KTH ROYAL INSTITUTE OF TECHNOLOGY

Master Thesis Report

Title involving {BLE, 802.15.4, Contiki}

Author: PrithviRaj Narendra

Supervisors: Simon Duquennoy

Examiner: Prof. Mats Brorsson, KTH, Sweden

Abstract

Hello Abstract.

Contents

1 | Introduction

1.1 General introduction to the domain of the thesis

What is BLE? Why was it developed and who/where is it being used? [1]

What is Contiki OS? Why was it and who/where is it being used? Explain the amount of R&D done in terms of communication protocols based on 802.15.4 [2].

1.2 Problem definition

Need for BLE in Contiki as a platform for Internet of Things (IoT). To understand the characteristics of BLE and 802.15.4 so that we can identify the applications suitable for these protocols.

1.3 Goal

To choose and integrate a BLE hardware platform with Contiki. To compare BLE and 802 in various performance criteria of data-rate, latency, reliability and energy consumption in environments with and without external interference.

1.4 Outline of the report

1.method??

2 | Background*

2.1 Overview of Bluetooth Low Energy

Rest of overview, from what is missed in 1st chapter.

2.1.1 Design objectives of BLE

BLE was designed by Bluetooth Special Interest Group (SIG) from the ground up, which helped it achieve certain design goals. These design goals for this wireless personal area network were *low cost, worldwide operation, short range, robustness* and *low power*[3].

Low Power BLE aims to use a tiny batteries such as button cell to keep a device operating for months to years. To achieve this goal, BLE was optimized to communicate small amounts of data, such as the states of devices. Also BLE is optimized to have lower peak power requirements, which allows use of button cells to be used with BLE devices.

Worldwide Operation For a technology to be adopted, it is important that there is uniform conformity to the regulations around the world. The 2.4 GHz Industrial, Scientific and Medical (ISM) radio band is the only one available license free worldwide. The technology to develop wireless devices in this band is mature making it the suitable radio band for BLE.

Short Range BLE was designed to be for personal area network like Classic Bluetooth, which means that it is not a network to work with a cellular base station network. This design criterion goes hand in hand with *low power*.

Low Cost Lower power requirements mean that the batteries in BLE devices need to smaller and have to be replaced less frequently, both resulting in a reduction of cost for both the manufacturer and the customer. The use of the ISM band for communication levels removes the licensing entry barrier for start-ups to develop BLE devices. BLE embraces simplicity in its pursuit to lower the cost. BLE supports only single-hop communication in a star network, which reduces the memory and processor requirement for supporting the protocol. Simplicity was the key factor for the choosing of Gaussian Frequency Shift Keying (GFSK) as the modulation scheme for BLE to result in low cost, small radio implementation of the radio in the Integerated Circuits (ICs) for BLE.

Robustness The 2.4 GHz space is crowded with devices communicating with various standards as well as spurious noise making the robustness a key criteria in developing BLE standard. BLE uses a multi-channel hopping mechanism called Adaptive Frequency Hopping (AFH) to detect, avoid and recover from interference. In addition to AFH, BLE uses Cyclic Redundancy Check (CRC) to detect and recover from bit-errors due to background noise.

*Includes text and images from minor thesis 'Business ideation for BLE'

2.1.2 BLE network architecture

Single and Dual devices, Advertisers and scanners, Master and Slave, Star Network, figure 2.1

Figure 2.1: Typical BLE network

2.2 BLE stack overview

Explain host and controller division. More details about link layer.

Image of the stack

The link layer of 802.15.4 consists of MAC layer on top of a Radio Duty Cycle (RDC) layer. For the RDC layer the Null-RDC and ContikiMAC driver will be tested in each of the test with Carrier Sense Multiple Access (CSMA) as the MAC layer. In case of Contiki-MAC the receiving node switches on periodically to sense if there are any packets that need to be received. The default time of this period is 125 ms. In case of null-RDC the radio receiver is never switched off, as the name suggests.

In BLE, the devices can assume different roles in the different layers of the protocol. In the link layer a device can be a 'Master' or a 'Slave'. In the Attribute Protocol (ATT) layer, a device can be a 'Client' and/or 'Server'. A server contains data and the client can request data from the server.

Physical layer

Link layer

Logical Link Control and Adaptation Protocol (L2CAP)

Generic Access Profile (GAP)

ATT

Generic Attribute Profile (GATT)

Security Manager (SM)

2.3 Overview of Contiki

??????????????? Which aspects of Contiki here?

2.4 Overview of 802.15.4

Mention that in this thesis the Contiki specific implementations of the 802.15.4 layers will be tested.

2.4.1 Physical layer

2.4.2 MAC layer

2.4.2.1 RDC layer

2.4.2.2 CSMA

3 | Methodology

3.1 Research process

Figure 3.1 shows tasks performed in this thesis boxes of the flow-chart and the steps of the research process on the left. The thesis report is also organized in this manner.

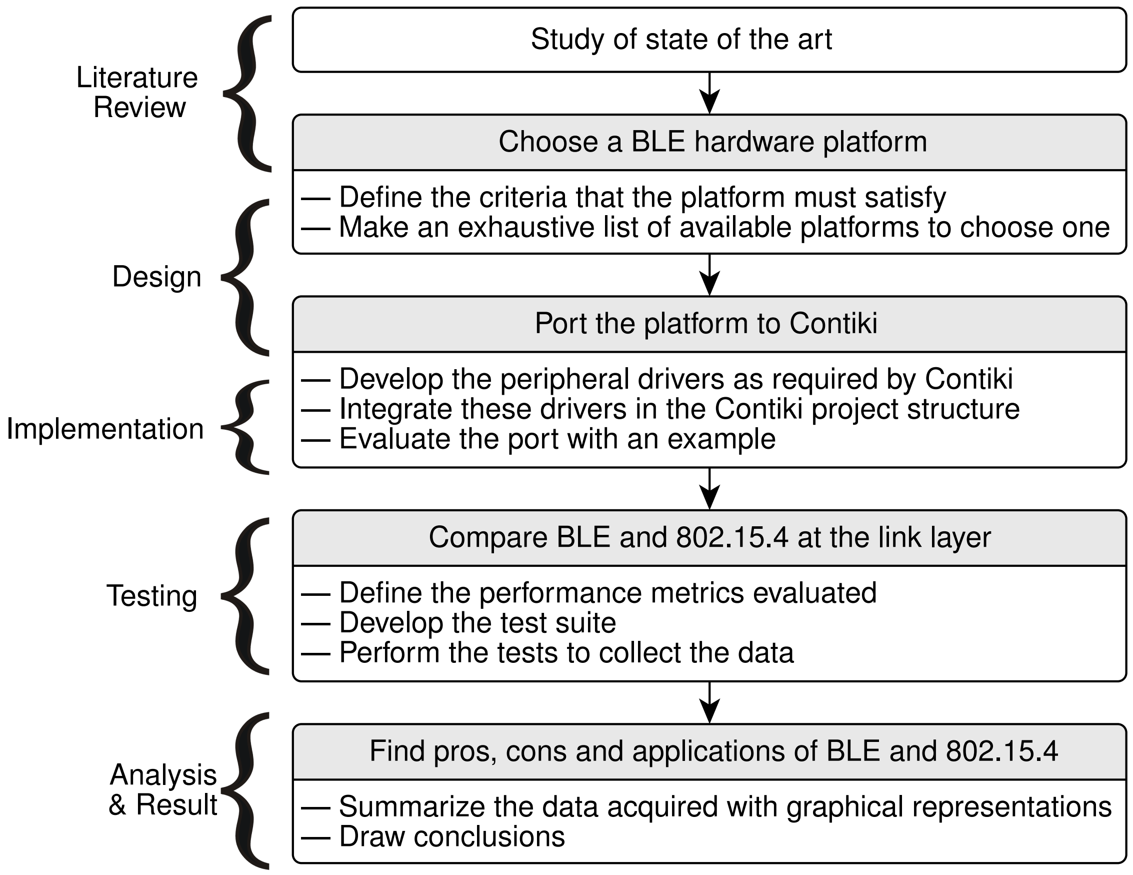

Literature Review
Study of state of the art
Choose a BLE hardware platform
Design
— Define the criteria that the platform must satisfy
— Make an exhaustive list of available platforms to choose one
Port the platform to Contiki
Implementation
— Develop the peripheral drivers as required by Contiki
— Integrate these drivers in the Contiki project structure
— Evaluate the port with an example
Testing
Compare BLE and 802.15.4 at the link layer
— Define the performance metrics evaluated
— Develop the test suite
— Perform the tests to collect the data
Analysis & Result
Find pros, cons and applications of BLE and 802.15.4
— Summarize the data acquired with graphical representations
— Draw conclusions

Figure 3.1: Thesis' tasks and steps in the research process

3.2 Research Method

In this experimental research large amount of data was generated to compare the two protocols mentioned, hence utilizing quantitative research method. The approach followed was deductive as the experiment started by defining the metric to measure, then designing the test to perform and finally deducting the conclusions based on the collected data from the tests.

The data collected was condensed with descriptive statistics and graphical representation was used to showcase this information. Univariate analysis of the various metrics of the two protocols was compare the two protocols from these graphical representation.

4 | Literature Study

This chapter provides a brief overview of the related work being done in the research community. Research articles involving the process of porting of Contiki OS to a hardware platform are presented in section 4.1. Performance evaluation of BLE, 802.15.4 and their comparison are presented in section 4.2, 4.3 and 4.4 respectively.

4.1 Porting Contiki OS to a new platform

Contiki OS project originating in 2002 has been ported to an increasing number of hardware platforms [4]. Since Contiki is an open-source BSD Clause-3 licensed project, there are many projects in various hardware platforms based on a fork from the Contiki repository.

These hardware platforms' processors range across a spectrum of 8-bit (8051 and AVR), 16-bit (MSP430) to 32-bit (ARM Cortex-M, PIC32). Contiki has support for 802.15.4 based wireless communication with various external transceivers and SoCs with built in radio transceiver. Various common features of many platforms such as Light Emitting Diodes (LEDs), buttons and serial port have modules in Contiki for common Application Programming Interface (API) across platforms.

In [5] the authors describe Contiki's port to a CC2430 based platform manufactured by Sensinode Ltd. CC2430 is an enhanced Intel 8051 processor based SoC having 802.15.4 physical layer compatible radio transceiver. The authors fully debugged the port which had of a code footprint of about 100 kilobyte (kB), varying based on the compiler mode and the features enabled. Many new features were added to the port including support for Analog to Digital Conversion (ADC) unit, all the sensor available on the platform (accelerometer, light sensor, voltage and temperature sensors), watchdog timer and the general purpose buttons. Because of the limited stack availability of 233 bytes, many optimizations such as moving variables to external Random-Access Memory (RAM) memory space and re-writing the radio driver for CC2430 to prevent stack from overflowing.

Contiki was ported to two new platforms, namely MicaZ and TelosB in a Bachelor thesis [6]. TelosB, similar to the TMote-Sky platform, consists of a 8MHz 16-bit MSP430 microcontroller (MCU), a CC2400 transceiver with 802.15.4 Physical Layer (PHY) and a host of sensors to measure light, temperature and humidity. The port to TelosB was done by changing the port to the fully supported TMote-Sky platform. MicaZ platform consisted of a 8-bit Atmel ATMega128L MCU and a CC2400 transceiver. MicaZ platform needed rewriting of the code for processor abstraction so that the high level Contiki APIs could work.

Contiki has been ported into platform based on a ARM Cortex M3 processor to create a device wired with Ethernet to connected to the Internet [7]. Ethernet based networking with the libraries present in Contiki was developed in this project to demonstrate as a proof of concept.

There are many unofficial ports of Contiki to various hardware platforms, including ones based on ARM Cortex M3 based processors. This can be seen in the port to STM32F10x based platform [8] and LPC1768 based platform [9], both having Cortex M3 processor.

4.2 Evaluating the performance of BLE

In a paper [1] providing an overview and evaluation of BLE, its various parameters such as energy consumption, latency, maximum piconet size and throughput have been evaluated. Theoretical calculations were done to predict the lifetime of a BLE slave device running of a 230 mAh coin cell battery for different values of connection interval and slave latency. The average current consumption of a TI CC2540 BLE platform was plotted for the entire range of connection interval keeping the slave latency as zero. The estimate of the lifetime was also done with respect to various Bit Error Rate (BER). **2.Talk about latency (676us) wrt Gomez2012, is what they are talking about called latency?**

The same experimental setup [1] shows the maximum throughput between two CC2540 devices as 58.48 kbps considering a payload of 20 bytes. This is attributed to the fact that in each connection event with an interval of 7.5 ms four packets are not transmitted as in an ideal case. An analytical model of the throughput [10] has been developed for different BER and connection interval. Simulation results validate this model developed. This model show that the maximum throughput in case the BER is zero is 236.7 kbps, independent of the connection interval. An assumption made in this paper is that the master and slave device do not have any limit on the number of packets communicated in a connection interval.

The energy consumption for an advertiser and scanner is modeled for the activity of the scanner discovering the advertiser [11]. The current consumption for each phase of advertisement and scanning is measured. Using this information the a model of the energy consumption is developed taking into account the advertising and scanning interval. The model developed is compared with experiments and validated.

The current consumption in different phases of operation, namely waking up, pre-processing, the transmission-reception cycles and the post-processing is measured [12] to estimate the lifetime. A recent journal article [13] develops a precise model of the energy consumption of BLE devices. A payload (20 bytes) throughput of 102 kbps is achieved in this paper consistent with the manufacturer's claims [14]. The model developed is unique in the sense that it is the first one which encompasses all the modes of operation, all the relevant parameters and their possible values. The model is based upon the actual measured duration of various parameters and measured current in various phases of operation. This leads to a model which at most has 6% variation from actual measurement. The code base for the model is available so that it can be ported to the system being evaluated.

Based on the model developed and evaluated, a set of guidelines are provided for developers of BLE system to reduce energy consumption [13]. In the unconnected mode, the scanner is recommended to be continuously scanning in case the advertisers is expected to be found soon. In case of where a lot of time is spent scanning idly, the parameters of advertising interval and duty cycle of the scanner can be tweaked to minimize both the energy consumption and latency of discovery. In the connected mode, the recommendation is to completely fill the payload as possible and communicate as much data as possible within a connection event. In both modes, it was noted that although reducing the transmission power appropriate to the the distance transmitted helped in reducing energy consumption, it was not as significant as the other factors.

4.3 Evaluating the performance of 802.15.4

[15] Adam Dunkels. “The ContikiMAC Radio Duty Cycling Protocol”. In: (Dec. 2011).

[16] Mathieu Michel and Bruno Quoitin. “Technical Report : ContikiMAC vs X-MAC performance analysis”. In: arXiv preprint arXiv:1404.3589 (Apr. 2014). arXiv: 1404.3589.

[17] M.-P. Uwase, M. Bezunartea, T.L. Nguyen, et al. “Experimental evaluation of message latency and power usage in WSNs”. In: 2014 IEEE International Black Sea Conference on

Communications and Networking (BlackSeaCom). IEEE, May 2014, pp. 69–72.

4.4 Comparison of BLE and 802.15.4

BLE is relatively new protocol compared to 802.15.4, due to which there are few studies conducted comparing the two low power wireless protocols. The energy consumption of BLE and 802.15.4 was compared in terms of the amount of payload can be communicated per Joule of energy i.e. the energy utility measured with unit kByte/J [18]. It was found that for BLE the energy utility was independent of the throughput and depended on the number of packets communicated per connection event. The energy utility varied from around 325 to 525 kByte/J when the packets per connection event rose from one to four respectively. In case of 802.15.4, the energy utility did depend on the throughput. Until 1 kBps the energy utility increased with respect to the throughput, then plateaued at 300 kByte/J.

In the same paper [18] the energy utility of both BLE and 802.15.4 is measured when transmitting a payload over the link layer and when transmitting payload with IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) with the application payload (up to 150 Bytes) as a factor. Both increase in a step-wise manner because of the maximum payload capacity in both protocols with BLE having larger number of step because of the lesser payload capacity of 27 byte at the link layer. The overheads of the different 6LoWPAN frames can also be seen in both the protocols.

The effect of WiFi interference on the reception of packets is also considered for the two protocols [18]. It should be noted that the test build did not support AFH to avoid interference, so the authors tested only the non-connected mode for BLE. In that case, up to 1 m the interference resulted in only 5 to 20% of the packets being received successfully, depending on the overlap of the interfering WiFi channel over the advertising BLE channel. 1.5 m and above the WiFi interference affected the communication little. In case of 802.15.4, the WiFi interference closer than 0.5 m resulted in only 35% of the packets being communicated. At a distance of 1 m and above, the interference had negligible effect on the packets being communicated.

Another journal article [19] compares BLE, 802.15.4 and another wireless protocol SimpliciTI over various criteria of throughput (theoretical and experimental), minimum turnaround time, energy consumption of the transceivers and the memory resources required for the stack of these protocols. Minimum turnaround time here is the time required for requesting data and receiving it. SimpliciTI is an flexible open-source low-power proprietary radio protocol developed by TI for their wireless products, compatible with 802.15.4 transceivers. Similar to [10], a the calculation of the maximum throughput for BLE provided a value greater than 300 kbps, now at the link layer. Since SimpliciTI does not have rigid specification, its parameters can be tweaked for the maximum throughput to be calculated as 350 kbps. 802.15.4's maximum throughput was calculated to be between 150 and 200 kbps. The experimental evaluation of the throughput for SimpliciTI and 802.15.4 peaked at about 160 kbps and 145 kbps respectively accounting to the pre-processing operations and CCA inc case of 802.15.4. In case of BLE, the stack allowed different number of packets per connection event based on the link layer payload size. This resulted in a the throughput increasing irregularly with the payload, peaking at 122.6 kbps.

The minimum turnaround time measured for BLE was estimated to be below 1 ms since the reply was expected after the Inter Frame Space (IFS) in the same connection event. Experiments show that this is actually 7.6 ms. This is consistent with the minimum connection interval of 7.5 ms after which the reply is received in the next connection event. The minimum turnaround time was estimated as 1.92 to 10.08 ms and 0.7 to 5 ms for 802.15.4 and SimpliciTI respectively based on the mode of operation. The measured value was between 1.5 and 3 ms higher than the estimated value for both 802.15.4 and SimpliciTI. The energy consumption for the three protocols were measure per transmission and per byte transmitted. It was found that BLE transceiver consumed 2 to 7 times lower energy than the other two transceivers depending on the mode of operation.

One more article compare the energy consumption of BLE, ZigBee and ANT wireless protocols for a low duty cycle application sending few bytes of data periodically [20]. Standardized in 2003, ZigBee is a wireless, low cost, low power, mesh networking standard [21] typically used in home automation, industrial control, wireless sensor networks and the like. ZigBee uses 802.15.4 for its physical and MAC layer. ANT is another 2.4 GHz ISM based wireless sensor network protocol supporting many network architectures as point to point, broadcasting and mesh. Its typical applications are in fitness and sports Personal Area Network (PAN). In the article [20] the test case involves devices based on these three protocols initiating connection and transferring 8 bytes of data and then disconnecting periodically at an interval ranging from 5 to 120 seconds. For this test case, it was found that the average current consumed by BLE was the lowest, followed by ZigBee and then ANT. **3.There is a flaw in their paper. They say that BLE connection takes time because of frequency hopping. There is no mention of advertising interval, scanning interval or duty cycle. These are the parameters which matters. Overall, it looks like they just bought a system, connected it to measuring device and ran tests without bothering about the protocol configuration. How did they submit to IEEE IWS symposium???**

5 | Porting a BLE platform to Contiki OS

This chapter describes the initial task of choosing the hardware platform supporting BLE and porting Contiki OS to this platform. This task was a pre-requisite to the next task of testing detailed in chapter 6. Section 5.1 explains the process of choosing a hardware platform and describes the chosen one. Section 5.2 describes the different aspects of porting Contiki to a new platform, with details of the specific port done in this project.

5.1 BLE Hardware Platform

As mentioned in section 4.1 Contiki has been ported to many hardware platforms. Since one of the main feature of Contiki is its communication stack capable of running in resource constrained systems, the primary aspect of a hardware platform is the MCU and the communication interface. When choosing a hardware platform there are many other aspects that need to be considered as well. These include the availability of source code for peripheral drivers and examples, development environment (compiler, linker, programmer and debugger), development boards, documentation of the entire system and online forum for discussion.

Before this project, Contiki only supported wireless communication based on 802.15.4 physical layer. There are two types of platforms, namely platforms which consist of a discrete radio transceiver with a MCU controlling it and platforms which consist of a SoC containing both the MCU and radio transceiver in the same IC. For including BLE support in Contiki, a hardware platform needed to be chosen and this process is explained in this section.

5.1.1 Requirements of the hardware platform

The *mandatory* requirements for the platform would be:

- A SoC based platform.
- Must have a well supported and documented processor with good specifications.
- For a open source project such as Contiki, a free (and preferably open source) development toolchain must support the platform.
- Availability of well documented datasheet and user manual.
- Availability of an evaluation/development kit.
- Enough memory to accommodate Contiki and BLE stack's requirements.
- Presence of basic peripherals such as timers and serial port required for Contiki.

The non-mandatory, although *nice to have* requirements for the platform would be:

- Presence of flexible power modes with low active and sleep power consumption.
- Availability of a good set of peripherals.
- Availability of BLE stack from the vendor, preferably with source code.

5.1.2 Comparison and selection of the hardware platform

With these requirements, based on the exhaustive comparison in Appendix A of the available SoC (BLE+MCU) solutions available today, a platform based on nRF51822 from Nordic-Semiconductors would be a suitable option. As seen from the table in Appendix A, this platform would satisfy all the requirements mentioned above except for that the BLE stack would be available as a binary file, without the source code.

As shown in the table in Appendix A, recently many new promising BLE based SoCs have be released such as Quintic 9020, Dialog Semiconductor DA14580, Lapis MLA7105 and Broadcom BCM20732. From the limited technical information available about them, their technical specifications would be suitable for a project like this. But because of the limited documentation about them, scarce availability and nascent support they are not suitable.

5.1.3 Overview of nrf51822 SoC and its platform

nrf51822 is a SoC made by Nordic Semiconductor for developing BLE and 2.4 GHz based wireless systems [22]. Most of the specification of this SoC can be found in the table in Appendix A. The ARM Cortex M0 present is a 32 bit, 3 stage pipeline processor with Von Neumann architecture. It is designed for low silicon die size, low cost and power. It has an integrated Nested Vectored Interrupt Controller (NVIC) responsible for handling processor exceptions and peripheral interrupts.

The development boards in the form of a USB dongle used for this thesis are called PCA10000. As seen in the figure 5.1, the top side of PCA10000 contains nrf51822 at the centre, powered from the USB port through a voltage regulator. This SoC is connected to a tri-colour RGB led, a 16 MHz crystal, a 32.768 kHz crystal and a PCB antenna with its matching network. On the other side of the PCB is the SEGGER JLink Lite Cortex M unit. This can program and debug using the Serial Wire Debug (SWD) port of nrf51822. Another useful feature of this board is that the SEGGER JLink unit provides a serial port over USB with hardware flow control (HWFC) to the computer that this dongle is connected to. This serial port is connected to the Universal Asynchronous Receiver/Transmitter (UART) port of nrf51822 [23].

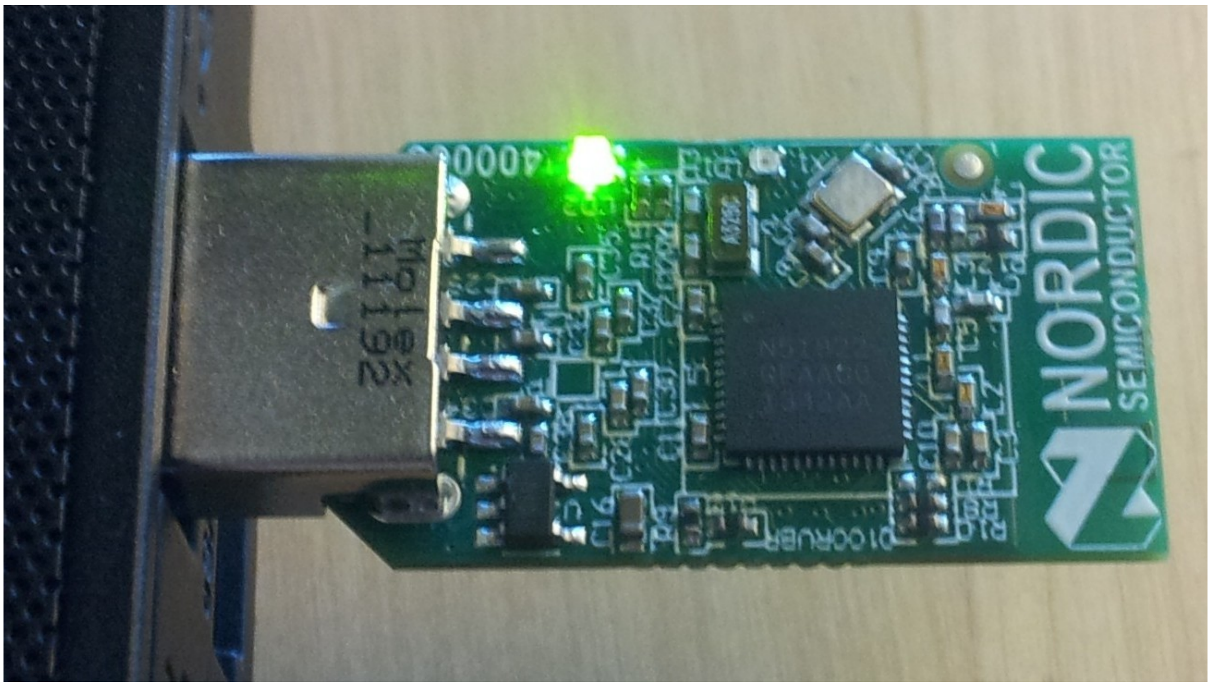


Figure 5.1: PCA10000 development board

Nordic Semiconductor provides BLE stack as a precompiled and linked binary file called *SoftDevices*. The SoftDevice is stored in a protected area in the flash memory and has access to a protected section of RAM memory, preventing unauthorized access by the application code. The SoftDevice can be accessed through a specified set of API calls. These APIs are accessed by making a supervisor call to the processor causing a exception handler to run the SoftDevice. All calls are non-blocking, which means that the call will not stall the application making the call. And there are synchronous and non-synchronous calls, where the synchronous calls immediately return the result while the asynchronous calls start an operation that will send the result as an event to the application. The SoftDevice is used for accessing the BLE stack as it was out of the scope of this thesis project to implement a BLE stack from scratch.

A Software Development Kit (SDK) is provided for nrf51822 which contains the peripheral drivers, examples, the interface header files for the SoftDevices and their documentation. Two tools provided by Nordic Semiconductor has been extensively used in this thesis. *nRF Sniffer* is a tool used with the PCA10000 board and *Wireshark* application to capture, view and save the information of BLE packets being sent between two devices. This greatly helps in learning about BLE packets and debugging problems. This tool is capable of providing detailed information about almost all the segments of the captured packets. It was found in during the thesis project that the nRF-Sniffer is not entirely capable of capture each and every packet being communicated, which does limit its use as tool to get feedback as one develops low level BLE drivers. Another invaluable tool provided by Nordic Semiconductor is *Master Control Panel*. It is an Android application which acts a generic BLE application capable of discovering BLE devices, connect and communicate with them while providing an overview of their Attribute database.

From actively using this platform for many months, the subjective pros and cons of this platform are stated below.

Pros:

- Large, mature and active community
- Support of open-source Eclipse and GNU-GCC development environment
- Availability at low price
- Cortex M0 processor with competitive specs
- Support in mbed, an online open source development platform
- Support of supplementary tools such as nRF-Sniffer and android application 'Master Control Panel'

Cons:

- The BLE stack available as binary reducing flexibility
- SDK not open for distribution

5.2 Porting PCA10000 platform of nrf51822 to Contiki*

5.2.1 Development Setup

Contiki was initially developed in a Linux based operating system, although Windows is also supported now. The porting of PCA10000 platform to Contiki was done in Ubuntu 13.10 and 14.04. The compiler suite used is GNU Tools for ARM Embedded Processors Version 4.8.3. The program used for programming the binary files compiled was SEGGER J-Link Commander V4.90 using the Segger JLink programmer on PCA10000. The front end used to write the code is the Eclipse IDE for C/C++ Developers Version Kepler Service Release 1 and Sublime Text.

*Available at `https://github.com/EarthLord/contiki` with complete Doxygen documentation

5.2.2 Folder structure of Contiki

When porting a new platform to Contiki, the three folders inside a Contiki distribution where additions need to be made are cpu, examples and platform. The content in these folders are as described below. The location of the Contiki distribution is represented by the path 'CONIKI'. The port of nrf51822 SoC with its PCA10000 board also follows this convention.

CONTIKI/cpu In this folder, all the files contain implementation which is solely dependent on the SoC is present. This includes the code for the processor abstraction, the drivers of the peripherals of the SoC, makefile with commands for compiling and linking the code, linker file and the documentation of these implementation. The boot-loader or start-up code, if required is also present here. The peripheral drivers for peripherals such as timers and serial port must use the API format of Contiki so that the libraries of Contiki using these peripherals can operate correctly. The exact implementations of these drivers is explained in section 5.2.3.

CONTIKI/platform An SoC over time has increasing number of boards or systems that are based on it and these are referred as platforms in this folder. Every board has its own folder in this platform folder which contains files which contain implementation which is dependant on the particular board. These include the specification of the connections to the LEDs, buttons, sensors and serial ports, power sources, the clock sources, memories present and any other board specific details. Any peripheral driver implementation specific to the board is present here. The default project specifications such as the source and frequency of clock and serial baud-rate are defined here. The main function where the execution of the program starts and initialization of all the peripherals and libraries used by Contiki happens is present here.

CONTIKI/examples The examples folder contains all the files related to projects that are implemented using Contiki with any of the supported hardware platforms. The makefile where the make command is executed is present here. The compiled object and binary files are also stored here. The default specifications of the platform can be overridden by creating a `project-conf.h` in the specific example.

5.2.3 Peripherals required for Contiki

Contiki clock Contiki clock, which is the source for all the timers, except the rtimer, is provided by the RTC1 peripheral of nrf51822. RTC1 is chosen because when a SoftDevice is used, RTC0 will not be accessible for the user application. The Real Time Clock (RTC) peripheral uses the Low Frequency Clock (LF-CLK) of nrf51822, which can be generated by a crystal or RC oscillator to produce 32.768 kHz. For Contiki clock, there are two implementations made, namely Tickless and Ticks which are described below.

Ticks For this implementation the RTC1 peripheral is configured such that an interrupt is called for its every increment or *tick*, hence the name. This happens at 64 Hz, which is the default value of `CLOCK_SECOND`. In the interrupt routine, the current clock tick is incremented, an etimer poll is requested if an etimer has expired and every `CLOCK_SECOND`th interrupt the second count is incremented. The variable storing the clock 'ticks' and 'seconds' is returned upon their request from any other process.

Tickless In this implementation, the processor will not be woken up at every tick of the RTC peripheral, hence the name. To enable this, a small addition is required to the etimer and clock module present at `CONTIKI/core/sys`. Every time the next etimer expiration is computed, the clock module is informed of it. With this additional information the RTC can configure a compare interrupt to poll the etimer upon its expiry. The RTC1 peripheral is also configured to provide an interrupt only on its overflow so that it can be accounted for when calculating the seconds elapsed. For the 24-bit timer of RTC1, with `CLOCK_SECOND` as 64, the overflow interrupt would happen only once every $(2^{24}/64)$ second, that is almost every three days as compared to interrupt happening few times a second in the 'Ticks' case.

With a Tickless implementation, both keeping track of the value of 'clock ticks' and polling the etimer upon its expiry is handled by the RTC peripheral rather than the processor, which would significantly reduce the power consumption without sacrificing any performance. The only additional development task is the addition in the core Contiki library as mentioned above.

Rtimer An rtimer is implemented in Contiki's port to nrf51822 by using the *TIMER1* peripheral, which runs on the High Frequency Clock (HF-CLK) of 16 MHz. TIMER1 was chosen as TIMER0 is required by SoftDevice, in case it is used. Since RTimer is required with higher granularity than Contiki Clock, rtimer increments at rate of 62.5 kHz in its current implementation, which is every 16 µs. This is achieved by TIMER1 is configured as a 8-bit timer providing an interrupt on overflow where the rtimer count is incremented. In the interrupt routine there is also a check to see if the scheduled rtimer task needs to be run by comparing the current rtimer count. It should be noted that Rtimer being run by TIMER1 peripheral requires the HF-CLK to be active, consuming power.

LEDs The PCA10000 board has a RGB LED unit on it. The port allows it to be controlled by the Contiki API. This is done through reading and writing to the General Purpose Input-Output (GPIO) port when the LED's status is read and LED's state is changed respectively. This implementation is present in `CONTIKI/platform/dev/led-arch.c` file since the LEDs are not a property of the nrf51822 SoC but the PCA10000 platform.

Button(s) The PCA10000 board does not have any buttons. In case of porting a platform with physical buttons, its implementation would be in the same location as the LEDs i.e. the platform folder.

Serial Port The port of Contiki to PCA10000 platform offers abstraction of the UART peripheral of nrf51822, which will communicate with the serial port of the computer to which PCA10000 is connected to. Writing to the serial port is achieved by using the `printf()` function present in the `stdio.h` library by redirecting `printf` call's character stream to the UART peripheral. Instead of the default NewLib library, the Newlib-nano library is used so that the amount of memory required is reduced.

For receiving the data from the the *serial-line* module of Contiki is used. This module broadcasts an event when a series of characters are received ending with a 'newline' (\n) character. To use this module, it is initialized on boot and all the characters received by the UART port is sent to this module in the UART receive interrupt routine.

Radio Since the radio peripheral is completely controlled by the SoftDevice for implementing the BLE stack, it is untouched in this port. To use the radio peripheral, the APIs provided by the SoftDevice is used.

5.2.4 Makefile structure of Contiki

Introduction to makefile and include. There are multiple makefiles present across different folders that are included in the way shown in figure 5.2 to form the complete makefile. In the most basic form these are the makefile in the example folder, 'makefile.include' in the Contiki root folder, 'makefile.TARGET' in the specific platform folder and 'makefile.CPU' in the specific cpu folder, where TARGET and CPU are specific to the project. The port of Contiki to nrf51822's platform follows this convention too.

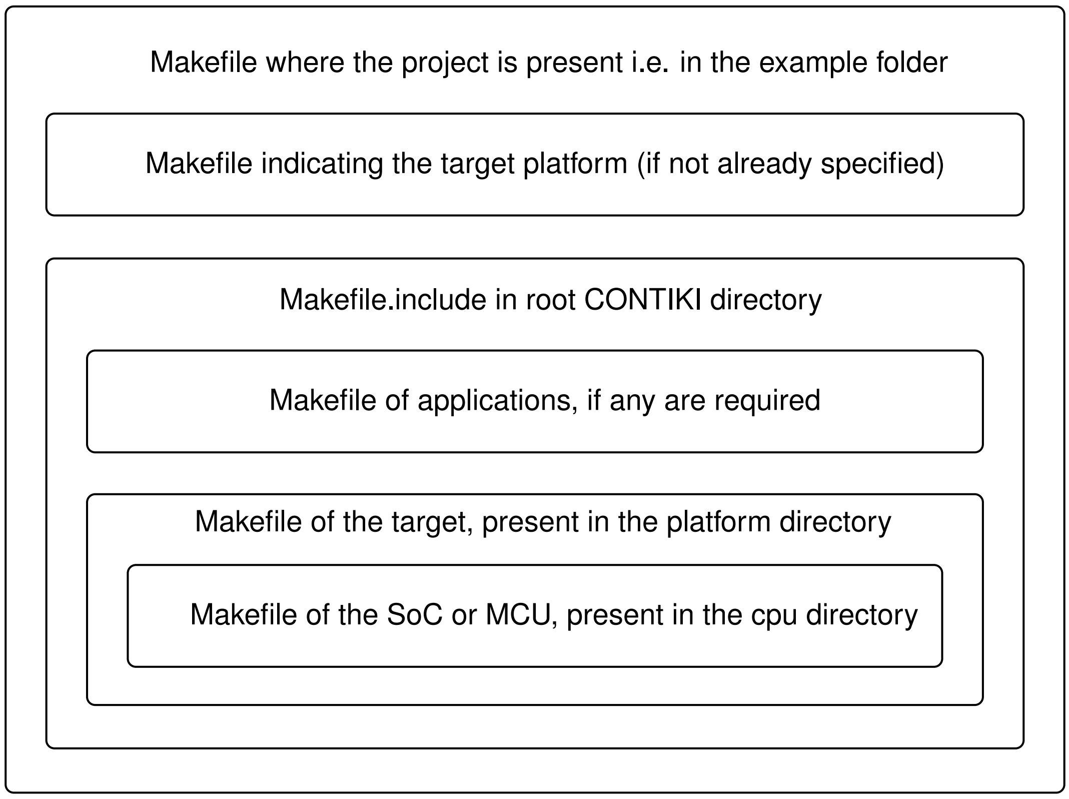

Makefile where the project is present i.e. in the example folder
Makefile indicating the target platform (if not already specified)
Makefile.include in root CONTIKI directory
Makefile of applications, if any are required
Makefile of the target, present in the platform directory
Makefile of the SoC or MCU, present in the cpu directory

Figure 5.2: Structure of Makefile inclusion in Contiki OS to form an example specific one

The makefile in the project specific example folder, the different project source files are specified and the 'makefile.include' present in the root CONTIKI folder is added. 'makefile.inlcude' glues together all the required components of Contiki and provides the default implementation of compiling and linking. The target makefile in the specific platform directory is included here. In 'makefile.TARGET' the source files present in this directory are added, any Contiki modules required are added and the SoC or MCU makefile in the specific directory is added. The SoC or MCU makefile adds all the source files present in the CPU directory, specifies the command for compiling the code, linking the objects generated, uploading the executable binary or hex file.

The makefile in the example folder is where the make command is called. The operations that can be performed with this make command depends on the makefile. Usually the operations are cleaning (removing the object and binary files), compiling the source files into object files , linking these object files to create an executable file, creating a binary file from this executable file, uploading the binary to the SoC to start execution and so on. For the nrf51822 SoC additional operations of uploading the SoftDevice and erasing the flash are supported.

6 | Test Cases

This chapter details the reasoning and design of the test cases used in this thesis. To provide a common platform for comparing BLE and 802.15.4, Contiki was ported to a platform supporting BLE as described in chapter 5. This chapter starts off with section 6.1 explaining all the metrics measured in the experiments performed. Section 6.2 follows by explaining the test platforms and tools used. Section 6.3 and 6.4 explains the two test suite designed to measure the metrics defined.

6.1 Performance Metrics Definition

The aim of this thesis is to compare the key characteristics of the link layer of BLE and 802.15.4. Note that as explained in **4.background**, 802.15.4 here means the physical layer of 802.15.4 standard. The 802.15.4 standard MAC later is not used, rather Null-RDC and ContikiMAC, both with CSMA will be used. In this section the four key performance metrics measured for both the wireless protocols are defined in the context of this thesis.

Data rate Data rate at the link layer is defined as the maximum amount of data transmitted per second for a protocol with packets containing the maximum allowed payload for the link layer. The data rate is expressed as both kilobits per second and packets per second.

Latency In this test the aim is to measure the effect of the link layer architecture and configuration on the latency observed by an layer on top when requesting data from another node. Latency is defined as the time from which the layer(s) above the link layer provides it the packet which asks for data from another node to the time the data from the other node is received. This is better understood by a generic illustration in figure 6.1.

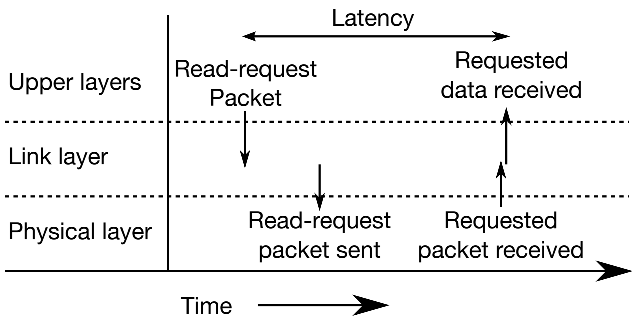


Figure 6.1: Illustration of latency measurement

Reliability Two criteria will be used for measuring reliability when two nodes are communicating. Packet Reception Ratio (PRR) is defined as the ratio of the number of packets successfully received to the number of packets sent, both by the link layer. Packet Delivery Ratio (PDR) is defined as the ratio of the number of packets delivered to the number of packets sent, by the layer above the link layer in both the sender and receiver. While PRR measures the reliability of transfer of a single transfer of a packet, PDR measures the reliability of the link layer to communicate a packet as seen by the layer above it.

Energy Consumption The energy consumption in this project is indirectly measured as the Radio Duty Cycle (RDC), which is the ratio of the time for which the radio is switched on to the total time of the experiment. Measuring the energy consumption as RDC for different scenarios will be more useful to developers as this information will change little across platforms for the same protocol, whereas the actual energy consumed is reducing with every new platform introduced in the market. The processor time utilized by the protocol stack is not considered as this information could not be extracted from the SoftDevice APIs and also because of the different processor types in the two platforms (32-bit vs 16 bit).

6.2 Test setup

6.2.1 BLE platform

The platform used for BLE is PCA10000, which is described in section 5.1.3. Since BLE is has an asymmetric architecture, a master device is required to communicate with a slave device. In all the tests the slave device is a PCA10000 board running on Contiki and peripheral BLE stack provided by the SoftDevice *S110*. Depending on the test, the master device varied between either a PCA10000 board or an Android device. When a PCA10000 was used as master device, it ran on bare-metal code with central BLE stack provided by the SoftDevice *S120*. Nexus 7 and Nexus 4, both at Android version 4.4.4 were the Android devices used for performing certain tests.

Communication will be done at the Generic Attribute (GATT) layer because the binary from Nordic Semiconductor used in this project does not provide APIs to access the lower layers in both peripheral and central devices, hence evaluation at the link layer will be done indirectly based on the inferences from the collected data.

6.2.2 802.15.4 platform

Tmote-Sky platform, which is widely used and supports all the features of Contiki was used as the platform to test 802.15.4 performance. Shown in figure 6.2, Tmote-Sky is based on MSP430-F1611 MCU and CC2420 2.4 GHz transceiver. It has a USB to serial converter on board to allow the MSP430 MCU to communicate with the serial port of a computer. It has additional sensors, buttons and memory which are not used in this project.

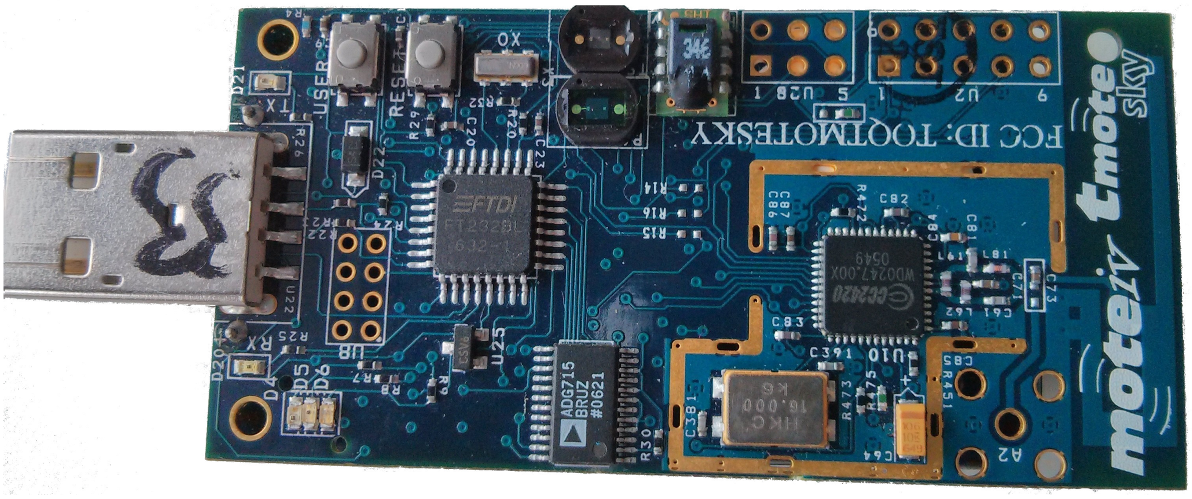


Figure 6.2: TmoteSky platform

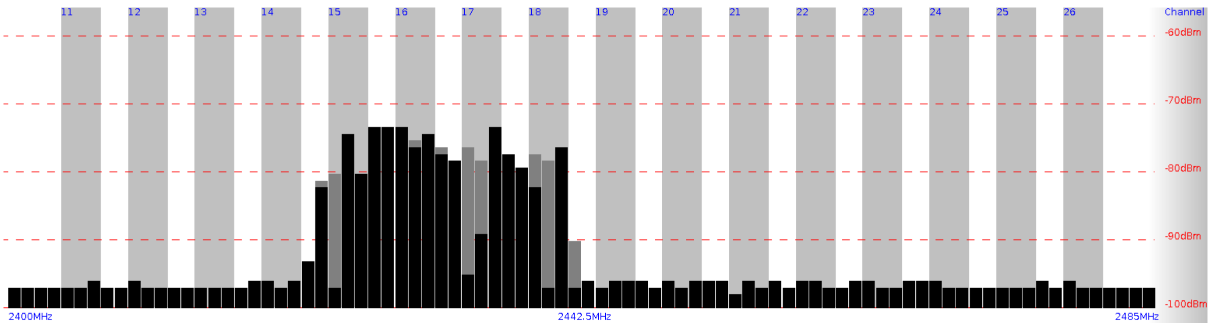
6.2.3 WiFi Interference

In some of the tests described in the following sections, WiFi interference needed to be created to test the reliability of the wireless protocols with external interference. A open-source tool called `iperf` was used for creating this interference by sending User Datagram Protocol (UDP) packets to the IP address of a router. The computer and router used are ThinkPad T420 and Netgear N300 router respectively, both supporting WiFi 802.11b/g/n standard. The aim of the interference pattern is to simulate streaming of large amount of data over WiFi. This was achieved by this command:

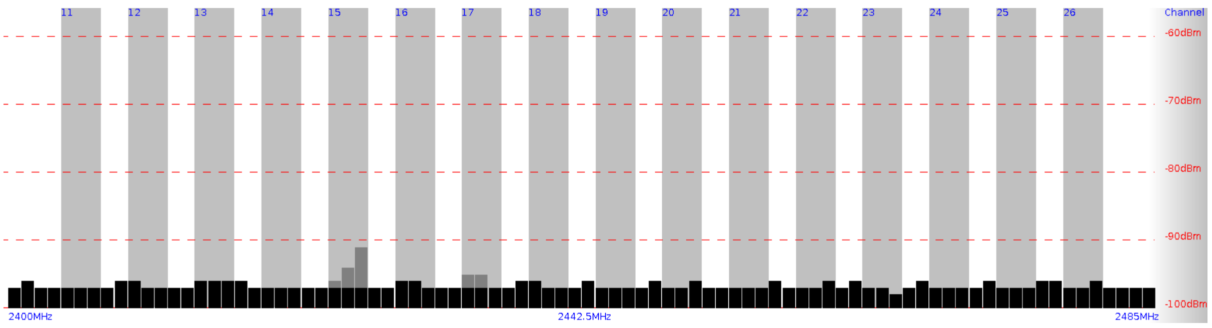
```
sudo iperf -c 192.168.1.1 -u -P 1 -i 1 -p 5001 -f M -b 54.0M -t 64 -T 1
```

Where '-c 192.168.1.1' specifies that the packets are sent to the IP address specified (router's IP), '-u' to use UDP packets, '-P 1' specifies one client thread to run, '-i 1' specifies the bandwidth reporting interval as one second, '-p 5001' specifies the server port to connect to as 5001, '-f M' specifies the reporting format as MegaByte per second, '-b 54.0M' specifies the data rate to be used as 54 Megabit per second, '-t 64' specifies the transmission to happen for 64 second and '-T 1' specifies that only one hop is allowed to reach destination. Although the data rate was specified as 54 Mbps i.e. 6.75 MBps, the actual data rate of transmission reported was around 3 MBps.

To verify that the interference was created and to know which frequency range was occupied, the 'rssi-scanner' project was used*. This project is based on Tmote-Sky platform running on Contiki. The screen-shot of rssi-scanner shows a snapshot of the environment where the test were conducted for the cases of with and without interference in figure 6.3a and 6.3b respectively, where y-axis shows the signal strength measured and x-axis is the frequency of the measurement.



(a) Environment while running iperf tool to create interference



(b) Environment without creating interference

Figure 6.3: 2.4 GHz environment captured by rssi-scanner

*Available at: http://sourceforge.net/p/contikiprojects/code/HEAD/tree/sics.se/rssi-scanner/

6.3 High-Throughput (HT) Test Design

High-Throughput (HT) test aims to measure the maximum possible data rate of BLE and 802.15.4 at the LL with and without WiFi interference. For both protocols, the link layer contains the maximum payload allowed. Each run of the test will last for one minute to collect enough data to calculate the mean data rate. In each of the tests, the amount of time the radio is switched on (RDC), either transmitting or receiving, is logged to measure the energy consumption. The metrics measured in this test are data rate, reliability and energy consumption. The two nodes were kept one metre apart and in case of environment with interference, the router to which the UDP packets were sent over WiFi was about 3 metre from these nodes.

BLE As shown in figure 6.4, the data rate is measured by sending data from the server to the client using *Notifications*, which are packets which are not acknowledged in the ATT layer. Since the link layer is not accessible with the SoftDevice used, a 20 byte payload was sent at the GATT layer as notification so that 27 byte data for the link layer will be achieved. The data rate is measured at the receiver i.e the Master device.

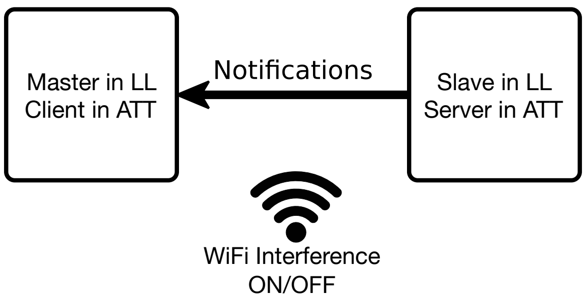


Figure 6.4: Setup to measure data rate with BLE

The number of packets sent and received at the ATT layer in the two nodes are logged to calculate the PDR. Also by logging the number of times the radio was switched on, the number of packets actually transmitted can be inferred, by which the PRR can be calculated.

BLE specification tells that a master device must change the channel map of a connection according the 2.4 GHz environment so as to not face any interference. This is the *adaptive* part of AFH, which requires the master device to have information about the activity in the frequency spectrum used by BLE . S120 SoftDevice used for the central BLE stack by default uses the complete channel map and has an option for configuring it manually, but the channel map does not change automatically based on external interference. So in the tests without WiFi interference, the channel map consisted of all the channels. When WiFi interference was introduced, one test had the complete channel map and another had a *WiFi free* map of channels (0 to 6) and (20 to 36), which is 2404 to 2416 MHZ and 2446 to 2478 MHz. This map was based on the interference noticed in figure 6.3a.

802.15.4 In case of 802.15.4, Null-RDC layer was modified to send packets with maximum payload size as soon as possible. One node is configured as transmitter and the other as receiver, with the data rate measured at the receiver. A payload of 110 byte was necessary for the link layer to achieve the maximum 127 byte packet size.

Three tests were designed to measure the data rate achievable with 802.15.4. In the first test without WiFi interference, channel 26 (2480 MHz) of 802.15.4 was used. In the second test with WiFi interference, channel 15 (2425 MHz) of 802.15.4 was used. In the third test, the Clear Channel Assessment (CCA) was disabled, so that the transmitting node ignores the external environment.

The various test cases conducted in the HT test suite are

- sky_sky_N_!CCA
- sky_sky_Y_26
- sky_sky_Y_15
- nrf_nrf_N_all
- nrf_nrf_Y_all
- nrf_nrf_Y_!Wifi
- nrf_N4_N
- nrf_N7_N

The legend of the naming of the test cases is

`{source}_{destination}_{presence of WiFi interference(Y/N)}_{other conditions}`

Where, sky = Tmote-Sky; nrf = nrf51822; N4 = Nexus4; N7 = Nexus7; !CCA = CCA disabled; 26,15 = channel used; all = All 40 BLE channels; !WiFi = WiFi free channels in BLE

6.4 Request-Response (RR) Test Design

The Request-Response (RR) test aims to determine the latency for reading data from another node as explained in the latency paragraph in section 6.1. In this tests also the time the radio is switched on is logged in both the nodes to calculate RDC to interpret the energy consumption. The idea is to find the relation between energy consumption and latency based on different link layer configurations. Thus, the metrics evaluated in this test are latency and energy consumption.

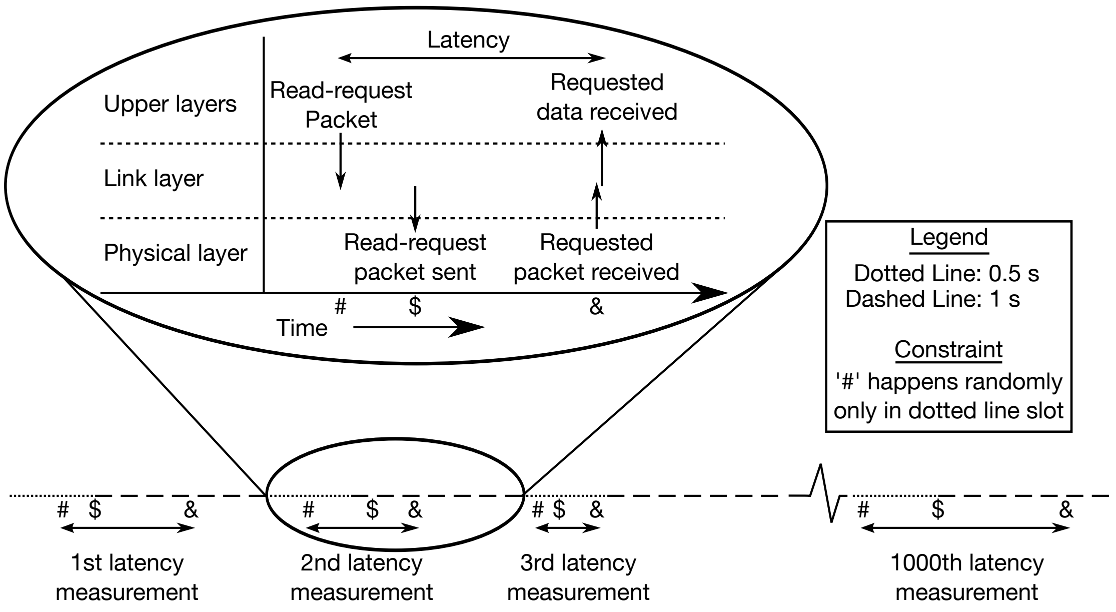
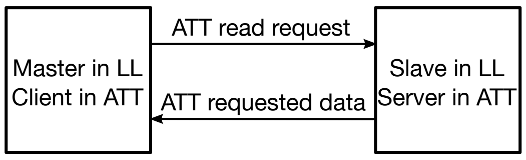


Figure 6.5: Illustration of the RR Test operation

The design of the test for both the protocols is shown in a generic diagram in figure 6.5. A latency measurement is done in every 1.5 second period, repeated thousand times. The link layer will be sent a ‘read-request’ packet from its upper layer after a random time in the first 0.5 second slot of this 1.5 second period, represented by ‘#’ in figure 6.5. This randomization of the issuing of the read request (‘#’ event) ensures that is there is any periodic communication between the nodes, the thousand measurements of latency will provide the range of values that can expected with a particular scenario along with the power consumption. As explained in the latency paragraph in section 6.1, the latency measurement is the difference of the time between the ‘&’ and ‘#’ events shown in figure 6.5.

BLE In case of BLE , the use case used for the latency test is a central device which is master at link layer and client at ATT layer gets data from a peripheral device which is slave at link layer and server at ATT. This can be achieved in two modes as shown in figure 6.6a and 6.6b. In both cases it is the link ATT client which gets the data from the server. In the first case, the client asks for the data from the server, which responds with data. The latency measurement for this interaction happens at the client. In the second case, the server sends the data in an ‘indication’ packet, which is acknowledged by the client. In this case since it is the server that is getting the response from the master, the latency measurement happens in the server.



(a) Read-response test case

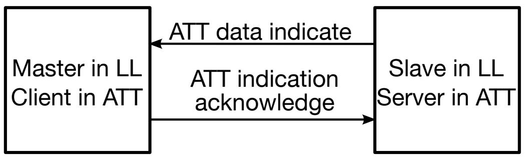

ATT data indicate
Master in LL Client in ATT
ATT indication acknowledge
Slave in LL Server in ATT
(b) Indicate-acknowledge test case

Figure 6.6: Test modes for measuring latency using BLE

In all the tests, the complete payload of 20 bytes for the ATT layer was used, which corresponds to 27 byte in the link layer. Two connection intervals of 7.5 ms and 125 ms were used for the tests. The 'supervision timeout' for all the tests was 1.5 second. For each 'connection interval', tests were conducted with two different 'slave latency' parameter. The table 6.1 shows these values and the configuration for all the test cases in RR test using BLE with the intent to find the effect of the link layer configuration on the latency. By using different 'slave latency' values, the energy consumption of the asymmetric connecting devices can be evaluated.

Test Case	Connection Interval	Slave Latency	Test Mode
7.5_0_r	7.5 ms	0	Read
7.5_0_i	7.5 ms	0	Indicate
7.5_65_r	7.5 ms	65	Read
7.5_65_i	7.5 ms	65	Indicate
125_0_r	125 ms	0	Read
125_0_i	125 ms	0	Indicate
125_3_r	125 ms	3	Read
125_3_i	125 ms	3	Indicate

Table 6.1: List of RR test cases using BLE

802.15.4 In the case of 802.15.4, one node is configured to send unicast messages and another is configured to receive these and send a response. The latency using both ContikiMAC and NullRDC will be measured. ContikiMAC uses the default 125 ms wake up interval for the receiver. For payload size is same as in the test cases with BLE for comparing the latency for the same payload. The BLE test cases are designed such that the 7.5 ms tests can compare with Null-RDC test of 802.15.4 while the 125 ms tests can compare with the ContikiMAC test of 802.15.4.

The sending of the unicast packet and getting its response followed the same thousand iterations of 1.5 s intervals containing a 0.5 s period where the packet was randomly requested to be sent. The latency measurement happens as illustrated by figure 6.1 used to define the term latency in section 6.1.

7 | Results and Analysis

This chapter provides a graphical representation of the data acquired from conducting the tests designed in the last chapter. The raw data used to generate this graphs can be accessed in Appendix B. **5.More overview**

7.1 High-Throughput Test

7.1.1 Graphical representation of the data acquired

The data acquired from the different test cases of the HT test suite is represented graphically below. Figure 7.1, 7.2 and 7.3 provides information about the measurement of data rate, reliability and energy consumption respectively. The legend used for naming the test cases is same as described in section 6.3.

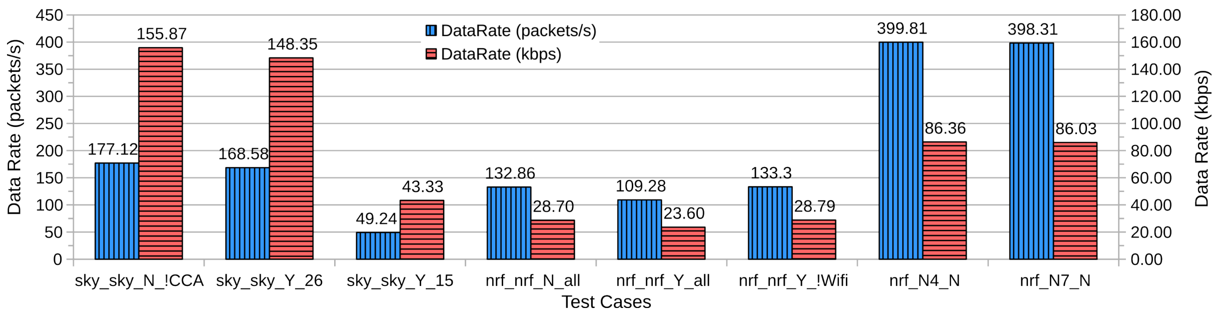


Figure 7.1: Data Rate

Note that in figure 7.1 there are two Y-axis representing the data rate in packets per second and kilobit per second. The link layer payload of 27 bytes and 110 byte was used for BLE and 802.15.4 respectively to calculate the data rate in kbps.

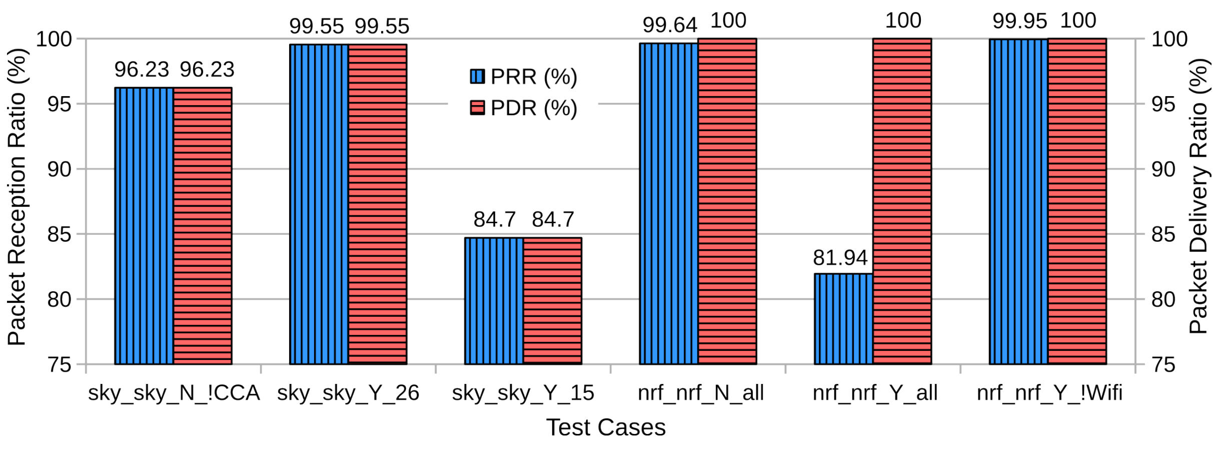


Figure 7.2: Reliability

The graph in figure 7.2 has Y-axes from 75 to 100 % for clearer representation of the PDR and PRR 6.Is this misleading?. As mentioned in section 6.3 the reliability was calculate in BLE by logging the number of times the radio was switched on as this information was accessible from the SoftDevice. This allowed the calculation of PRR with one packet per connection event. Since the android devices can communicate multiple packets per connection event, the PRR could not be calculated. Hence, those test cases are not plotted.

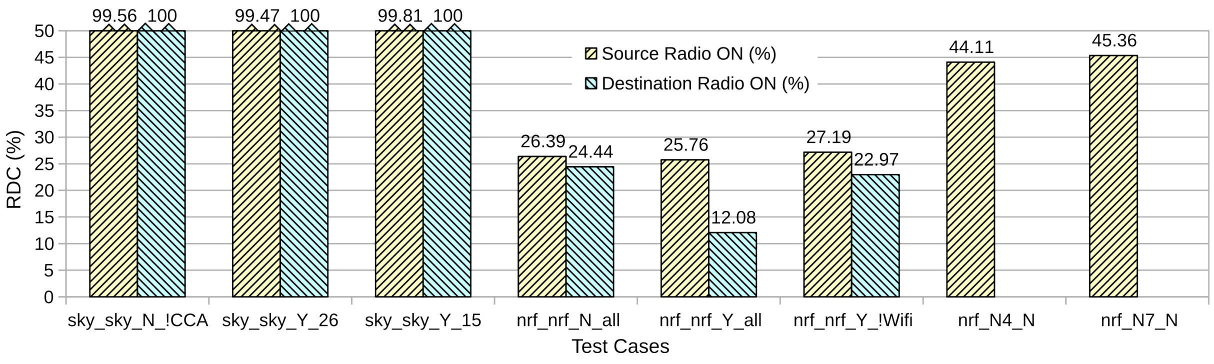


Figure 7.3: Energy Consumption

For a clearer representation of the graph, the y-axis in figure 7.3 is limited to 50%. The break in the graph shows that when Tmote-Sky uses 100% of radio duty cycle.

7.1.2 Analysis

Data rate The absolute data rate is higher in 802.15.4 than BLE although the number of packets transmitter per second is higher in BLE as seen in figure 7.1. This is because of the higher packet size of 802.15.4, even though the bit rate of transmission of BLE is four time the bit rate of 802.15.4 at 1 Mbps.

For 802.15.4 the peak data rate of 155.87 kbps is achieved when CCA is not used. The more practical approach of using CCA decreased in the data rate slightly to 148.35 kbps. When WiFi interference was introduced, most of the transmission slots were not utilized because of backing off by CCA. Appendix B shows that only 21% of the transmission slots were used dropping the data rate considerably. With this 43.33 kbps data rate was achieved, which is higher than the data rate achieved by BLE communication between nodes of PCA10000 platform. Since 802.15.4's NullRDC tries to send a packet as soon as possible, the inference is that the WiFi traffic was not present 43.33/148.35 i.e. 29.2% of the time.

The SoftDevice used to implement the central stack allows only one packet to be communicated per connection interval. With this the theoretical data rate that can be achieved can be found out by

$$\text{Data Rate (packet/second)} = \frac{1}{\text{connection interval}} = \frac{1}{7.5ms} = 133.33\, packet/second$$

$$\text{Data Rate (kbps)} = \frac{\text{(bits per byte)}\times\text{(link layer payload in bytes)}}{\text{connection interval}} = \frac{8\times 27}{7.5ms} = 28.8\, kbps$$

The experimental results seen in figure 7.1 shows this is achieved accurately. When WiFi is introduced, the data rate drops from 28.7 kbps to 23.6 kbps, when all the channels are used. This is 82% of the WiFi free data rate. Figure 6.3a shows that 10 out of the 37 data channels are interfered by WiFi traffic. This means that the data rate should reduce by (37-10)/37, which is about 73% of the WiFi free data rate. The greater data rate achieved experimentally can be accounted to the fact that the WiFi traffic wasn't interfering 100% of the time as inferred. Considering that the 10 channels were interfered 29.2% of the time as calculated, the data rate achievable can be calculated as $(0.292 \times 10/37 + 1 \times 27/37) \times 28.8 = 24.1$ kbps which is closer to the achieved 23.6 kbps.

When the WiFi free channels were used the data rate recorded was close to the theoretical data rate predicted. There was no influence of the WiFi traffic at all for BLE communication in this case.

Communication of the PCA10000 with a Nexus4/7 master device achieved a greater data rate of about 86 kbps or 400 packet/second. This is because the BLE stack in these Android devices support communication of multiple packets in a connection event, increasing the data rate achievable. With a connection interval of 7.5 ms, at an average there were around $(400 \times 7.5/1000)$ i.e. 3 packets of data received by the Android device per connection event.

Reliability In case of 802.15.4, the PDR and PRR is same since it does not have any communication failure detection and retransmission mechanism. This is left to the upper layer to handle. The link layer just tries initiate communication when there is no external interference present with its CCA. Figure 7.2 shows that the PRR achieved when CCA is switched off is 96.23%. While it can be seen that without CCA the data-rate is higher, the reliability decreases. By using CCA in a WiFi free channel, the PRR increases to 99.55%, which means that the CCA is effective in case of less interference. When there was heavy WiFi traffic in the same channel of communication, the CCA did work since only 21.11% of the transmission slots were used. Even then the PRR was reduced to 84.7%. This could happen when the interference starts after the carrier assessment, causing the packet to get corrupted. Here it can seen that CCA of 802.15.4 has reduced effectiveness in case of heavy interference.

In case of BLE, figure 7.2 shows that the PDR is always 100% since BLE employs a simple acknowledgment scheme to detect if the communication has not been successful so that the packet can be retransmitted. This ensures that the upper layers can safely assume that the link layer is completely reliable. Without WiFi interference PRR was about 99.64%, which means that there had to few retransmissions. With all the channels employed and WiFi interference, PRR reduced to 81.94%. This number is high because of the frequency hopping used by BLE, which ensures that interference in one frequency range has little effect on the communication happening over 37 data channels over the 2.4 GHz ISM band. This shows that the frequency hopping when used over the complete channel map helps in sustaining the communication when there is external interference. When only the WiFi free channels were employed, there was no influence of the WiFi interference on PRR which was 99.95%. This shows that when a master device has the capability of detecting external interference, it can adapt the frequency hopping channels so that there is negligible effect on the communication. Note that AFH can only help negate narrow band interference in the 2.4 GHz ISM band.

Energy Consumption

7.2 Request-Response Test

7.3 What we learnt

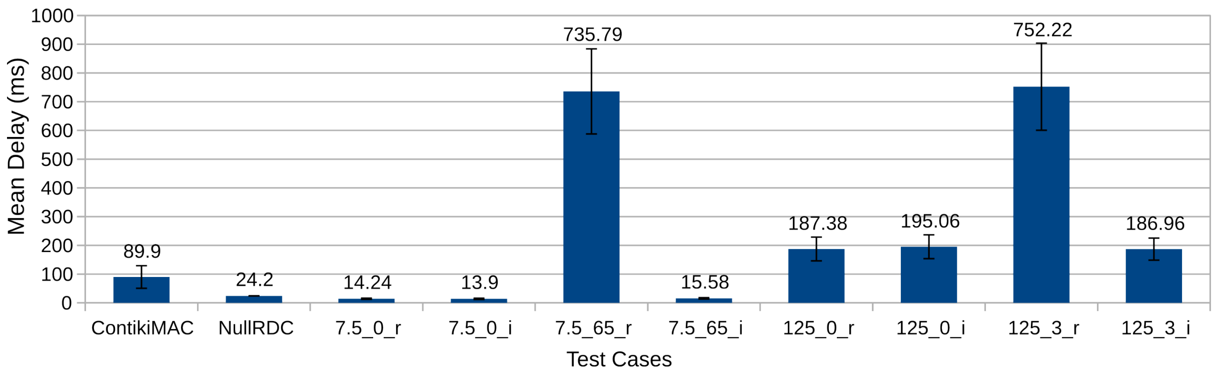


Figure 7.4: Latency

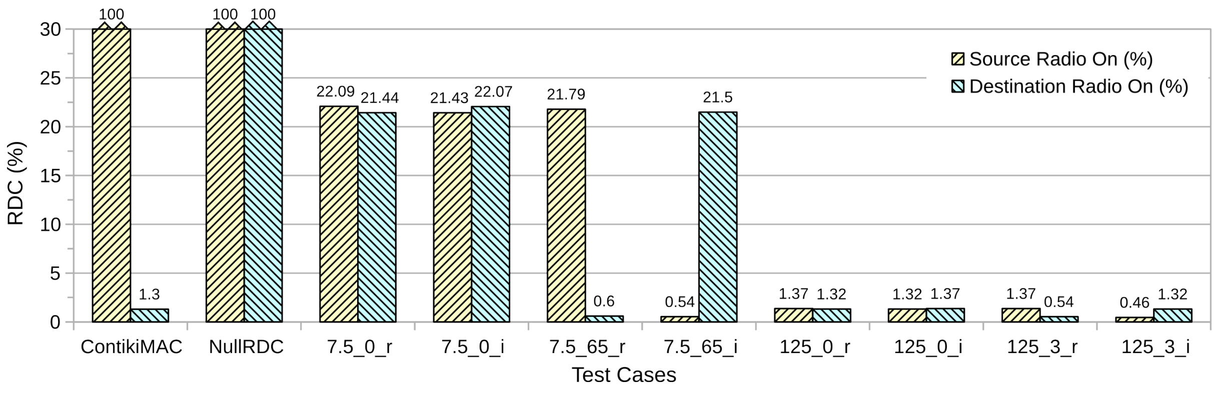


Figure 7.5: Energy Consumption

8 | Other Contributions

8.1 Firmware for demo application

8.2 Radio driver for BLE advertisement packet logger

9 | Conclusion and Future Work

References

[1] Carles Gomez, Joaquim Oller, and Josep Paradells. “Overview and evaluation of bluetooth low energy: an emerging low-power wireless technology.” In: *Sensors (Basel, Switzerland)* 12.9 (Jan. 2012), pp. 11734–53.

[2] *Contiki: The Open Source Operating System for the Internet of Things*. URL: `http://contiki-os.org/`.

[3] Robin Heydon. *Bluetooth Low Energy: The Developer’s Handbook*. 1st ed. Prentice Hall, 2012, p. 368.

[4] *Contiki Hardware*. URL: `http://contiki-os.org/hardware.html`.

[5] George Oikonomou and Iain Phillips. “Experiences from porting the Contiki operating system to a popular hardware platform”. English. In: *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*. IEEE, June 2011, pp. 1–6.

[6] Alexandru Stan. “Porting the Core of the Contiki operating system to the TelosB and MicaZ platforms”. Bachelor Thesis. International University Bremen, 2007.

[7] Adriana Wilde, Richard Oliver, and Ed Zaluska. “Developing a low-cost general-purpose device for the Internet of Things”. English. In: *2013 Seventh International Conference on Sensing Technology (ICST)*. IEEE, Dec. 2013, pp. 490–494.

[8] Zoltan Padrah, Ales Verbic, Marko Mihelin, et al. “Connecting Contiki enabled Versatile Sensor Nodes via CC1101 radio”. In: *NEWCOM++* (2011).

[9] Voravit Tanyingyong, Robert Olsson, Markus Hidell, et al. “Design and Implementation of an IoT-controlled DC-DC Converter”. eng. In: *2013 Sustainable Internet and ICT for Sustainability, SustainIT 2013* (2013), p. 6685199.

[10] Carles Gomez, Ilker Demirkol, and Josep Paradells. “Modeling the Maximum Throughput of Bluetooth Low Energy in an Error-Prone Link”. In: *IEEE Communications Letters* 15.11 (Nov. 2011), pp. 1187–1189.

[11] Jia Liu and Canfeng Chen. “Energy analysis of neighbor discovery in Bluetooth Low Energy networks”. In: *Nokia.(nd)* (2012).

[12] Elke Mackensen, Matthias Lai, and Thomas M. Wendt. “Performance analysis of an Bluetooth Low Energy sensor system”. In: *2012 IEEE 1st International Symposium on Wireless Systems (IDAACS-SWS)*. September. IEEE, Sept. 2012, pp. 62–66.

[13] Philipp Kindt, Daniel Yunge, Robert Diemer, et al. “Precise Energy Modeling for the Bluetooth Low Energy Protocol”. In: *CoRR* abs/1403.2 (Mar. 2014). arXiv: `1403.2919`.

[14] Mikko Savolainen. *Throughput with Bluetooth Smart technology : Bluegiga Technologies*. 2013. URL: `https://bluegiga.zendesk.com/entries/24646818-Throughput-with-Bluetooth-Smart-technology`.

[15] Adam Dunkels. “The ContikiMAC Radio Duty Cycling Protocol”. In: (Dec. 2011).

[16] Mathieu Michel and Bruno Quoitin. “Technical Report : ContikiMAC vs X-MAC performance analysis”. In: *arXiv preprint arXiv:1404.3589* (Apr. 2014). arXiv: `1404.3589`.

[17] M.-P. Uwase, M. Bezunartea, T.L. Nguyen, et al. “Experimental evaluation of message latency and power usage in WSNs”. In: *2014 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. IEEE, May 2014, pp. 69–72.

[18] Matti Siekkinen, Markus Hiienkari, Jukka K Nurminen, et al. “How low energy is bluetooth low energy? Comparative measurements with ZigBee/802.15.4”. In: *2012 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, Apr. 2012, pp. 232–237.

[19] Konstantin Mikhaylov, Nikolaos Plevritakis, and Jouni Tervonen. “Performance Analysis and Comparison of Bluetooth Low Energy with IEEE 802.15.4 and SimpliciTI”. In: *Journal of Sensor and Actuator Networks* 2.3 (Aug. 2013), pp. 589–613.

[20] Artem Dementyev, Steve Hodges, Stuart Taylor, et al. “Power consumption analysis of Bluetooth Low Energy, ZigBee and ANT sensor nodes in a cyclic sleep scenario”. English. In: *2013 IEEE International Wireless Symposium (IWS)*. IEEE, Apr. 2013, pp. 1–4.

[21] ZigBee Alliance. *ZigBee Specification*. URL: `http://www.zigbee.org/Specifications.aspx`.

[22] Nordic Semiconductor. *nRF51822*. URL: `https://www.nordicsemi.com/eng/Products/Bluetooth-R-low-energy/nRF51822`.

[23] PrithviRaj Narendra. *Introduction to PCA10000 and using JLink Software with it in Ubuntu*. 2014. URL: `http://rumblingsofearthlord.wordpress.com/2014/01/22/introduction-to-pca10000-and-using-jlink-software-with-it-in-ubuntu/`.

A | Appendix | BLE SoC Comparison

The dense table in the next page compares many different parameters of the BLE SoCs available. This table was compiled in the first half of 2014. Because of the high pace at which the industry around BLE is progressing, many details here might not be accurate as time progresses.

The legend for the table is

NA	: The information is not available
-	: The feature is not available

Platform	Unit	TI CC2540/1	CSR1001/ CSR1000	NordicSemi nRF51822	DialogSemi DA14580	Broadcom BCM20732	Quintic 9020	Lapis ML7105
Processor		8051	16 bit proprietary	Arm Cortex M0	Arm Cortex M0	Arm Cortex M3	Arm Cortex M0	Arm Cortex M0
Flash size	kB	128/256	64 (ROM)	128/256	32 (OTP)	NA	NA	64
RAM size	kB	8	64	16	42K SRAM & 8K Retention RAM	NA	NA	12 User code, 16 Data
RX Sensitivity	dBm	-93	-92.5	-93	-93	NA	-95	-85
TX Max Power		4	7.5	4	NA	NA	NA	3
GPIO		21	32	31	32	NA	31	
ADC		12 bit, 8 chl, up to 30 kHz	10 bit, 3 ch, up to 700 Hz	10 bit, 8 ch, upto 14 kHz	10 bit	Yes	Yes	-
I2C		1, if not USB	1	2	1	1	Yes	1
SPI		-	1	2	1	1	Yes	1
UART		2	1	1	1	1	Yes	1
USB		Full-Speed	-	-	-	-	-	-
DMA		5 Channel	1 channel	16 PPI,No DMA	NA	NA		-
Timer 32 bit		-	-	1				
Timer 16 bit		1	-	2	2+wakeup timer			
Timer 8 bit		2	-	-				
Other peripherals		OPAMP, Comparator, WDT, AES Encryption	AES, LED PWM	AES, WDT, Internal DC/DC regulator, RNG QDEC, 4 PWM channels,	AES, WDT, Internal DC/DC regulator, QDEC	DAC	DC/DC Regulator	AES, WDT
Active RX	mA	19.6	16	13	3.8	NA	8	9
Active TX at 0 dBm	mA	27	16	10.5	3.8	NA	8	9
Idle	mA		1	275 uA/MHz				3
Processor Off	uA	235						
RTC Timer on	uA	0.9	1.5	2.3 uA	0.6			2.9
External Interrupt	uA	0.4	0.4	0.5	0.6			0.7
Package		40 pin QFN	56 pin QFN	48 pin QFN	48&40 pin QFN, 34 WLCSP	32 pin QFN		WQFN32
BLE stack		Free, from TI	From CSR	From NordicSemi	Integratedin ROM	Yes, already in ROM		
Support Forum		TI E2E Community		Nordic DevZone	support.dialog-semiconudctor.com/			
IDE to use		IAR Studio	Proprietary IDE	Keil, Eclipse+GCC	Keil	NA		
Development Board		CC2540DK, CC2540EMK	DK-CSR1000-10048	nRF51822-EK	DA14580 Motherboard + WCSP or QFN daughterboards	BCM920732_BLE_KIT		
Price of Dev Board		101	300	108	NA	166		
Price of IC (single IC)	$	5.62	3.1	3.84	NA	NA	NA	NA
Price of IC (bulk quantiy)	$	2.59	NA	1.91	NA	NA	NA	NA

B | Appendix | Data Acquired

The entire data acquired and results computed for both High-Throughput test and Request-Response test can be found in the tables in the following pages.

High-Throughput Test

Test Cases	**sky_sky_N_!CCA**	**sky_sky_Y_26**	**sky_sky_Y_15**	**nrf_nrf_N_all**	**nrf_nrf_Y_all**	**nrf_nrf_Y_!Wifi**	**nrf_N4_N**	**nrf_N7_N**
Source	Tmote-Sky	Tmote-Sky	Tmote-Sky	PCA10000	PCA10000	PCA10000	PCA10000	PCA10000
Destination	Tmote-Sky	Tmote-Sky	Tmote-Sky	PCA10000	PCA10000	PCA10000	Nexus4	Nexus7
WiFi	No	No	Yes	No	Yes	Yes	No	No
Condition	No CCA	CCA on, ch 26	CCA on, ch 15		All channels	Wifi free channels	Wifi OFF in device	Wifi OFF in device
DataRate (packets/s)	177.12	168.58	49.24	132.86	109.28	133.3	399.81	398.31
DataRate (kbps)	155.87	148.35	43.33	28.70	23.60	28.79	86.36	86.03
Conn Interval (ms)	NA	NA	NA	7.5	7.5	7.5	7.5	7.5
TX slots	11008	10496	15360	7998	7997	7998	8006	8004
Packets sent	11008	10496	3243	7969	6553	7994	23997	23907
Packets received	10593	10449	2747	7969	6553	7994	23997	23907
PRR (%)	96.23	99.55	84.7	99.64	81.94	99.95	NA	NA
PDR (%)	96.23	99.55	84.7	100	100	100	100	100
TX slot utilization (%)	100	100	21.11	99.64	81.94	99.95	NA	NA
Source RDC (%)	99.56	99.47	99.81	26.39	25.76	27.19	44.11	45.36
Destination RDC (%)	100	100	100	24.44	12.08	22.97	NA	NA
Packet size (Bytes)	110	110	110	27	27	27	27	27

Request-Response Test

Cases	ContikiMAC	NullRDC	7.5_0_r	7.5_0_i	7.5_65_r	7.5_65_i	125_0_r	125_0_i	125_3_r	125_3_i
Mean delay (ms)	89.9	24.2	14.24	13.9	735.79	15.58	187.38	195.06	752.22	186.96
Stan Dev (σ) (ms)	39.3	0.2	2.02	2.23	147.93	2.44	41.52	41.44	151.45	38.42
Max delay (ms)	159.7	25.4	17.19	24.08	1077.96	19.11	399.92	376.54	1431.17	273.71
Min delay (ms)	39.8	23.7	10.54	10.15	498.04	12.11	128.41	127.7	128.33	137.04
Source Radio On (%)	100	100	22.09	21.43	21.79	0.54	1.37	1.32	1.37	0.46
Destination Radio On (%)	1.3	100	21.44	22.07	0.6	21.5	1.32	1.37	0.54	1.32
Slave Latency (count)	NA	NA	0	0	65	65	0	0	3	3
Connection Interval (ms)	NA	NA	7.5	7.5	7.5	7.5	125	125	125	125
Supervision Timeout (ms)	NA	NA	1250	1250	1250	1250	1250	1250	1250	1250