parsing 6LoWPAN packets or creating 6LoWPAN packets. In order to receive a 6LoWPAN packet to parse, or send a 6LoWPAN packet

the IEEE 802.15.4 stack was created, which manages the point to point communication between the nodes. The IEEE 802.15.4 stack then interfaces with the radio driver to transmit and receive 802.15.4 frames containing the 6LoWPAN packets.

5.5.1 6LoWPAN Development Hardware

The hardware used to develop the nodes and gateway used existing hardware development kits coupled with platforms to run embedded Linux. The sensor nodes used either the Texas Instruments EM430F6137RF90 development board, or the OLIMEX MSP430-CCRF development board. The gateway software was developed on the WRT54GL router.

5.5.1.1 6LoWPAN Node

A microcontroller coupled to a radio is needed to create a sensor node, several companies provide a single chip solution that contains a radio and microcontroller. The CC430 microcontroller range by Texas Instruments was chosen as the platform for the sensor nodes because the development environment is freely available and provides good debugging functionality. The CC430 microcontroller is an MSP430 microcontroller coupled with a C1101 radio [100] in the same package which reduces the number of components and board size.

The development boards used are shown in Figure 123, they use different variants of the CC430 microcontroller. The Texas Instruments EM430F6137RF900 [101] is a development board created by Texas Instruments for the CC430 microcontroller, it comes in a kit with two development boards and a JTAG debugger. The Olimex MSP430-CCRF [102] is a development board created by Olimex for the CC430 microcontroller and is sold individually without debugging hardware.

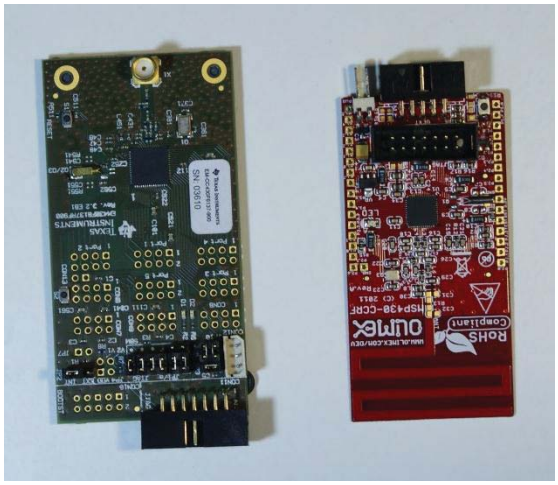

Figure 123 Texas instruments EM430F6137RF900 (left) and Olimex MSP430-CCRF (right)

The differences and similarities between the development boards are shown in Table 12.

Table 12 CC430 Development Boards Comparison

	TI EM430F6137RF90	Olimex MSP430-CCRF
Microcontroller	CC430F6137	CC430F5137
RAM	4 KB	4KB
Flash	32 KB	32 KB
LCD controller	Yes	No
Antenna	External (Omni directional)	PCB

In order to develop software for the 6LoWPAN modules a debugging interface and software environment was used. The software environment (Code Compose Studio) is described in section 5.1.1, Code Compose Studio requires the debugging interface to upload code and debug the development boards. The Texas Instruments evaluation development board was provided as a kit with a debugging interface which can be seen attached to the board in the bottom of Figure 2. The Olimex development board however is provided without a debugging interface. Texas instruments developed a starter development kit call the MSP430 Launch Pad [103] which can provide debugging capabilities to any MSP430 product that supports the Spy-Bi-Wire (SBW) protocol [104]. An adaptation board that connects the debug connecter to the SBW pins in the launch pad is shown in Figure 124, which allows Code Composer Studio to upload code and performing debugging.

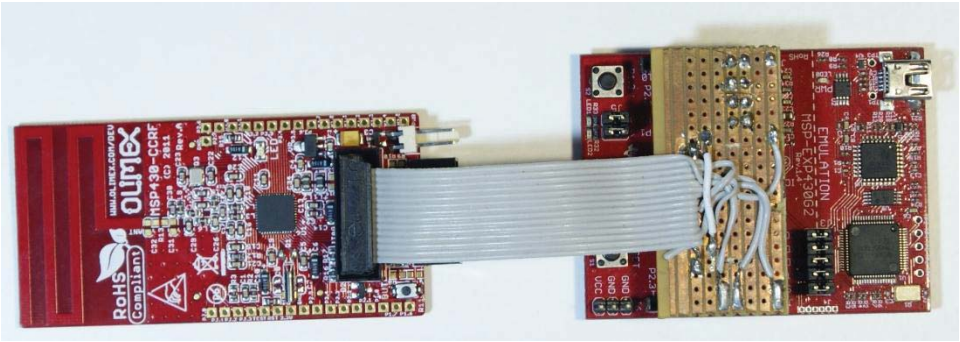

Figure 124 MSP430 Launchpad attached to the Olimex MSP430-CCRF for firmware upload and debugging

5.5.1.2 6LoWPAN Gateway

For the 6LoWPAN gateway the WRT54GL router and Raspberry PI were used. The gateway was primarily developed using the WRT54GL router, as the Raspberry PI was unavailable. The gateways required a radio interface in order to communicate with the 6LoWPAN nodes. A development board was used to provide the radio interface via a serial connection to the gateway.

Figure 125 shows the initial development system for the gateway, with the WRT54GL router attached via its serial port to the serial port of a Texas Instruments Development Board. The development board is also attached to a JTAG debugger to allow debugging with Code Compose Studio, see section 5.1.1 for details. The WRT54GL can be debugged with Eclipse via an Ethernet connection, see section 5.1.2.2 for details.

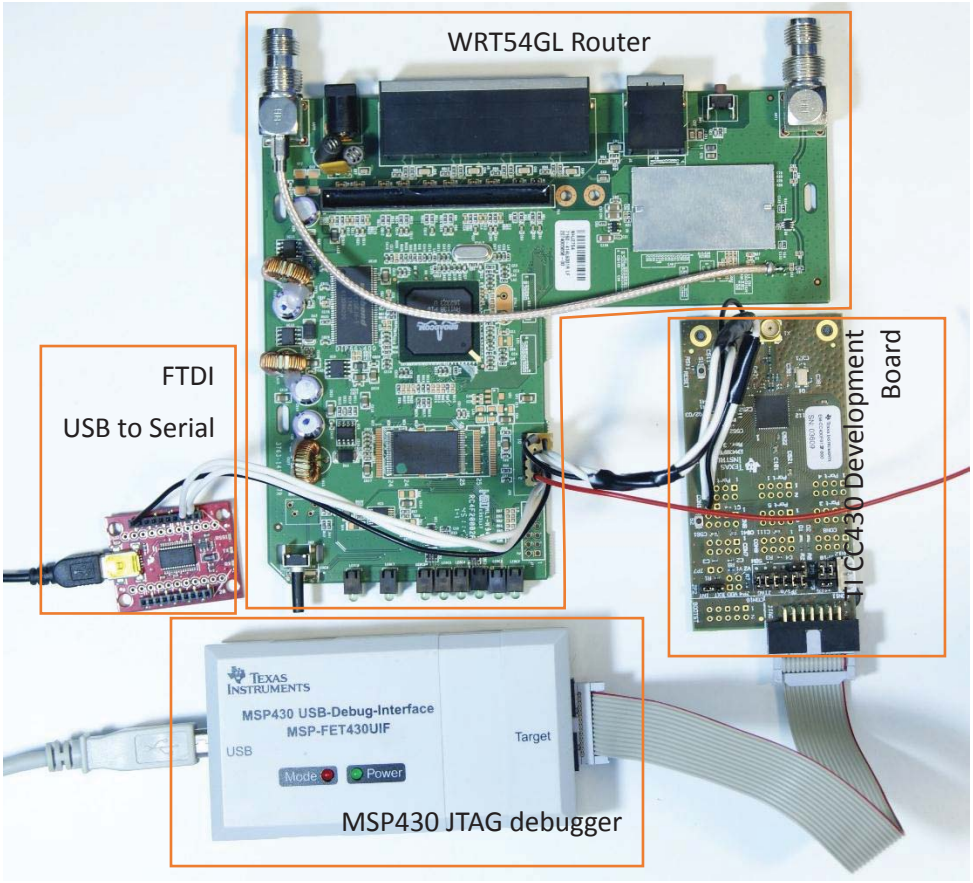

Figure 125 6LoWPAN gateway using a WRT54GL router connected to a Texas Instruments CC430F6137 development board with MSP430 JTAG debugger attached.

Figure 126 shows the Olimex MSP430-CCRF attached to the Raspberry PI, which provides power and serial communication. The Olimex board has a voltage regulator that can be supplied up to 12 V [102] which means it can be attached to the 5V supply pin of the Raspberry PI (refer to section 5.4.1.1). The serial port pins are provided on the UEXT header of the Olimex development board, which are connected via the cable seen in Figure 126 to the Raspberry PI header to create the serial connection.

Figure 126 6LoWPAN gateway using the Raspberry PI connected to an Olimex CC-RF development board

5.5.2 6LoWPAN Node Firmware Implementation

The firmware on the 6LoWPAN node has three components, which are the 6LoWPAN stack, the radio interface, and the application to quire and send sensor data using the 6LoWPAN stack. The 6LoWPAN stack performs the parsing of packets received by the radio interface, and can generate me packets to be sent via the radio interface. The radio interface performs the necessary setup and functions to operate the internal CC1101 radio in order to send or receive packets. The application reads analogue voltages using the ADC, and packages them into a UDP packet to be transmitted to the IoT server.

5.5.2.1 6LoWPAN stack

The 6LoWPAN stack incorporates IEEE 802.15.4 for the lower level communication between nodes. A lower level Media Access layer (MAC) and physical layer are required by the 6LoWPAN stack to operate [105]. The IEEE 802.15.4 stack performs basic network joining functions, and facilitates the transmission and reception of data packets [57].

In order for the 6LoWPAN node to send a UDP packet to the server a LoWPAN header, and compressed UDP header must be created. Table 13 shows the structure of these headers.

Table 13 6LoWPAN header for sending UDP packet to server

<table>
<tr><th>Section</th><th>Bytes\Bits</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th></tr>
<tr><td rowspan="2">LoWPAN IPHC</td><td>0</td><td>0</td><td>1</td><td>1</td><td colspan="2">Traffic & Flow</td><td>Nxt Head</td><td colspan="2">Hop Limit</td></tr>
<tr><td>1</td><td>Context</td><td>Src Adr C</td><td colspan="2">Src Adr Mode</td><td>Mlti Cst</td><td>Dst Adr C</td><td colspan="2">Dst Adr Mode</td></tr>
<tr><td rowspan="16">IPv6 Dst Adr</td><td>2</td><td colspan="8" rowspan="16">Server IPv6 Address</td></tr>
<tr><td>3</td></tr>
<tr><td>4</td></tr>
<tr><td>5</td></tr>
<tr><td>6</td></tr>
<tr><td>7</td></tr>
<tr><td>8</td></tr>
<tr><td>9</td></tr>
<tr><td>10</td></tr>
<tr><td>11</td></tr>
<tr><td>12</td></tr>
<tr><td>13</td></tr>
<tr><td>14</td></tr>
<tr><td>15</td></tr>
<tr><td>16</td></tr>
<tr><td>17</td></tr>
<tr><td rowspan="3">UDP</td><td>18</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>Chk Sum</td><td colspan="2">Ports</td></tr>
<tr><td>19</td><td colspan="5">Source Port</td><td colspan="4">Destination Port</td></tr>
<tr><td>N</td><td colspan="9">UDP payload</td></tr>
</table>

The 6LoWPAN header compression standard [105] defines the technique to create the LoWPAN compressed header. This LoWPAN compression header specifies which address information in an IPv6 header can be omitted and obtained from the IEEE 802.15.4 layer or a context. A context is a number assigned to an address which is shared between the node and the router, however the technique of sharing this address and its' context is in the draft stages [106]. The servers IPv6 address cannot be obtained from the IEEE 802.15.4 information and is therefore included after the LoWPAN header. Any information that cannot be compressed by 6LoWPAN appears in the order that it appears in an IPv6 header, after the LoWPAN compression header. The 6LoWPAN header compression uses information from the IEEE 802.15.4 header to reconstruct the IPv6 address of the 6LoWPAN nodes.

The compression of the UDP header is also defined in [105], which operates in a similar way to the compression of IPv6 information. A UDP compression header defines which fields have been compressed, and is then followed by the compressed or uncompressed fields as they appear in an uncompressed UDP header. After the uncompressed fields is the payload of the UDP packet.

The format of the header that precedes the 6LoWPAN header is given in Table 14.

Table 14 IEEE 802.15.4 header for 6LoWPAN UDP packet

Section	Bytes\Bits	1	2	3	4	5	6	7	8
Frame Control	0	Frame Type			Security	Frm Pend	AR	Pan ID	0
	1	0	0	Dst Adr Mode		0	1	Src Adr Mode	
Seq	2	Sequence Number							
Addressing	3	Destination PAN							
	4								
	5	Destination Address							
	6								
	7	Source Address							
	8								
6LowPAN	N	6LoWPAN header, Compressed UDP header, and UDP payload							

5.5.2.2 Microcontroller Radio Interface for 6LoWPAN stack

The IEEE 802.14.5 component of the 6LoWPAN stack requires access to a radio for transmission and reception of packets. Functions to initialise the radio, transmit a packet using the radio, and wait for a packet to be received from the radio are provided by the radio interface to the IEEE 802.15.4 component of the 6LoWPAN stack.

The interface between the CC1101 radio and the MSP430 microcontroller is provided through registers on the MSP430 which define logical channels [107]. The logical channels provide access to the CC1101 registers and commands. Feedback from the CC1101 is also obtained via and interrupt, which is used to indicate certain events have occurred such as receiving a preamble, or the transmit buffer or receive buffer is below a threshold.

The initialisation of the CC1101 radio involves setting registers that control the following.

- Carrier frequency - this is the base frequency for the first channel that the radio will operate at. The operating frequency of 915 MHz was chosen.
- Channel spacing – the frequency spacing between adjacent channels.
- Channel number – the channel number determines which carrier frequency the radio will operate at and is the channel number multiplied by the channel spacing added to the Operating Frequency. The channel number can be set using a function provided by the radio interface. Initially the channel is set to channel 0.
- Data rate – this is the rate at which information will be sent. 38 kilo-bits per second was chosen as the data rate.

- Modulation type – this is the type of modulation to use, which can be Gaussian Frequency Shift Keying (GFSK) or Frequency Shift Keying (FSK) or Amplitude Shift Keying (ASK) or On/Off Keying (OOK). FSK was the modulation type chosen.

The values for the registers can be obtained using a tool provided by Texas Instruments called "SmartRF Studio", which can be obtained here [108]. It allows easy configuration by specifying the parameters above which generates a list of registers and values. This list is then put into an array and iterated through programmatically by the radio interface to setup the CC1101 radio.

Transmitting or receiving data using the CC1101 radio is achieved using buffers. There are individual buffers for transmitting and receiving, both are 64 bytes in size. The buffers can be read and written to while transmitting or receiving in order to accommodate packets larger than 64 bytes. This is achieved using interrupts that occur when the amount of data in the buffers are below a certain threshold so that appropriate action can be taken.

The process in Figure 127 is used to transmit a buffer containing a packet.

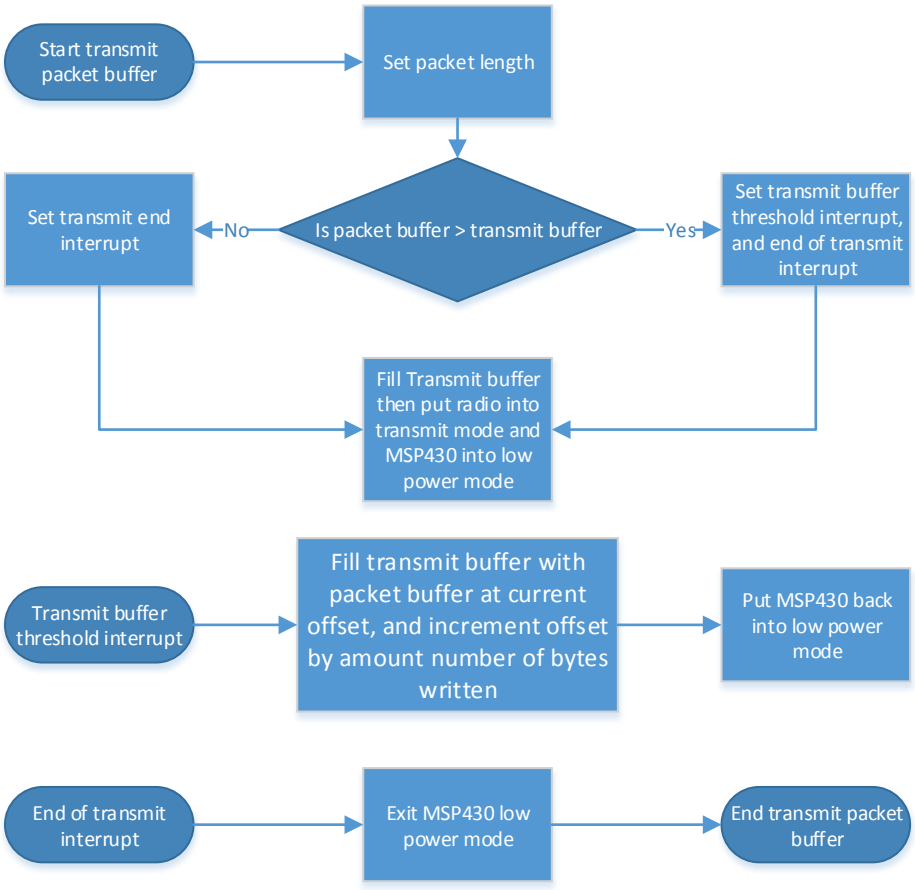

Figure 127 Flow chart for process of transmitting data using CC1101 radio

The first part of the process determines which interrupts should be used based on the length of the packet buffer. Several Interrupts can be generated by the radio, so if the packet buffer's length is less

than the transmit buffer the end transmit interrupt is used, otherwise both the end transmit interrupt, and transmit buffer threshold interrupt is used. After the appropriate interrupts are chosen the transmit buffer is filled with the packet buffer and the length of the packet buffer is set as the packet length register, the radio is put in transmit mode, and the MSP430 is put to sleep. In the sleep mode the MSP430 can be woken up by interrupts caused by the CC1101 radio. The CC1101 knows when a packet has been transmitted as a register containing the length of the data to be transmitted is set before putting the radio in transmit mode.

When the transmit buffer threshold interrupt occurs it means that the transmit buffer is nearly empty and must be refilled. After refilling the transmit buffer with the packet buffer, the MSP430 re-enters the low power sleep mode.

When the end of transmit interrupt occurs the CC1101 radio has completed sending the packet buffer and therefore the MSP430 is taken out of low power mode to continue with any other required operations.

The process to receive a packet is similar to transmitting a packet, however the start of receiving a packet needs to be detected with an interrupt. The flowchart in Figure 128 shows this process.

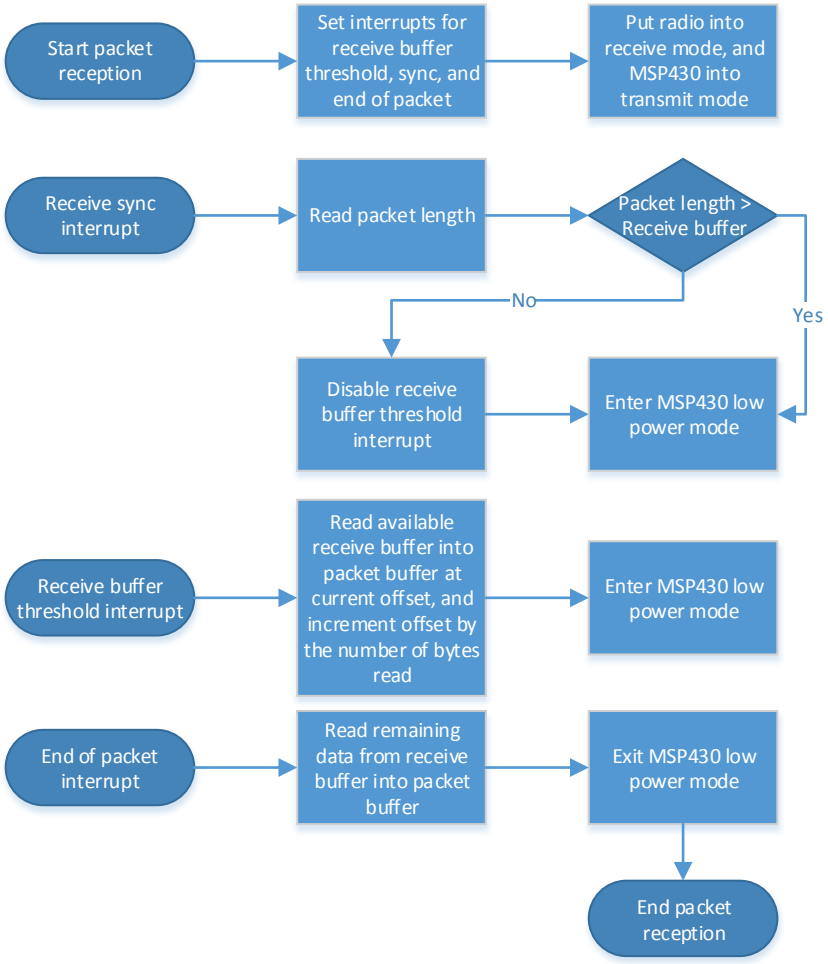

Figure 128 Flowchart for process of receiving a packet using the CC1101 radio

As with the transmitting process, the receiving process starts with enabling interrupts. These interrupts are as following.

- Receive sync interrupt occurs when the preamble and packet length have been received.
- Receive buffer threshold interrupt occurs when the receive buffer has been filled past a predetermined threshold.
- End of packet interrupt occurs when the packet has been received.

The receive sync interrupt is used to determine if the receive buffer threshold interrupt should be enabled based on the length of the packet being received. After the interrupt occurs the MSP430 is put back into sleep mode, to wait for either the end of packet interrupt or receive buffer threshold interrupt.

The receive buffer threshold interrupt is used to empty the read buffer into the packet buffer. The MSP430 is put into sleep mode once the buffer is emptied in order to wait for either another receive buffer threshold interrupt, or the end of packet interrupt.

The end of packet interrupt means the packet reception is complete and any remaining data must be flushed from the receive buffer into the packet buffer. After the interrupt is complete the MSP430 exits the low power mode to continue with processing the packet.

5.5.2.3 Application to acquire and send sensor data using 6LoWPAN

A simple application to acquire sensor data and send it to the server using 6LoWPAN was developed to mimic the behaviour of the XBee based IoT platform. The sensor data is in a format similar to the format of a sensor sample produced by the XBee Gateway implementation. This means that when the server receives a UDP packet containing a data sample it can be interpreted using the same technique as a sample produced by and XBee module. The 6LoWPAN gateway will produce UDP packets containing sensor samples which have an identical format to those produced by the XBee gateway.

5.5.3 6LoWPAN Gateway Implementation

The gateway implementation comprises of firmware for a CC430 development module and software for the gateway (the WRT54GL or Raspberry PI). The firmware for the CC430 development module provides an interface to configure the radio, transmit packets using the radio, and listen for packets from the radio. The software for the gateway uses a serial interface to communicate with the 6LoWPAN network, a tunnel interface to communicate with the IPv6 network, and performs translation between the two networks.

5.5.3.1 Radio interface for 6LoWPAN gateway

The radio interface uses the same code as the radio driver for the CC430 development board in section 5.5.2.2. The radio driver provides transmit and receive functions which are accessed by commands sent from the gateways' serial port to the CC430 development boards' serial port.

The transmit command payload contains the packet to be transmitted by the CC1101 radio. When the transmit command is received from the serial port of the CC430 developer board, the radio driver initiates the transmit function using the transmit commands' payload as the packet buffer. After the packet is transmitted a confirmation message is sent out the serial port back to the gateway to indicate that the packet has been sent.

When the receive packet command is received from the gateway, the CC430 board uses the receive function of the radio driver to wait for a packet to be received. Once a packet is received it is transmitted via the serial port to the gateway.

5.5.3.2 Software for 6LoWPAN Gateway

The software running on the gateway performed the task of translation between IPv6 and 6LoWPAN. The software was developed using the environment described in section 5.1.2. 6LoWPAN packets received by the radio interface which are destined for networks other than the 6LoWPAN network are decompressed into IPv6 packets by the software running on the gateway. The technique to decompress the 6LoWPAN packets if defined in [105]. Once they are decompressed they are sent using the tunnel interface technique described in section 5.3.1. IPv6 packets received by the tunnel interface which are destined for the 6LoWPAN network are compressed into 6LoWPAN packets and sent using the radio interface. The technique to compress IPv6 to 6LoWPAN is defined in [105].

5.6 IoT Server Implementation

A customised software service running on a server acquires data from the WSNs and stores it in the database. A website then provides access to the stored sensor data in the form of graphs. The sensor data from WSN is sent to the server via a gateway, and the gateway is connected via the internet to the server using a secure tunnel (OpenVPN). A service running on the server collects the received sensor data and stores it in a database. The sensor data can be accessed, and plotted in graphs from a website.

5.6.1 Sensor Data Acquisition and Storage

The software service running on the server performs two tasks, which are acquiring the sensor data and storing the sensor data. The first task uses a UDP listening socket to receive the UDP packets containing sensor data created by the gateway. Once a UDP packet is received, the sample data is extracted from the payload, and the source address of the XBee Module that sent the data is extracted from the IPv6 source address of the UDP packet. The second task involves using the information extracted from a UDP packet to create a database entries for each sensor sample extracted from the UDP packet.

5.6.1.1 Receiving Sensor Samples

In many operating systems the application interface to perform basic network communication is called a socket. There are two modes of operation for a socket, client and server. In client mode the socket is used to connect to a server to exchange information with the server. In server mode the socket listens for incoming connections from clients, once a connection is made a new socket for that client is created in order for data to be exchanged with the client.

The IoT service used a socket in server mode to listen for incoming UDP packets from clients (XBee modules). However, as UDP is connectionless there is no connection stage when a socket is in the

server mode, therefore a single socket must handle data coming in from multiple clients. When data is received from the socket, the address of the source of the data is also present.

5.6.1.2 Storing Sensor Samples

When taken individually sensor data is meaningless, in order to give the sensor data meaning it must have a context. The context of the sensor data is the source of the sensor data which is the sensing modules address (which includes the channel the sensor data was obtained from), and the time the sensor data was received. Therefore, the structure of database to store acquired sensor data must be well defined in order for the sensor data stored in it to have relevant meaning.

The sensor data decoded from the UDP packet is stored in a MySQL database so that it can be accessed from the website. The sensor data must be stored with the source address and channel of the sensor data as well as the time the data was received in order to maintain the context of the sensor data. This means the following columns in the database table are required.

- Sensor ID – the IPv6 address of the sensor module that sent the sample data.
- Sensor Channel – the channel that the sample was taken from.
- Date and time – the date and time the UDP packet arrived containing the sample information.
- Sample Data – the raw sample data.

5.6.2 Graphical Web Interface

The purpose of the graphical web interface is to display the sensor data from each sensor in a graph on a web page. In order to do this sensor data from a specific channel and device is recalled from the database and sent to the web browser. The web browser converts the raw sample data into the unit that the sensor measuring then displays it in a graph. Screenshots of the graphing website can be found in the results, section 6.1.

Additionally simple control of the digital output pins was also implemented to demonstrate remote control of the sensor modules. A simple webpage with a button that toggles the state of the digital output pins was created to control an electrical device.

5.6.2.1 Retrieving Sensor Samples from Database

In order to retrieve the sensor samples from the database the sensor id, sensor channel, and data and time range are required. These parameters are used to create a query to the database to retrieve the required sensor samples. Conversion of the sensor samples is also required meaning the conversion function for the sensor data must also be retrieved.

A sensor data retrieval PHP script was written to obtain sensor data for a given time period from a given sensor and channel. The PHP script creates a SQL query with the sensor identifier, sensor channel, and date range to return a set of results with the required sensor data. The sensor data is then formatted into a Java Script Object Notation (JSON) array with the sensor sample and corresponding time the sample was taken. The JSON array is then output.

A sensor information retrieval script was written to obtain sensor information for a given sensor. The PHP script formats a SQL query with the sensor identifier and sensor channel in order to return the sensor information, such as the sensors name, and conversion formulae. The result is formatted into a JSON object, which is the output.

5.6.2.2 Plotting Retrieved Sensor Samples in Graph

The output of the sensor sample retrieval is placed into a JSON array, which is transmitted to the web browser. In order to convert the sensor samples to units a conversion function is needed, which is also retrieved from the database. The web browser uses the conversion function on the sensor samples to produce points to plot in a graph.

The sensor data retrieval PHP script provided access to the samples from a given sensor, for a given time period. The output of the script is an array with the sample and time the sample was taken. A JavaScript script running on the web browser queries the PHP script for sensor data. The sensor information retrieval script provided access to the conversion function, which is queried by the JavaScript script to obtain the formulae. The conversion of the sensor data to standard units that represent the measurement of the sensor is performed by the JavaScript script running in the browser. After the sensor data has been converted it is plotted in a graph using a graphing library called HighCharts [109].

5.6.2.3 Controlling WSN Nodes

Control of the sensor modules digital output is given via a webpage. A PHP script on is called by webpage, which creates an IPv6 UDP packet with a control message to send to the XBee module.

5.7 Summary

Section 5.1 provides the details of the software tools used to develop firmware and software required for the WSN-IoT platform. The firmware software tools were required to develop the 6LoWPAN nodes. The software required for the gateway needed a custom toolset for cross compiling.

Section 5.2 provides the details of the software required to support he WSN-IoT platform. Connection between the gateway and server was provided by a secure tunnel software which is open source. The website and database running on the server uses open source software to provide these services.

Sections 5.3 provides the details of the implementation technique used to achieve networking connectivity on the gateway, and techniques used for parsing information in network packets.

Section 5.4 provides the details of the ZigBee based IoT platform implementation. The ZigBee platform consists of a ZigBee based WSN and a gateway to interconnect the ZigBee based WSN with a server. Details of various sensors fabricated for the system are provided. The implementation of the software required for the gateway to interconnect the ZigBee WSN with the server is given.

Section 5.5 provides the details of the 6LoWPAN based IoT platform implementation. The 6LoWPAN platform consists of 6LoWPAN nodes, and a 6LoWPAN gateway. Details of the 6LoWPAN sensor nodes, and the required firmware implementation are given. The software implementation to translate between 6LoWPAN and IPv6 for the gateway is also given.

Section 5.6 provides the details of the IoT Server platform implementation. The IoT server platform consists of an IoT service to receive sensor information, a database to store sensor information, and a website to display sensor information. The IoT service can receive data samples from the 6LoWPAN gateway, and the ZigBee gateway.

6 Experimental Results

The following sections give results related to the ZigBee based WSN-IoT platform. The 6LoWPAN system was tested, however it was not stable enough to run for prolonged periods to produce satisfactory results.

6.1 Fabricated Sensor Modules for monitoring the Smart Home

The developed system was tested by installing smart sensing units and setting up a ZigBee based WSN connected to the gateway at a household. Interconnecting ZigBee network with IPv6 network is performed by connecting and configuring the gateway. The Integrated system was continuously used and generated real-time graphical representation of the sensing information on request.

Figure 129, Figure 130, and Figure 131 show the graphical representation of different types sensing unit's information in real-time on the website. Measurements related to the hot water system described in section 5.4.1.3 are shown in Figure 129. Measurements from a light intensity sensor, and temperature sensor are shown in Figure 130. Measurements from an electrical sensing unit attached to the electric water supply pump are shown in Figure 131.

Figure 129 shows temperature measurements taken at the hot water cylinder and the solar heater, and the power consumption of the hot water cylinder. From the graphs it is easy to determine when hot water was being used and the reduction in power consumption when the solar heater is active.

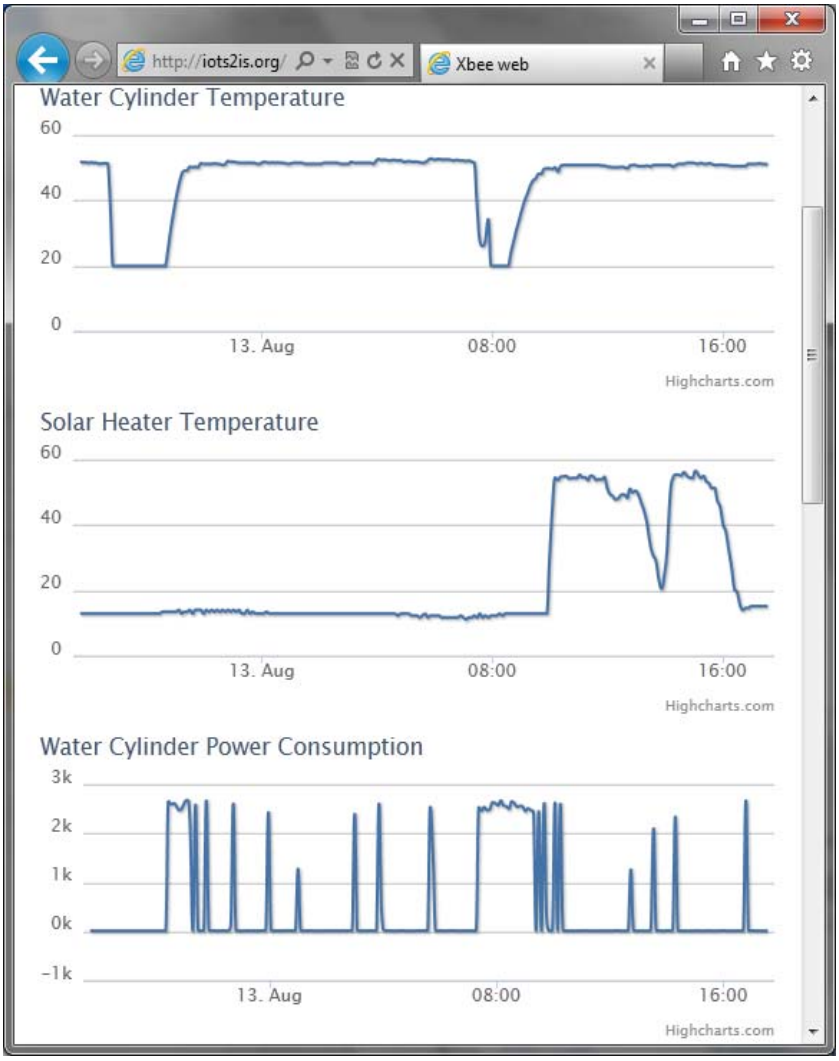

Figure 129 Real-time Graph of sensor information for the solar water heater, and hot water cylinder

Figure 130 shows temperature measurements taken outside the home and the approximate light intensity that the solar heater is subjected to. When compared with Figure 129 it can be seen that when the light intensity and temperature outside the home is above a threshold the solar heater becomes active.

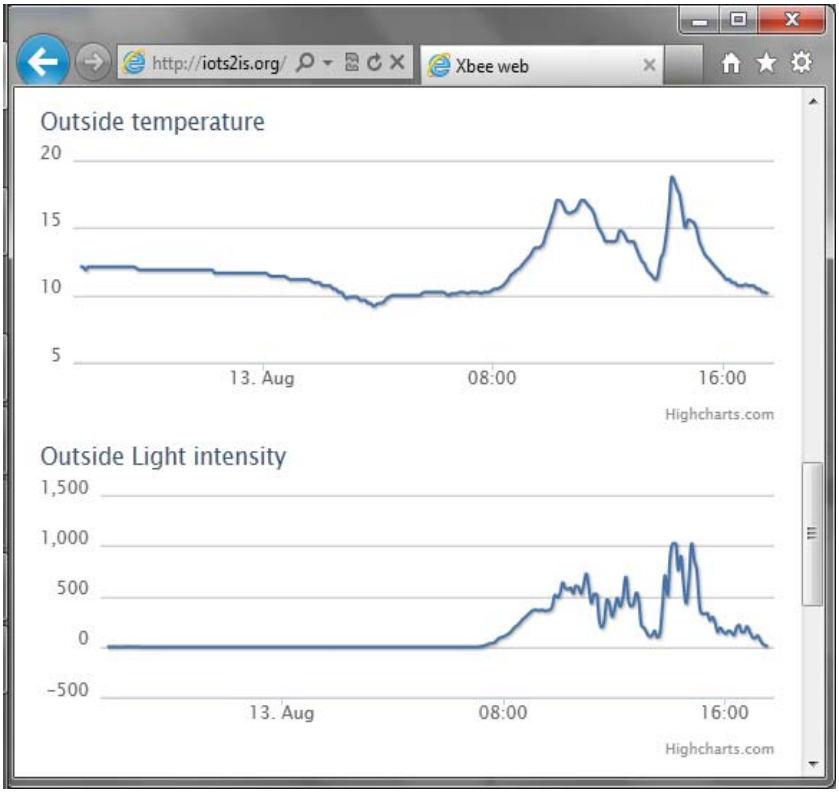

Figure 130 Real-time graph of sensor information on website for light intensity and ambient temperature

Figure 131 shows measurements of the voltage and current at the water supply pump. The voltage measurement has large amount of noise, and is due to several factors such as ungrounded inputs, insufficient filtering and power supply fluctuations.

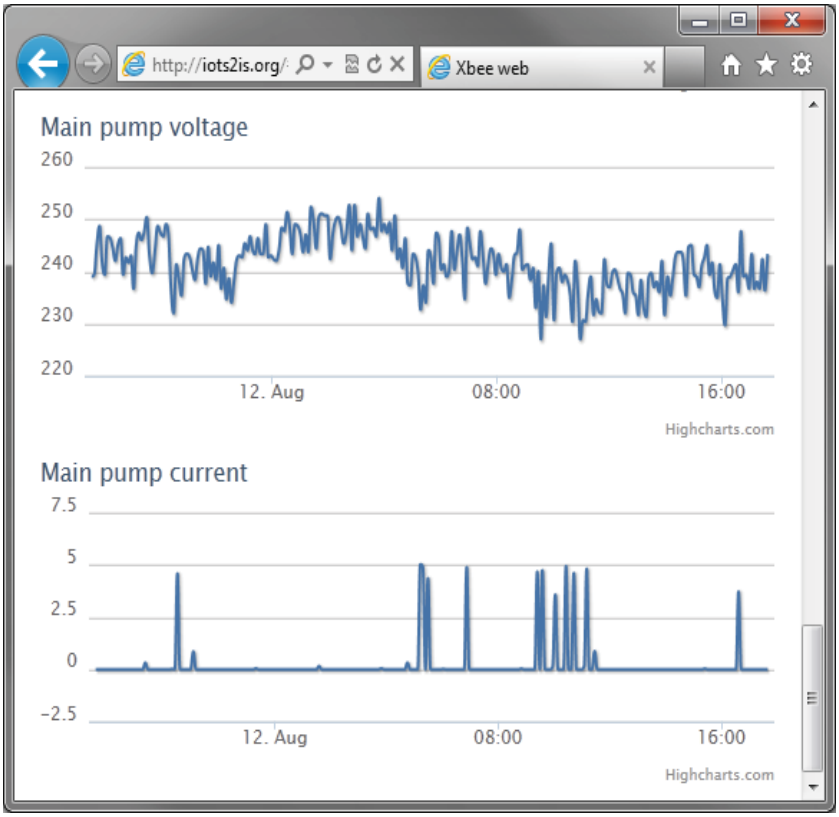

Figure 131 Real-time graph of sensor information on website for voltage and current usage of electric pump

Figure 132 shows measurement of three same type of temperature sensor fabricated on a single board to study the variation and degradation of the temperature sensors. It was clearly observed that the temperature sensors are not shown any variation from the mean in the collected readings.

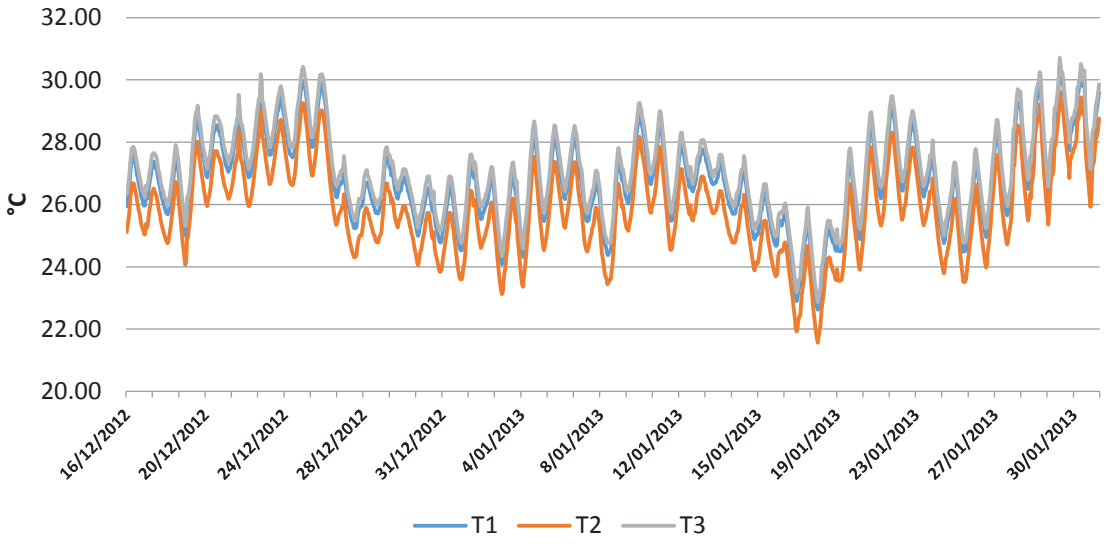

Figure 132 Graph of temperature data from three temperature sensors for 45 days

Figure 133 depicts the correlation of typical household usage scenario of an inhabitant. The parameters measured are: the temperatures of the hot water cylinder and solar heater, the power

consumption of the hot water cylinder and the light intensity outside the home. These parameters allow for analysis of the water heating system.

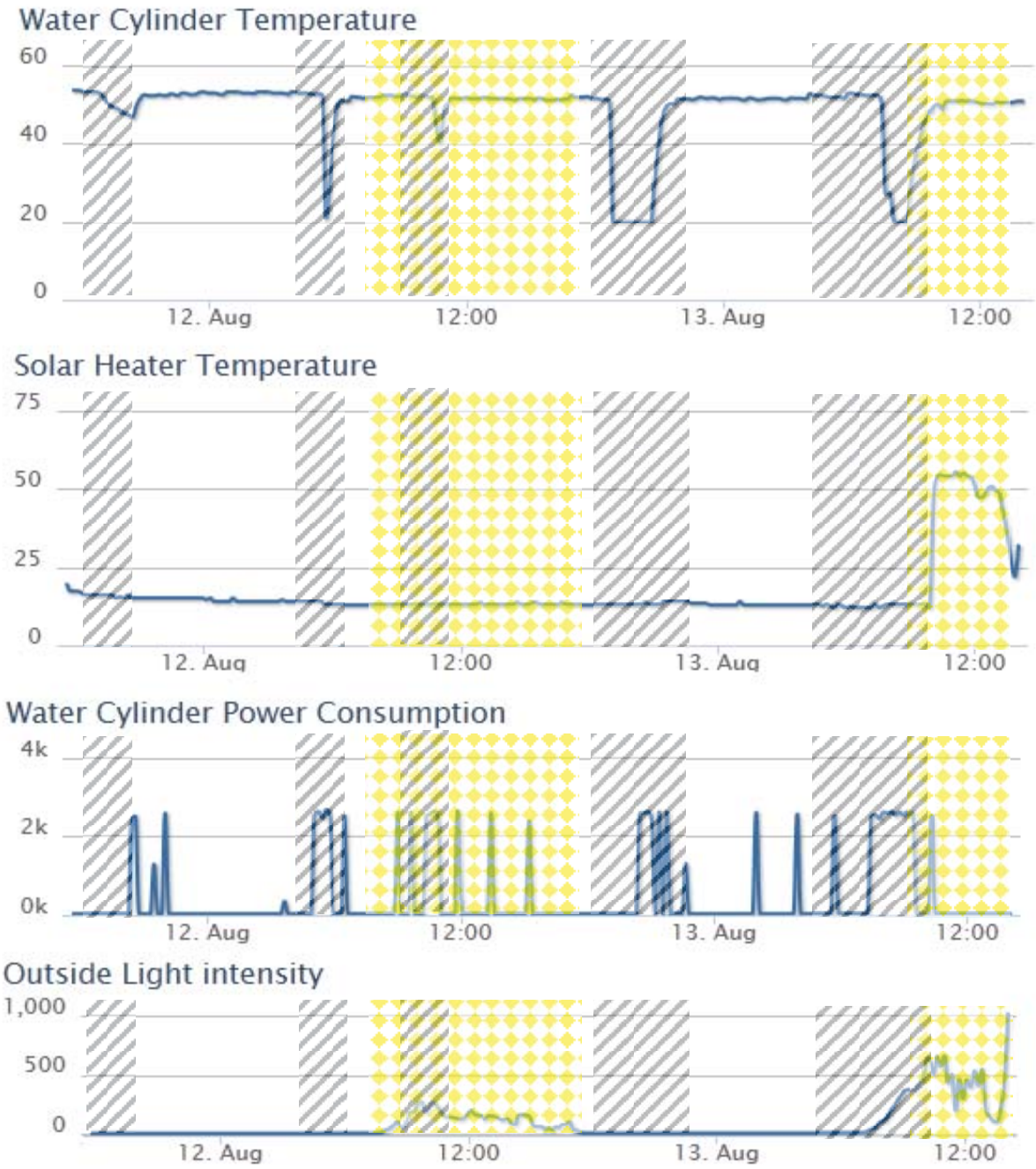

Figure 133 Correlation of sensor information depicting two days typical usage

The times when hot water was used can be clearly identified when the water cylinder temperature decreases, subsequent increases in power consumption by the water cylinder can be seen. These times are highlighted in blue. The light intensity measurements give an indication of when there is an adequate amount of sunlight to heat the water in the solar heater. The first yellow section shows that light intensity was too low to heat the water, as this was an overcast day. The second yellow section

shows that the light intensity was significantly higher and therefore the solar heater heated the water. The temperature of water in the solar heater does not exceed the water cylinder temperature due to a circulation pump providing colder water.

Figure 134 shows the ambient temperature of a room over a continuous period of 45 days. It is observed that the day light temperatures are higher and night time temperatures are lower when compared with the average temperature of a day. This is a real-time graphical depiction of ambient readings of a room in a smart home.

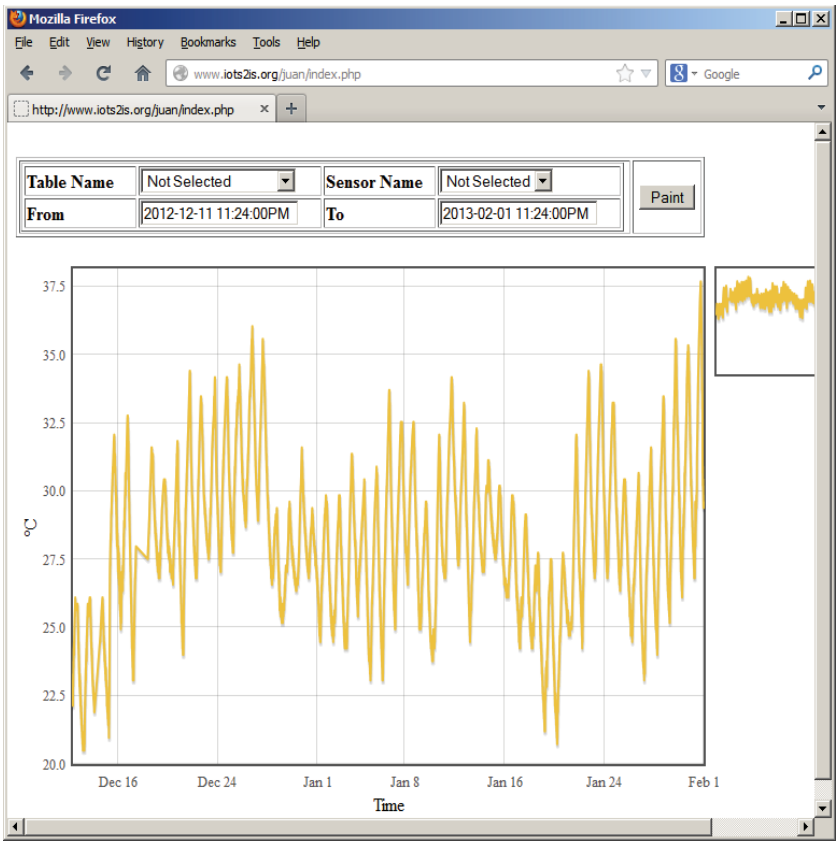

Figure 134 Real-time graph of temperature data from a temperature sensor for 45 days

6.2 Efficient mechanism for sensor data storage

Compression of the real-time data storage for the temperature sensor recordings is done by considering the variations on the successive temperature readings. If there is a change in the consecutive temperature readings then the new reading is stored thereby reducing the amount of sensor recordings to be stored. This technique of variable sensor data recordings has improved the performance for plotting the real-time graphs on the web significantly. This is mainly because the client browser is receiving fewer points of data to be plotted. Table 15 shows the amount of compression achieved for each sensor devices, which was an average of 78% less stored data.

Table 15 Comparison of compressed and uncompressed data for temperature sensors

Sensing Unit	Channel #	Sample Packets received	Samples Stored	Compression (%)
Module 89	T1	74836	11277	84.93
	T2	74836	10576	85.87
	T3	74836	8555	88.57
Module 50	T1	74866	19262	74.27
	T2	74866	16377	78.12
	T3	74866	15552	79.23
Module 94	T1	74932	21166	71.75
	T2	74932	22032	70.6
	T3	74932	20568	72.55
Average				78.43

6.3 Quality of Service parameters for XBee IoT Platform

The data from two sensors of hot water system for the duration of a month was analysed to determine the reliability, throughput and jitter of the system. The Xbee- S2 devices were configured to send samples every 10 seconds. The arrival time of these samples on the server are recorded in the database accordingly. The time between these recorded times is the interval at which the Xbee device is sending sample information. The total amount of sample information received by the server was calculated by dividing running time of the system with the sample interval.

6.3.1 Reliability

The reliability of the system was determined by comparing the calculated value with the amount of sensor information received correctly. The difference between arrival times of successive sensor information gives the interval value. If the time interval is greater or less than 10 seconds then there was an error. When the interval is less than 10 seconds then the sample information received was incorrect or duplicated and therefore it is erroneous. When the interval is greater than 10secs sample information has been lost, the amount of information lost can be determined by dividing the greater interval by the expected interval, Table 16 shows the reliability of two devices sensor information transmission of ZigBee data encapsulated in IPv6 packet.

Table 16 Reliability of the data transmission in the integrated ZigBee –IPv6 networks of two sensing units for a period 31 days

Sensing module	Average time between correctly received samples (seconds)	Number of lost packets	Expected number of packets	Number of correctly received packets	Reliability (%)
1	9.80	7502	272908	265406	97.25
2	10.55	7354	253774	246420	97.10

6.3.2 Throughput

The throughput of the sensing module is the amount of data sent from the sensing module to the server in a given time period [20]. The amount of data in each sample packet was 16 bytes, which is sent every 10 seconds. Therefore the throughput of the sensing module was 1.6 bytes per second. To measure the throughput of the sensing module, the number of packets received in a 5 minute time span was considered. The throughput was obtained by dividing the time interval of 5 minutes. Figure 136 shows the throughput of a sensing unit for the period of one month.

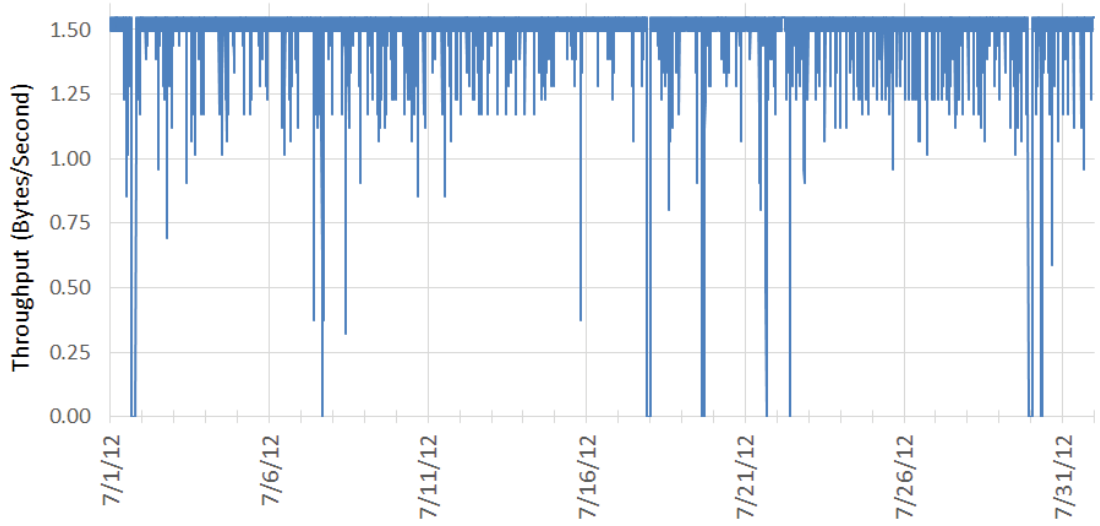

Figure 135 Throughput of an electrical sensing unit for a period of one month

The average throughput is 1.55 bytes/second since the reliability was 97%. Investigation into the sporadic significant change in throughput is in process. Causes are likely to be the interferences from other networks such as WiFi.

The throughput for **temperature sensor units** was measured by considering the number of packets received in an hour. Figure 136 Throughput of temperature sensor units shows the throughput of the temperature sensors for a period of 45 days. The average throughput was 61.3 packets per hour this is

because the XBee module timers are inaccurate, as they were set to 1 minute interval. The reason for throughput above average is either due to: XBee module resetting and resending or the application gateway duplicating packets. The reason for throughput below the average is either due to: XBee module failing to send packet or application gateway failing to send packet.

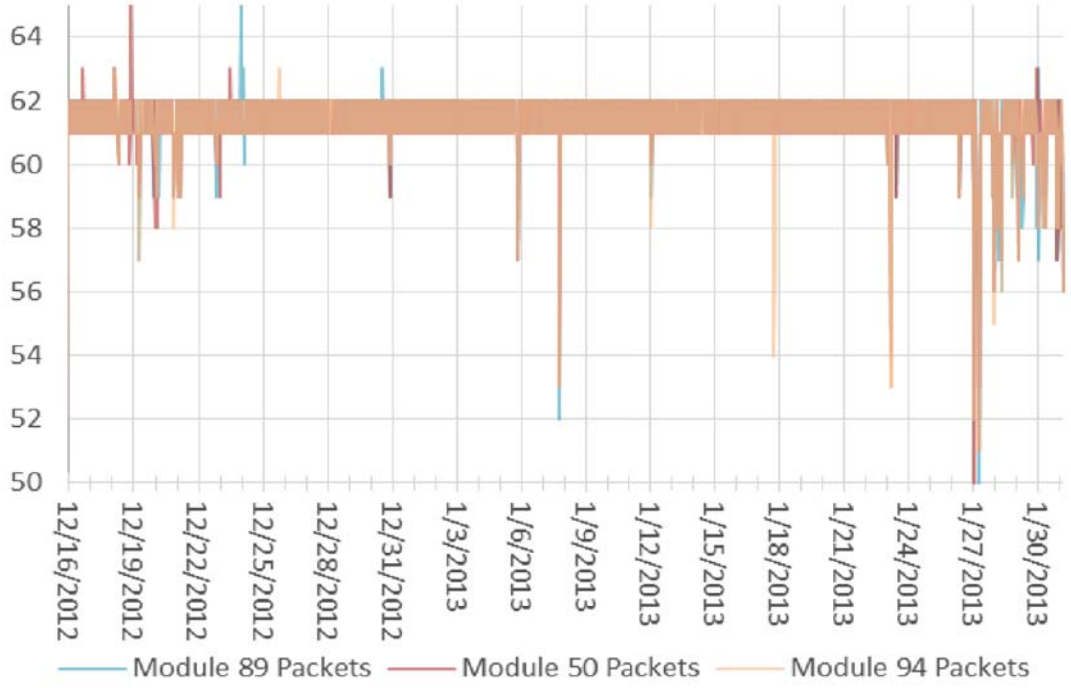

Figure 136 Throughput of temperature sensor units

6.3.3 Jitter

The jitter was measured as the delay between two consecutive packets following the technique as mentioned in [110]. Figure 137 shows the jitter of sensing data for a period of one month. The instances of large jitter relate to the correlated low throughput. The other QoS parameters such as delay, energy consumption measurements are in the trial stage.

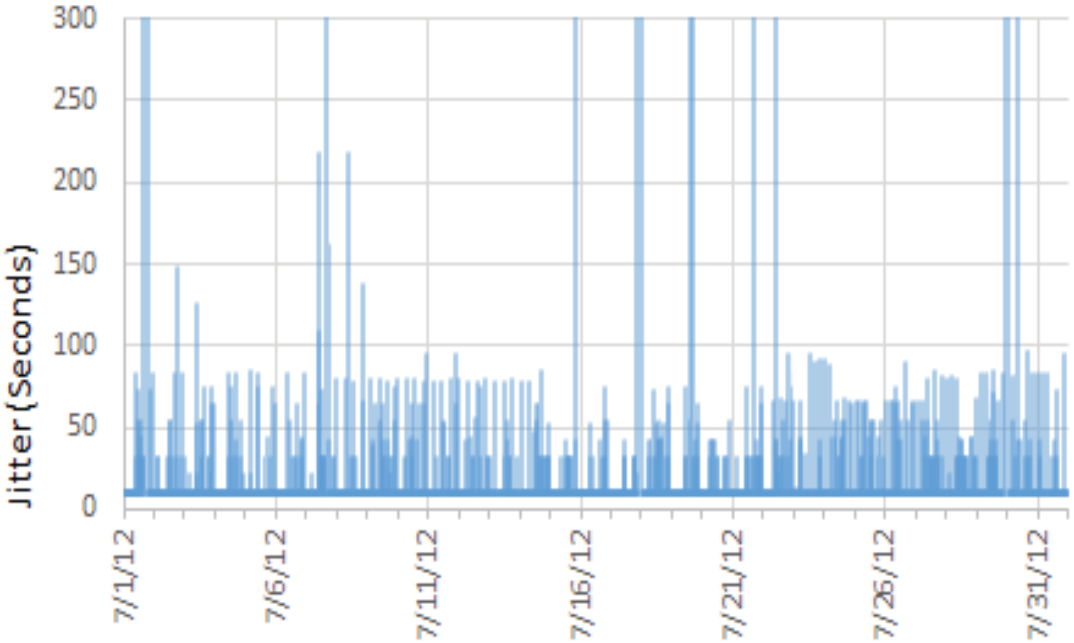

Figure 137 Jitter for a sensing unit for a period of one month

7 Conclusion

With the advancements in technology, it is expected that the availability of internet is everywhere and online at all time. Low-cost smart sensor node development enabled things to be connected easily and corresponding information can be accessible globally. With the features of scalability, fault tolerance and effective power consumption of nodes and transceiver “Internet of Things” have facilitated ubiquity computational ability to internetwork heterogeneous smart devices easily and facilitate availability of data anywhere. An efficient method for internetworking of WSN and the internet is given. The key idea of proposed method is to provide low-cost solution and flexible connection mechanisms for integrating Internet of things with home automation systems. The advantages of the developed system are to have greater control over routing of packets (security and customization) and ability to adapt to other wireless sensor networks.

Chapter 2 provides a background into the IoT to give a context and scope to the WSN-IoT platforms developed.

Chapter 4 provides the structure of the ZigBee based WSN-IoT platform, and the 6LoWPAN based WSN-IoT platform.

Chapter 5 provides the implementation details of developing the WSN-IoT platform. Details of the software tools, and software packages required for the IoT-WSN platform are given. Two WSN-IoT platforms were developed using ZigBee and 6LoWPAN. The implementation of the ZigBee based WSN-IoT platform consists of the ZigBee WSN, and ZigBee IoT gateway. The implementation of the 6LoWPAN based WSN-IoT platform consisted of the 6LoWPAN nodes, and a 6LoWPAN gateway. An IoT server to collect the sensor data from the ZigBee based platform and 6LoWPAN platform was implemented. The server consisted of an IoT service to collect sensor information, a database to store sensor information, and a website to display sensor information.

Chapter 6 provides results from the WSN-IoT platform, which are data collected from a smart home, and quality of service parameters of the platform.

8 Challenges and Opportunities

Issues like availability of IPv6 connectivity may be major concern in implementing the methods as discussed. As most of the internet domains still operate using IPv4 and IPv6 adoption is low. Better compression techniques can be implemented for minimizing storage requirements and effective retrieval of data. Security Issues related to data transmission in the WSN need to be investigated to ensure sensor data is transmitted securely to the gateway.

8.1 Future Works

There are several improvements that can be made to both implementations (ZigBee based and 6LoWPAN based). The following sections give details of improvements that could be made to each implementation.

8.1.1 ZigBee based IoT Platform Future Works

It was observed that collecting multiple sensor data over long periods of time generates a huge amount of data. In order to reduce the amount of data stored without losing important information we have implemented the variable sensor data recording technique. This has resulted in significant performance improvement for the real-time data display. The next step will be investigating an appropriate model for continuous heterogeneous sensor data storage mechanism.

Additional management of ZigBee nodes needs to be implemented to facilitate the deploying and tracking of wireless sensor nodes. Storage of information such as a nodes location, calibration information, and networking parameters would facilitate the management of the sensor nodes.

8.1.2 6LoWPAN based IoT Platform Future Works

The 6LoWPAN implementation had several stability problems, which can be improved with more rigorous testing methods. The combination of several software and hardware components made debugging errors very difficult when the system was running. Compartmentalising the software components into controlled simulated environments would allow for rigorous long term testing.

The 6LoWPAN node needs to be tested for long term stability in an environment where there are no other sources of error. This means the 6LoWPAN stack needs to be tested independently of the radio driver. The radio driver may cause the 6LoWPAN stack to fail or vice-versa, making finding the source of the problem difficult when it occurs at random after a long period.

The radio interface provided to the gateway also needs to be tested independently to ensure it is reliable and fault tolerant if the gateway fails. The radio interface also needs to be tested in

combination with the radio driver for the 6LoWPAN node to ensure data transmission is reliable and incorrectly received data is handled appropriately.

Once the stability and reliability of the 6LoWPAN nodes has been improved the following features need to be implemented in order to fully utilize 6LoWPAN technology.

- Neighbour discovery [106] which will allow for address auto configuration.
- Constrained Application Protocol (CoAP) [111] which will allow for RESTful services to be developed, making the wireless sensor network behave similar to most web related protocols. However this protocol is still in the draft stages, meaning it should not be used until it becomes a RFC.

9 References

[1] K. Ashton, “That 'Internet of Things' Thing,” [Online]. Available: http://www.rfidjournal.com/articles/view?4986. [Accessed 20 05 2013].

[2] P. Friess and P. Guillemin, “Internet of things strategic research roadmap,” The Cluster of European Research Projects, 2009.

[3] BCC Research, “Sensors: Technologies and global markets,” BCC Research, 2011.

[4] I. F. Akyildiz, Y. Weilian, Y. Sankarasubramaniam and E. Cayirci, “A survey on sensor networks,” *Communications Magazine, IEEE,* vol. 40, no. 8, pp. 102-114, August 2002.

[5] O. Corcho and R. Garcia-Castro, “Five Challenges for the Semantic Sensor Web,” *Semantic Web,* vol. 1, no. 1,2, pp. 121-125, 2010.

[6] L. Atzori, A. Iera and G. Morabito, “The Internet of Things: A survey,” *Computer Networks,* vol. 54, no. 1, pp. 2787-2805, 2010.

[7] A. Dunkels and J. P. Vasseur, *IP for Smart Objects,* IPSO Alliance White Paper No. 1, 2008.

[8] D. Surie, O. Laguionie and T. Pederson, “Wireless sensor networking of everyday objects in a smart home environment,” in *Intelligent Sensors, Sensor Networks and Information Processing*, Sydney, 2006.

[9] H. Sundmaeker, P. Guillemin, P. Friess and S. Woelffle, Vision and challenges for realising the Internet of things, Luxembourg: European Union, 2010.

[10] C. Bizer, . J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak and S. Hellmann, “A crystallization point for the Web of Data,” *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 7, no. 3, pp. 154-165, 2009.

[11] M. Compton, C. Henson, L. Lefort, H. Neuhaus and A. Sheth, “A Survey of the Semantic Specification of Sensors,” in *8th International Semantic Web Conference (ISWC 2009), 2nd International Workshop on Semantic Sensor Networks*, Washington DC, 2009.

[12] K. Aberer, M. Hauswirth and A. Salehi, “The Global Sensor Networks middleware for efficient and flexible deployment and interconnection of sensor networks,” in *7th International*

Middleware Conference, 2006.

[13] LogMeIn, Inc, “Xively – Public Cloud for the Internet of Things,” [Online]. Available: https://xively.com/. [Accessed 2 June 2013].

[14] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz and D. Culler, “sMAP: a simple measurement and actuation profile for physical information,” in *SenSys '10 Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, New York, 2010.

[15] A. Kansal, S. Nath, J. Liu and F. Zhao, “SenseWeb: An Infrastructure for Shared Sensing,” *IEEE MultiMedia,* vol. 14, no. 4, pp. 8-13, 2007.

[16] “Feel, Act, Make sense • Sen.se,” [Online]. Available: http://open.sen.se/. [Accessed 17 11 2012].

[17] Etherios Inc., “Device Cloud: Driving the Internet of ANYthing,” [Online]. Available: http://www.etherios.com/products/devicecloud/. [Accessed 5 June 2013].

[18] J. Beutel, “Metrics for Sensor Network Platforms,” in *ACM Workshop on Real-World Wireless Sensor Networks (REALWSN'06), ACM Press*, New York, 2006.

[19] D. Bri, H. Coll, M. Garcia and J. Lloret, “A Wireless IP Multisensor Deployment,” *Journal On Advances in Networks and Services,* vol. 3, no. 1,2, p. 14, 2010.

[20] C. Lynch and F. O'Reilly, “PIC-based TinyOS Implementation,” in *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, 2005.

[21] M. Eisenhauer, P. Rosengren and P. Antolin, “A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems,” in *6th Annual IEEE Communications Society Sensor, Mesh and Ad Hoc Communications and Networks Workshops, SECON*, Rome, 2009.

[22] S. Hong, D. Kim, M. Ha, S. Bae, S. Park, W. Jung and J. Kim, “SNAIL: an IP-based wireless sensor network approach to the internet of things,” *IEEE Wireless Communications,* vol. 17, no. 6, pp. 34-42, 2010.

[23] N. Bui, A. Castellani, P. Casari and M. Zorzi, “The internet of energy: a web-enabled smart grid system,” *IEEE Network,* vol. 26, no. 4, pp. 39-45, 2012.

[24] A. Iera, C. Floerkemeier, J. Mitsugi and G. Morabito, “The Internet of things,” *IEEE Wireless Communications,* vol. 17, no. 6, pp. 8-9, 2010.

[25] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton and T. Razafindr alambo, “A survey on facilities for experimental internet of things research,” *IEEE Communications Magazine,* vol. 49, no. 11, pp. 58-67, 2011.

[26] M. Zorzi, A. Gluhak, S. Lange and A. Bassi, “From today's INTRAnet of things to a future INTERnet of things: a wireless- and mobility-related view,” *IEEE Wireless Communications,* vol. 17, no. 6, pp. 44-51, 2010.

[27] A. Sehgal, V. Perelman, S. Kuryla and J. Schonwalder, “Management of resource constrained devices in the internet of things,” *IEEE Communications Magazine,* vol. 50, no. 12, pp. 144-149, 2012.

[28] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura and E. Jansen, “he gator tech smart house: A programmable pervasive space,” *IEEE Computer,* vol. 38, no. 3, pp. 50-60, 2005.

[29] D. Cook, “Learning Setting-Generalized Activity Models for Smart Spaces,” *IEEE Intelligent Systems,* vol. 27, no. 1, pp. 32-38, 2012.

[30] F. Doctor, H. Hagras and V. Callaghan, “A fuzzy embedded agent-based approach for realizing ambient intelligence in intelligent inhabited environments,” *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans,* vol. 35, no. 1, pp. 55-65, 2005.

[31] J. Kientz, S. Patel, B. Jones, E. Price, E. Myantt and G. Abowd, “The Georgia Tech aware home,” in *Proceedings of the Extended Abstracts on Human Factors in Computing Systems*, 2008.

[32] I. S. Larson, K. Tapia, E. Beaudin, J. Kaushik, P. Nawyn and J. R. Rockinson, “Using a Live- In Laboratory for Ubiquitous Computing Research,” in *Pervasive Computing, Lecture Notes in Computer Science*, Berlin, Springer Berlin Heidelberg, 2006, p. 349–365.

[33] N. K. Suryadevara and S. C. Mukhopadhyay, “Wireless Sensor Network based Home Monitoring System for Wellness Determination of Elderly,” *IEEE Sensors Journal,* vol. 12, no. 6, pp. 1965-1972, 2012.

[34] L. C. DeSilva, M. Chamin and M. P. Iskandar, “State of the art of Smart Homes,” *Engineering Applications of Artificial Intelligence,* vol. 25, no. 7, pp. 1313-1321, 2012.

[35] I. Howitt and J. A. Gutierrez, “IEEE 802.15.4 low rate - wireless personal area network coexistence issuues,” in *Proceedings of the IEEE-Wireless Communications and Networking*, New Orleans, 2003.

[36] L. Angrisani, M. Bertocco, D. Foortin and A. Sona, “Assessing coexistence problems of IEEE 802.11b and IEEE 802.15.4 wireless networks through cross-layer measuremennts,” in *Instrumentation and Measurement Technology Conference Proceedings*, Warsaw, 2007.

[37] H. Khaleel, C. Pastrone, F. Penn a, M. A. Spirito and R. Garello, “mpact of Wi-Fi traffic on the IEEE 8022.15.4 channels occupation in indoor environments,” in *International Conferrence on Electromagneticcs in Advanced Applications*, 2009.

[38] N. Kushalnagar, G. Montenegro, J. Hui and D. Culler, “Transmission of IPv6 Packets over IEEE 802.15.4 Networks, IETF RFC 4944,” September 2007. [Online]. Available: http://tools.ietf.org/rfc/rfc4944.txt.

[39] K. Lee, J. Won and C. Bae, “IGN2IP: Internetworking Intelligent Gadget Network based on 802.15.4 with IP Network,” in *Proceedings of the International Conference on Consumer Electronics, ICCE 2008, Digest of Technical Papers*, Las Vegas, 2008.

[40] F. Mesrinejad, F. Hashim F, N. K. Noordin, M. F. A. Rasid and R. S. A. R. Abdullah, “The Effect of Fragmentation and Header Compression on IP-Based Sensor Networks (6LoWPAN),” in *Proceedings of the 17th Asia-Pacific Conference on Communications (APCC)*, Sabah, 2011.

[41] R. Roman and J. Lopez, “Integrating Wireless Sensor Networks and the Internet: a Security Analysis,” *Internet Research,* vol. 19, no. 2, pp. 246-259, 2009.

[42] D. Christin, A. Reinhardt, P. S. Mogre and R. Steinmetz, “Wireless Sensor Networks and the Internet of Things: Selected Challenges,” in *Proceedings of the 8th GI/ITG KuVS Fachgespräch Drahtlose Sensornetze (FGSN)*, 2009.

[43] HART Communication Foundation, “HART Communication Protocol and Foundation - Home Page,” [Online]. Available: http://www.hartcomm.org/. [Accessed 9 June 2013].

[44] ZigBee Alliance, Inc., “ZigBee Home Automation Public Application Profile, revision 26,” 8 February 2010. [Online]. Available: https://docs.zigbee.org/zigbee-docs/dcn/07/docs-07-5367-02-0afg-home-automation-profile-for-public-download.pdf. [Accessed 9 June 2013].

[45] Z-Wave Alliance, “Welcome To The Z-Wave Alliance,” [Online]. Available: http://www.z-wavealliance.org/. [Accessed 9 June 2013].

[46] INSTEON, “INSTEON - Wireless Home Control Solutions for Lighting, Security, HVAC, and A/V Systems,” [Online]. Available: http://www.insteon.com/. [Accessed 9 June 2013].

[47] Coronis, “wavenis-wireless-technology-presentation,” [Online]. Available: http://www.coronis.com/en/wavenis_technology.html. [Accessed 9 June 2013].

[48] C. Gomez and J. Paradells, “Wireless home automation networks: A survey of architectures and technologies,” *Communications Magazine, IEEE ,* vol. 48, no. 6, pp. 92-101, 2010.

[49] Etherios, Inc., “Device Cloud Services - Device Cloud by Etherios,” [Online]. Available: https://myaccount.etherios.com/Profile/Service.aspx. [Accessed 10 June 2013].

[50] element 14 Ltd, “XB24-BWIT-004 - DIGI INTERNATIONAL - ZIGBEE MODULE, XBEE ZNET, WIRE | element14 New Zealand,” [Online]. Available: 8. http://nz.element14.com/digi-international/xb24-bwit-004/module-zigbee-xbee-znet-2-5/dp/1546390 (02/03/2013). [Accessed 2 March 2013].

[51] RS Components Ltd , “Buy Wireless Routers ConnectPort X2 Smart Energy Starter Kit Digi International XK-SE1-EC-W online from RS for next day delivery.,” [Online]. Available: http://newzealand.rs-online.com/web/p/wireless-routers/7043629/. [Accessed 10 June 2013].

[52] N. K. Suryadevara, A. Gaddam, R. K. Rayudu and S. C. Mukhopadhyay, “Wireless Sensors Network Based Safe Home to Care Elderly People: Behaviour Detection,” *Procedia Engineering,* vol. 25, pp. 96-99, 2011.

[53] C. Ranhotigamage and S. C. Mukhopadhyay, “Field Trials and Performance Monitoring of Distributed Solar Panels Using a Low Cost Wireless Sensors Network for Domestic Applications,” *IEEE Sensors Journal,* vol. 11, no. 10, pp. 2583-2590, 2011.

[54] K. Kaur, S. C. Mukhopadhyay, J. Schnepper, M. Haefke and H. and Ewald, “ ZigBee Based Wearable Physiological Parameters Monitoring System,” *IEEE Sensors Journal,* vol. 12, no. 3, pp. 423-430, 2012.

[55] H. Alabri, S. C. Mukhopadhyay, G. A. Punchihewa, N. K. Suryadev ara and Y. M. Huang, “Comparison of applying Sleep Mode function to the Smart Wireless Environmental Sensing Stations for Extending the Life time,” in *Proceedings of the IEEE International Instrumentation*

and Measurement Technology Conference (I2MTC), Graz, 2012.

[56] G. M. Mendez, M. A. M. Yunus and S. C. Mukhopadhyay, "A WiFi based Smart Wireless Sensor Network for Monitoring an Agricultural Environment," in *Proceedings of the IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, Graz, 2012.

[57] "IEEE Standard 802.15.4-2006," 8 September 2006. [Online]. Available: http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf.

[58] element 14 Ltd., "ATAVRZRAVEN - ATMEL - ATMEGA1284PV, RADIO TXRX, EVAL | element14 New Zealand," [Online]. Available: http://nz.element14.com/atmel/atavrzraven/atmega1284pv-radio-txrx-eval-kit/dp/1562233. [Accessed 28 February 2013].

[59] element 14 Ltd., "ATAVRDRAGON - ATMEL - IN SYSTEM DEBUGGER / PROGRAMMER | element14 New Zealand," [Online]. Available: http://nz.element14.com/atmel/atavrdragon/in-system-debugger-programmer-avr/dp/1455088. [Accessed 02 February 2013].

[60] Atmel Corporation, "ATmega1284P," [Online]. Available: http://www.atmel.com/devices/atmega1284p.aspx?tab=parameters. [Accessed 28 February 2013].

[61] Atmel Corporation, "AT86RF230," [Online]. Available: http://www.atmel.com/devices/at86rf230.aspx?tab=parameters. [Accessed 28 February 2013].

[62] element 14 Ltd., "STM32WC-RFCKIT - STMICROELECTRONICS - KIT, RF CONTROL, STM32W | element14 New Zealand," [Online]. Available: http://nz.element14.com/stmicroelectronics/stm32wc-rfckit/kit-rf-control-stm32w/dp/2215468. [Accessed 28 February 2013].

[63] element 14 Ltd., "ST-LINK/V2 - STMICROELECTRONICS - ICD/PROGRAMMER, USB 2.0, JTAG | element14 New Zealand," [Online]. Available: http://nz.element14.com/stmicroelectronics/st-link-v2/icd-programmer-usb-2-0-jtag-for/dp/2119510. [Accessed 28 February 2013].

[64] STMicroelectronics, "STM32W108CC - STMicroelectronics," [Online]. Available: http://www.st.com/web/catalog/mmc/FM141/SC1169/SS1581/PF251886. [Accessed 28 February 2013].

[65] Mouser Electronics, “EM250-JMP-R Silicon Labs | Mouser,” [Online]. Available: http://nz.mouser.com/ProductDetail/Silicon-Labs/EM250-JMP-R/?qs=sGAEpiMZZMv6Wlt51A2SLc49eozxKRPa. [Accessed 2 March 2013].

[66] Silicon Laboratories Inc., “EM250 Data Sheet,” [Online]. Available: http://www.silabs.com/Support%20Documents/TechnicalDocs/EM250.pdf . [Accessed 15 March 2013].

[67] element 14 Ltd, “MSP430-CCRF - OLIMEX - CC430F5137, TRANSCEIVER, DEV BOARD | element14 New Zealand,” [Online]. Available: http://nz.element14.com/olimex/msp430-ccrf/board-dev-ti-cc430f5137/dp/2061336. [Accessed 13 March 2013].

[68] element 14 Ltd., “MSP-EXP430G2 - TEXAS INSTRUMENTS - MSP430G2XX, LAUNCHPAD, DEV KIT | element14 New Zealand,” [Online]. Available: http://nz.element14.com/texas-instruments/msp-exp430g2/msp430g2xx-launchpad-dev-kit/dp/185379301. [Accessed 15 March 2013].

[69] Texas Instruments Inc., “CC430F613x, CC430F612x, CC430F513x MSP430 SoC With RF Core (Rev. G),” [Online]. Available: http://www.ti.com/lit/ds/symlink/cc430f5137.pdf. [Accessed 9 August 2012].

[70] Texas Instruments Inc., “Download CCS,” 16 05 2013. [Online]. Available: http://processors.wiki.ti.com/index.php/Download_CCS.

[71] Red Hat Inc., “Cygwin Download,” [Online]. Available: http://cygwin.com/setup.exe. [Accessed 27 11 2011].

[72] “CMake for cygwin download,” [Online]. Available: http://www.cmake.org/files/cygwin/make.exe. [Accessed 30 09 2011].

[73] OpenWRT Project, “OpenWRT Feeds,” [Online]. Available: http://wiki.openwrt.org/doc/devel/feeds. [Accessed 1 9 2012].

[74] OpenWRT Project, “Linksys WRT54G, WRT54GL and WRT54GS - OpenWrt Wiki,” [Online]. Available: http://wiki.openwrt.org/toh/linksys/wrt54g. [Accessed 13 03 2013].

[75] The Eclipse Foundation, “Eclipse Download Page,” [Online]. Available: http://www.eclipse.org/downloads/. [Accessed 8 10 2011].

[76] Oracle, “Java SE downloads,” [Online]. Available: http://www.oracle.com/technetwork/java/javase/downloads/index.html. [Accessed 08 10 2011].

[77] Microsoft, “Download | Microsoft Visual Studio 2012,” [Online]. Available: http://www.microsoft.com/visualstudio/eng/downloads#d-express-windows-8. [Accessed 3 June 2013].

[78] Oracle Corporation, “MySQL :: Download Connector/Net,” [Online]. Available: http://dev.mysql.com/downloads/connector/net/. [Accessed 4 June 2013].

[79] “Download WampServer from SourceForge.net,” [Online]. Available: http://sourceforge.net/projects/wampserver/files/WampServer%202/WampServer%202.2/wampserver2.2e/wampserver2.2e-php5.3.13-httpd2.2.22-mysql5.5.24-32b.exe/download. [Accessed 3 June 2013].

[80] Microsoft, “Download Microsoft Visual C++ 2010 SP1 Redistributable Package (x86) from Official Microsoft Download Center,” [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=8328. [Accessed 1 June 2013].

[81] OpenWRT Project, “OpenWRT Buildroot - Usage,” [Online]. Available: http://wiki.openwrt.org/doc/howto/build. [Accessed 7 3 2012].

[82] OpenWRT Project, “OpenWRT Buildroot - Installation,” [Online]. Available: http://wiki.openwrt.org/doc/howto/buildroot.exigence. [Accessed 9 03 2012].

[83] S. Kelly, “Custom openwrt firmware for WRT54GL,” [Online]. Available: http://228899seankelly.googlecode.com/svn/trunk/WRT54GL/firmware/openwrt-brcm47xx-squashfs.trx. [Accessed 1 June 2013].

[84] “How to Change your IP Address (Windows),” [Online]. Available: http://www.wikihow.com/Change-your-IP-Address-(Windows). [Accessed 1 June 2013].

[85] Colorado State University, “Basic vi commands,” [Online]. Available: http://www.cs.colostate.edu/helpdocs/vi.html. [Accessed 14 09 2011].

[86] OpenVPN Technologies, Inc, “Community Downloads,” [Online]. Available: http://openvpn.net/index.php/download/community-downloads.html. [Accessed 4 June

2013].

[87] OpenVPN Technologies, Inc, “HOWTO,” [Online]. Available: http://openvpn.net/index.php/open-source/documentation/howto.html. [Accessed 2 June 2013].

[88] OpenWRT Project, “The UCI System,” [Online]. Available: http://wiki.openwrt.org/doc/uci. [Accessed 24 02 2012].

[89] Digi International, “XBee/XBee-Pro ZB RF Modules - Product Manual,” [Online]. Available: http://ftp1.digi.com/support/documentation/90000976_P.pdf. [Accessed 20 04 2013].

[90] J. Reynolds and J. Postel, “Assigned Numbers, IETF RFC 1700,” 10 1994. [Online]. Available: http://www.ietf.org/rfc/rfc1700.txt. [Accessed 25 02 2013].

[91] R. Meyers, “The New C: X Macros,” 05 2001. [Online]. Available: http://www.drdobbs.com/the-new-c-x-macros/184401387. [Accessed 17 02 2013].

[92] OpenWRT Project, “OpenWrt,” [Online]. Available: https://openwrt.org/. [Accessed 16 11 2012].

[93] NewMedia-NET GmbH, “www.dd-wrt.com | Unleash Your Router,” [Online]. Available: http://www.dd-wrt.com. [Accessed 21 04 2013].

[94] J. Zarate, “Tomato Firmware | polarcloud.com,” [Online]. Available: http://www.polarcloud.com/tomato. [Accessed 21 04 2013].

[95] “RPi Low-level peripherals - eLinux.org,” [Online]. Available: http://elinux.org/RPi_Low-level_peripherals. [Accessed 3 09 2012].

[96] STMicroelectronics, “LExxAB, LExxC Datasheet,” [Online]. Available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00000545.pdf. [Accessed 15 05 2013].

[97] Analog Devices, “TMP35/TMP36/TMP37 Datasheet,” [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/TMP35_36_37.pdf. [Accessed 13 06 2012].

[98] B. Carter and R. Mancini, Op Amps for Everyone, Third Edition, Newnes, 2009.

[99] Talema, “ASM Series Datasheet,” [Online]. Available: http://www.tme.eu/en/Document/4bf379646b30b1f30becba759f9411ed/asm_series.pdf. [Accessed 16 04 2012].

[100] Texas Instruments Inc., “CC430 Product Bulletin,” [Online]. Available: http://www.ti.com/lit/ml/slat124a/slat124a.pdf. [Accessed 29 03 2012].

[101] Texas Instruments Inc., “CC430 Wireless Development Tool - EM430F6137RF900 - TI Tool Folder,” [Online]. Available: http://www.ti.com/tool/em430f6137rf900. [Accessed 19 January 2012].

[102] Olimex, “MSP430-CCRF User Manual,” [Online]. Available: https://www.olimex.com/Products/MSP430/Starter/MSP430-CCRF/resources/MSP430-CCRF.pdf. [Accessed 19 September 2012].

[103] Texas Instruments Inc., “MSP430 LaunchPad (MSP-EXP430G2) - Texas Instruments Wiki,” [Online]. Available: http://processors.wiki.ti.com/index.php/MSP430_LaunchPad_(MSP-EXP430G2). [Accessed 1 June 2013].

[104] Texas Instruments Inc., “MSP430™ Programming Via the JTAG Interface,” [Online]. Available: http://www.ti.com/lit/ug/slau320i/slau320i.pdf. [Accessed 2 June 2013].

[105] J. Hui and P. Thubert, “Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks, IETF RFC 6282,” September 2011. [Online]. Available: http://tools.ietf.org/rfc/rfc6282.txt.

[106] Z. Shelby, S. Chakrabarti, E. Nordmark and C. Bormann, “Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs), IETF RFC 6775,” November 2012. [Online]. Available: http://tools.ietf.org/rfc/rfc6775.txt.

[107] Texas Instruments Inc., “CC430 Family User's Guide,” [Online]. Available: http://www.ti.com/lit/ug/slau259e/slau259e.pdf. [Accessed 15 February 2013].

[108] Texas Instruments Inc., “SmartRF Studio - SMARTRFTM-STUDIO - TI Software Folder,” [Online]. Available: http://www.ti.com/tool/smartrftm-studio. [Accessed 19 January 2013].

[109] Highsoft Solutions AS, “Highcharts - Interactive JavaScript charts for your webpage,” [Online]. Available: http://www.highcharts.com/. [Accessed 5 July 2012].

[110] A. Zakaria, “Quality of service in Wireless Sensor networks,” [Online]. Available: http://cs.uwindsor.ca/richard/cs510/survey_zakaria.pdf. [Accessed 6 October 2012].

[111] Z. Shelby, K. Hartke and C. Bormann, *Constrained Application Protocol (CoAP),* IETF Internet draft, work in progress, 2013.

[112] R. Khoshdelniat, G. R. Sinniah, K. A. Bakar, M. H. M. Shaharil, Z. Suryady and U. Sarwar, “Performance Evaluation of IEEE802.15.4 6LoWPAN Gateway,” in *Proceedings of the 17th Asia-Pacific Conference on Communications*, Sabah, Malaysia, 2011.

[113] Digi International Inc, “X-CTU Software - Diagnostics, Utilities & MIBs - Digi International,” [Online]. Available: http://www.digi.com/support/productdetail?pid=3352&osvid=57&type=utilities. [Accessed 5 June 2013].

10 Publications

- S.D.T.Kelly, N.K.Suryadevara and S.C.Mukhopadhyay, Integration of Zigbee-IPv6 Networks for Smart Home Sensor Data Transmission to Augment Internet of Things, Proceedings of the 7th IB2COM, November 5-8, 2012, Sydney, Australia, pp. 44-49.
- Kelly, S.D.T.; Suryadevara, N.; Mukhopadhyay, S.C., "Towards the Implementation of IoT for Environmental Condition Monitoring in Homes," *Sensors Journal, IEEE (early access article)*, vol.PP, no.99, pp.1,1, 0, doi: 10.1109/JSEN.2013.2263379