Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

# Simulation Models for the Performance Analysis of Bluetooth Low Energy In Multi-device Environment

Hariharan Mari

|  |  |
|---|---|
| **Course of Study:** | Information Technology/INFOTECH |
| **Examiner:** | Prof. Dr. Kurt Rothermel |
| **Supervisor:** | Dr. Frank Dürr & Dr. Péter Bakucz |
| **Commenced:** | 2017-03-27 |
| **Completed:** | 2017-09-27 |

# ACKNOWLEDGEMENT

My thanks to **Prof. Kurt Rothermel** for giving me the opportunity to work in this department on an interesting and challenging topic. I would like to thank my supervisor **Dr. Frank Dürr** of the Institute of Parallel and Distributed Systems at Universität Stuttgart. His support was very valuable for the entire duration of my work. I was given the freedom to express my thoughts, opinions and ideas. His suggestions and teachings were a guiding light for me in the right direction.

I would also like to thank my thesis supervisor **Dr. Péter Bakucz** of the Bosch Connected Devices and Solutions for his support in this study. This work would not have been possible without his expertise and periodic interactions with him always added value to my work. I would also like to thank Bosch Connected Devices and Solutions system architect **Mr. Alexander Paharukov** for his inputs, passionate participation and review of my work.

Finally, I must express my very profound gratitude to my parents and my sister Ms. Jeyashree Mari for providing me with unfailing support and continuous encouragement throughout my years of study. I would forever be indebted to my friends without whose help and support, this study would not have been possible. Thank You.

Hariharan Mari
Stuttgart, Germany

## Abstract

Bluetooth Low Energy (BLE) is a short range, low power wireless networking technology that is increasingly used in various applications of the Internet of Things (IoT). Whereas the performance of classical Bluetooth has already been intensively studied using analytical models and simulative methods. These studies are not directly applicable to BLE since BLE is based on a fundamentally different design targeting extremely low power consumption rather than high data rates. In particular, there are no sufficient models to simulate the performance of BLE in multi-device environments where several BLE peripheral devices are connected to a master device in a piconet (star topology). This lack of comprehensive models for the simulative analysis hinders the development and evaluation of IoT systems based on BLE.

Keeping them in mind, the goals of this thesis is to design, implement, and evaluate simulation models for BLE focusing on multi device environments. Relevant performance metrics include the delay and energy consumption. Suitable abstraction of the different protocol layers has been defined to allow for implementing relevant aspects for the simulator. The simulation results are compared with real world hardware to measure the difference and to find the reasons for the differences.

## Table of Contents

# Table of Figures

## Table of Tables

# 1 Introduction

## 1.1 Motivation

Bluetooth Low Energy is the power-version of classical Bluetooth that was built for the Internet of Things (IoT). Bluetooth is wireless technology standard using short range radio links, intended to replace the cables connecting portable electronic devices. This technology offers internet to any electronic systems. Services over the internet have been evolved based on the needs that have been identified from person-to-person interaction, machine-to-person interaction and machine-to-machine interaction thus building to pervasive computing. Such a computing is being built to automate tasks and build a smart world. Internet of Things (IoT) allows a lot of services to be built over the network and serves as a meeting point between the real world and virtual world. The Internet of Things (IoT) refers to the interconnection of these devices which are denoted as “smart objects”.

BLE is the distinctive feature of the Bluetooth 4.0 specification. BLE-based sensors are able to operate on a coin cell for several months to several years, depending on its processing and communication demands and the parameterization of the (BLE) protocol. The protocol has been standardized in 2010 as the part of Bluetooth Core Specification version 4.0. While there is some overlap with classic Bluetooth, BLE actually has a completely different lineage.

A multi-device environment is where many smart devices are connected to share data among them. In a BLE based multi-device environment, there exist many smart devices operated by batteries which are connected through BLE wireless protocol. BLE protocol thus allows the devices to have very low power consumption. The BLE connection can be configured using various configuration parameters allowed by Bluetooth core specification. The performance of the smart objects changes depending on configuration parameters. Thus it is evident that these configuration parameters have a greater effect on overall performance of a multi-device environment.

Modeling and simulation of a BLE environment will aid in tuning the right configuration parameter for any application scenario. The configuration parameters which play a major role are advertising interval of the slave device, scanning interval and scanning window of the master device, connection interval and connection latency of both slave and master devices. Deducing sensitivity of each configuration parameters will assist in realizing any application scenario.

Validation of the simulation is done by implementing the same scenario in a real hardware environment. Comparison of results from hardware implementation with the results from simulation will expose the accuracy of our software model. Achieving the above will allow developers to use the simulation model to realize any new application scenarios. The model should also be flexible when new core specification is announced by Bluetooth SIG. The simulation is done in OMNeT++ network simulator.

## 1.2 Objectives

The objectives of the master thesis would be to

- Definition of exemplary multi-device scenario, and analysis of the requirements with respect to the relevant properties to be modelled
- Modelling the parameters of BLE environment
- Simulating the BLE sensor network in OMNeT++ (Network Simulator)
- Validation of results with real world hardware
- Evaluating the accuracy of the simulation model by comparing with hardware results

## 1.3 Thesis Organisation

The thesis has been organized as described below:
Chapter 2 provides an overview of topics which are necessary for understanding to the thesis.
It includes an overview on OMNeT++, MiXiM, Bluetooth Low Energy concepts.
Chapter 3 provides a description of system model and the problem statement of the thesis.
Chapter 4 provides a description of implementation involved for the proposed system.
Chapter 5 provides an overview of the evaluations and an analysis of results.

# 2 Background

## 2.1 Simulation tools for Wireless technology

In the expeditiously growing Internet of Things (IoT), applications from personal electronics to industrial machinery are getting connected wirelessly to the internet. There are numerous wireless standards deployed in the market, spreading over various frequency bands and using different communication protocol. Choosing the right wireless technology is becoming more and more challenging every day. Bluetooth Low Energy (BLE) is a fresh addition to the Bluetooth Specification. Bluetooth Smart is a synonymous term to BLE. Designed specifically for lower data throughput, BLE significantly reduces the power consumption, enables years of operation using coin cell batteries. One of the most important advantages of Bluetooth standard is that it includes application profiles. These profiles define how application exchange information to achieve the specific job.

Three main traditional techniques for analyzing the performance of wireless networks; analytical methods, computer simulation, and physical measurement. Traditionally, formal modeling of systems has been via a mathematical model. The mathematical model attempts to derive analytical solutions to problems and so that enable the prediction of the behavior of the systems from a set of parameters and initial conditions. However, it is widely popular that comprehensive models for wireless ad-hoc networks are mathematically intractable. Each individual layer and the interactions between the layers in the protocol stack magnifies complexity. The construction of real test beds for any predefined application scenario is usually expensive in terms of time and effort. Simulation is the most common approach to develop and test new or existing protocol for a wireless network [1]. It is also understood that the simulation model cannot represent the 100% of a real hardware. An acceptable abstraction of the real hardware should be achieved.

It is a necessity to analyze different simulation tools that are suitable to test and evaluate wireless technology particularly BLE. Few advantages of these simulators are that it allows simulating geographically distributed devices and allows code reuse. Some of the simulation tools that are available are OMNeT++, Ns-3, GloMoSim, J-Sim, OPNeT, QualNet, and JiST/SWANS. These tools can be used in different areas of applications. Since there exist various simulation tools, a detailed study and comparison of all the most important tools are needed.

### 2.1.1 Discrete event simulation

In the computer simulation, the behavior of the system is modeled over time. Discrete event simulation is one of many simulation approaches that has been proposed in history. With respect to other simulation approaches, it has good expressiveness and is easy to use. A discrete event simulation is represented by a simulated model with a set of state variables and their sequence of events is processed in chronological order. Here each event occurs at a given instant of time and represents a change in the simulated model state. The whole evolution of the simulated system is obtained through an ordered sequence of events that are stored, created and processed. Here each event is tagged by a timestamp that specifies the simulated time the event has to be processed. This has substantial advantages compared to sequential simulation

### 2.1.1 OMNeT++

OMNeT++ is a C++ based discrete event simulator for modeling communication networks, multiprocessors, queueing networks and distributed and parallel systems. It is an open source discrete event simulation tool that can be used by educational, research and academic institutions. OMNeT++ represents a framework approach. Instead of directly providing simulation components it provides the basic machinery and tools to write such simulation. It supports various simulation models and frameworks such as Mobility Framework and MiXiM framework [2]. The common file types that are present in OMNeT++ include

- .ned -- where the simulation components and networks are defined
- .cc -- C++ source files of simulation components
- .msg -- message descriptions that are automatically translated to C++ classes
- _m.cc,_m.h -- generated files created from msg files during the build process
- .ini -- parameter settings and configuration options for simulation

#### 2.1.1.1 Design of OMNeT++

OMNeT++ was designed to support network simulation on a large scale. It consists of modules that communicate by passing messages. The active modules are the simple modules which are written in C++ using the simulation class library. Simple modules can be grouped into compound modules Figure 1. Messages are sent via connections that connect the modules or directly to their destination modules. The simple and compound modules are instances of module types. The user creates a system module as a network module which is a compound module without gates to the external world. These modules can be reused with the help of model frameworks like mixed signal (MiXiM) framework, INET framework and Mobility framework.

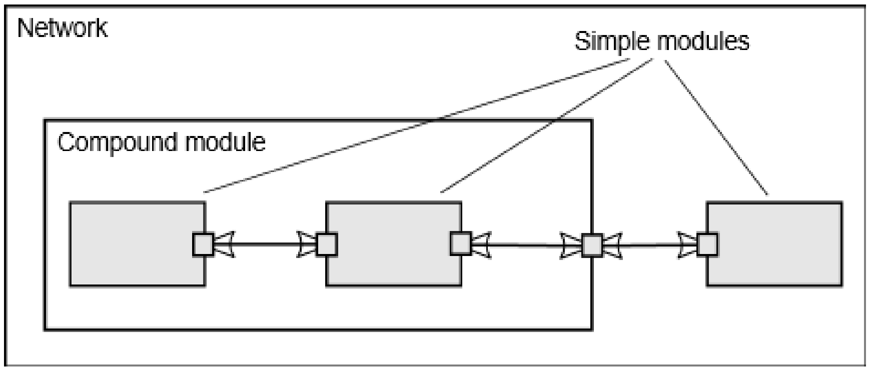


*Figure 1: Model structure in OMNET++ [2]*

The modules communicate with messages which contains some attributes and data. Gates are the input and output interfaces of the modules: messages are sent out through output gates and they arrive through input gates. The input and the output gate are linked by a connection. Properties such as propagation delay, data rate and bit error rate can be assigned to connections. We can also assign some specific properties termed channels and reuse them at several places [2].

##### 2.1.1.1.1 Design of NED Language

The structure of the model (the modules and interconnection) in OMNeT++ are described using topology description language file called NED. Typical NED descriptions include simple module declarations, compound module declarations and network definitions. Simple module declarations describe the interface of the module, the gates and parameters. Compound module definitions consist of the declaration of the module’s external interfaces like gates and parameters and definition of submodules and their interconnection. Network definitions are self-contained simulation models with the simple modules and compound modules. Some of the major features of NED language in OMNeT++ include

- Inheritance
- Interfaces
- Packages
- Inner types
- Metadata annotations

OMNeT++ consists of a graphical editor which helps to edit network topology files graphically or in NED source view.

##### 2.1.1.1.2 Model and Experiments

In OMNeT++ model behavior is captured in C++ files as code while model topology is defined by the NED files. This approach allows to keep the different aspects of simulation at different places thus allowing to have cleaner model. The different inputs for the simulation are stored in the INI files. INI files are used to specify the parameters during the simulation.

##### 2.1.1.1.3 Simple Module Programming Model

Simple modules are the active elements in a model. These modules are programmed in C++ using the OMNeT++ simulation class library. OMNeT++ has an integrated C++ development environment to write, run and debug the code. The functionality of the simple module is implemented using the cSimpleModule class. Functionality can be added via one of the two programming models. They are coroutine based and event processing function [2].

### 2.1.2 Ns-3

Ns-3 is a discrete event network simulator for internet systems. It is a free software available for research, development and use. The ns-3 software infrastructure encourages the development of simulation models which are realistic to allow ns-3 to be used as real time network simulator, interconnected with the real world thereby allowing many real world protocol implementations to be reused within ns-3. Ns-3 is built using C++ and Python scripts. Some of the distinguishing features of ns-3 are

- Ns-3 is designed as a set of libraries that can be combined with external software libraries. While some simulation platforms provide users with a graphical user interface environment, ns-3 is more modular in this regard. External simulators, data analytics and visualization tools can be used with ns-3. However unlike OMNeT++, in ns-3 we have to work at command line.
- Ns-3 is primarily used in Linux systems
- Ns-3 is not officially supported by any company. It is being developed by the interests of ns-3 users [3].

### 2.1.3 JiST/SWANS

JiST represents a interesting approach in building a high performance Java based simulation environment. It modifies the Java virtual machine to run the programs in simulation time instead of real time. JiST is basically a simulation kernel and as such it lacks most of the features present in OMNeT++ package. SWANS is a scalable wireless network simulator which is built atop the JiST platform. The development of this software has been stopped since 2005 [4].

### 2.1.4 J-SIM

J-Sim is an application development environment based on component based software architecture, Autonomous component Architecture (ACA). J-Sim has a specific platform dedicated to network simulation. J-Sim is a real time process driven simulator. It runs like a real system where the event execution are run real time as opposed to fixed time points in discrete event simulation. Two languages are used in J-Sim: Java to describe and implement models and a script language to construct, configure and control the simulation at run time. Here it makes implementions with graphical editors impossible like Ns-3. J-Sim provides Runtime Virtual commands to simplify the manipulation and configuration of the network components during run time. J-Sim is similar to OMNeT++ in several points. But it doesn't have features like OMNeT++. Here the simulation performance is significantly weaker than with C++ [5].

### 2.1.5 OPNET

It provides a global environment to model, simulate and evaluate all kinds of wired and wireless communication networks and distributed systems. The OPNET environment consists of graphical tools for scenarios and model conception, scenarios simulation, data analysis and data collection. OPNET has rich simulation libraries. Its simulation library is based on C and its architecture is similar to OMNeT++ but with few restrictions. OPNET models are of fixed topology. The network is defined using graphical editor, where the editor stores models in a proprietary binary file format and they are usually difficult to generate by program. Its main advantage over OMNeT++ is that it has large protocol library but makes development and problem solving harder [6].

### 2.1.6 MATLAB

MATLAB is a numerical computing environment. It allows matrix manipulations, plotting of data and functions, implementation of algorithms, math functions and creation of user interface. MATLAB is primarily intended for numerical computing but additional package like Simulink allows model based design for dynamic and embedded systems. It has lot of additional blocksets like communication system toolbox, Data acquisition toolbox, Image acquisition, SimEvents, Stateflow and many more. SimEvents and Stateflow are the two discrete simulators of MATLAB.

## 2.2 Comparison of OMNeT++ with other simulation tools

OMNeT++ offers many advantages when compared to other tools like Ns-3, and OPNET Modeler. Some of the advantages include they have good graphical environment, support wide variety of tools, good simulation performance, reuse of existing models and by allowing parametric topologies. In the Table 1 a comparison of different simulation tools over their characteristics is made. In the Table 2 a comparison over the merits and demerits of different tools is been made.

| Simulator / Characteristics | OMNET++ | Ns-3 | J-SIM |
|---|---|---|---|
| **Language supported** | C++ | C++, Python | Java |
| **Network support type** | Wired network, Wireless managed node | Wired and wireless network | Wired and wireless network |
| **Platform** | Windows, Linux, Mac OS | Linux, Windows | Windows, Linux |
| **GUI support** | Yes | No | Yes |
| **Time taken to learn** | Moderate | Long | Moderate |
| **Interaction with real systems** | Yes | Yes | Yes |
| **Vendor and available site** | Omnest http://www.omnetpp.org/component/docman/cat_view/17-downloads/1-omnet-releases | http://www.nsnam.org/ns-3-13/download | https://sites.google.com/site/jsimofficial/downloads |

*Table 1: Comparison of Tool*

| Features | NS-3 | JSIM | OMNeT++ |
|---|---|---|---|
| **Merits** | -It has network visualization tools<br>-It has analysis tool and trace file<br>-It is possible to design and modify the network scenarios<br>-Supports both wired and wireless communication of protocols<br>-Interaction with real time system is possible | -It has network visualization tools<br>-It has analysis tool and trace file<br>-It is possible to design and modify the network scenarios<br>-Supports both wired and wireless communication of protocols<br>-Provides support for energy modeling with the exception of radio energy consumption<br>-supports mobile wireless networks and sensor networks<br>-component oriented architecture | --It has network visualization tools<br>-It has analysis tool and trace file<br>-It is possible to design and modify the network scenarios<br>-Supports both wired and wireless communication of protocols<br>-powerful graphical user interface making tracing and debugging easy<br>-simulate power consumption problem |
| **De Merits** | -Python bindings do not work on Cygwin<br>-Only IPv4 is supported | -Low efficiency of simulation-the only MAC protocol provided for wireless networks is 802.11<br>-unnecessary run time overhead | -number of protocols are not enough |

*Table 2: Comparison of tools over properties*

## 2.3 MiXiM Framework

OMNeT++ provides a clear and powerful simulation framework. But there are no concise models for wireless communication. Mixed simulator (MiXiM) combines and extends several existing simulation frameworks developed for wireless protocol simulations in OMNeT++. MiXiM provides detailed models of a wireless channel like fading, wireless connectivity, mobility models, models for obstacles, many communication protocols at the medium access control (MAC) level and a supporting infrastructure. This framework provides a user-friendly graphical representation of wireless and mobile networks and supports debugging in OMNeT++. Simulating wireless communication systems can be achieved only when a befitting abstraction of the environments, the radio channels, and the physical layer [7] are designed.

### 2.3.1 Environmental models

The abstraction of the environmental area or playground where wireless nodes and objects are placed has an ample impact in the simulation. Node embodies the wireless devices with their protocol stack. They are modeled as isotropic radiators not involving a physical dimension. An object, in contrary, is something with a physical dimension that occupies the propagation environment. These objects tend to attenuate a wireless signal. Both nodes and objects can be mobile. It is also possible for a node to combine with objects e.g. sensor node mounted on a car. MiXiM provides a user-defined update interval for mobility modeling of nodes and objects. This parameter defines how often the position of an object is updated.

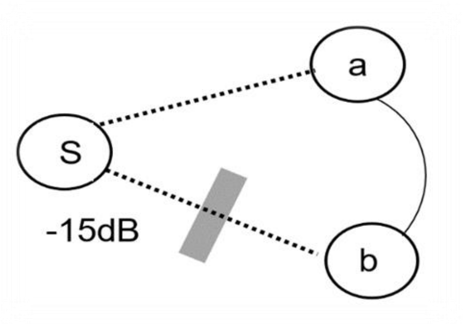


*Figure 2: Environment with 'S' as signal transmitter and 'a' and 'b' as signal receiver*

The position information contains the start time, start position, direction, and the speed of an object. The shadowing effect of objects that reside in the propagation environment causes different received signal strengths at equal distance. This property is called the stochastic model and has to be adapted to the characteristics of the environment that are to be simulated. Objects are characterized by position, an angle of rotation, dimension, and frequency- dependent attenuation factor. Like in Figure 2 the wireless signal can get affected by any object in the line of sight between transmitter and receiver.

### 2.3.2 Connection modeling

Compared to wired simulations, connectivity modeling is a complicated task in the wireless simulation. It can be defined in two parts, the wireless channel and its attenuation property is the first part and the connectivity between nodes becomes the second part. MiXiM channel models enable multiple parallel radio channel in frequency and space. The radio propagation effect in each of this channel is expressed as a time variant factor of instantaneous Signal-to-Noise Ratio (SNR) of the received signal. MiXiM modular structure enables to include models operating on digital signal level e.g. modulation symbols. In SNR abstraction level, MiXiM includes widely accepted channel models for path-loss, shadowing, large and small scale fading.

In theory, a signal sent out by a node will be affected by any other nodes in the simulation (if operating in the same frequency range). As a result of signal attenuation, the received signal power at nodes that are far away from the sending node will be so low. In MiXiM, nodes are connected only when they are within the maximal interference distance in order to reduce the computational complexity. The maximal interference distance is a conservative bound on the maximal distance at which a node can still possibly disturb the communication of a neighbor. It is also important to note that the maximal interference distance does not specify the maximal distance at which messages can be received. Nodes which wants to receive a message from a communication peer also receives all interference signals and can thus, decide on the interference level and resulting bit errors. The existence of objects in the propagation environment also affects the maximal interference distance [7].

### 2.3.3 Physical layer

The physical layer is the core part of a wireless node in MiXiM. It is accountable for message sending and receiving, bit error calculation and collision detection. Additionally, it is responsible for applying channel models used in the simulation. The MiXiM physical layer is divided into three parts, which are described in detail in the following subsections. The base physical layer provide the interfaces to the MAC layer and the physical layers of other nodes. The Analog models are responsible for simulating the attenuation (like fading, shadowing, and path-loss) of a received signal. The decider helps in evaluation (classification as noise or signal) and demodulation (bit error calculation) of the received messages. In the physical layer, the modulation that has been used, forward error correction (FEC) coding and decoding functions define the bit error rate and throughput of a system. The effects of these functions in the wireless channels can be modeled at SNR level.

#### 2.3.3.1 Notion of the Signal

The signal strength of a message sent from one node to another is controlled by the environment it travels through. This can be modeled with attenuation factors as a result of path loss, shadowing and fading. Moreover, a message can be sent using multiple frequencies (e.g. OFDM) and using multiple antennas (MIMO). As a result of all these phenomena, a message can have varying sending power, bit rate, and attenuation. In MiXiM, the signal class is created to model this complex process. During the process of sending a message, a node has to specify the sending power and bit-rate in the appropriate dimensions. The receiving node then adds the attenuation. Based on the whole signal, bit errors can be calculated [8, 9].

#### 2.3.3.2 BasePhylayer

Other than message sending and receiving, the BasePhylayer works as an interface between physical layer messages (Airframes) and the analog model and the Decider. In node, when receiving a message, the physical layer first passes the message to the analog model. The analog model calculates the attenuation part of the signal. The physical layer then simulates propagation and transmission delay of the message. The message is passed at least twice to the decider: at the beginning and at the end of the message. Finally, after the decider calculates the bit errors, the message will then be handed over to MAC layer.

#### 2.3.3.3 Analogue models

MiXiM also simulates feature like path loss, shadowing and fading. The attenuation of a signal is calculated by implementations of shadowing, fading and path-loss models. Any number of analog models can be plugged into the physical layer of MiXiM. Each model is simply a filter class for a signal. Integrating the attenuation of all analog models gives the attenuation part of the signal, which is calculated at the start of the reception of a message. By sending the power of a received packet the decider can, later on, calculate the SNR and thus, bit errors.

#### 2.3.3.4 Decider

The decider has three main tasks. Firstly, the decider has to classify incoming messages into receivable message or noise. Secondly, the decider has to calculate the bit errors for the message at the end of receiving receivable messages. Finally, it has to hand over the information about the current state of the channel. MiXiM has several models determining how and when a physical layer decides whether a message can be received or is just noise. The decider can request the message from physical layer at the end of the receiving process of the message. At this time, the decider has to calculate the bit errors for the message. For this process, decider requests all intersecting messages from channel info in order to calculate the SNR for the message. After this, decider can make a simple binary decision (received correctly or not). Providing channel state is the last task of a decider. The channel state is needed at the MAC layer. The decider will sense the channel for a certain amount of time on the request of the MAC layer. The decider then returns whether the channel is currently idle or busy [7].

### 2.3.4 MiXiM Protocol library

MiXiM permits every module to be replaced by another module in the simulation, adding functionality to the base implementation. There is a wide choice of implemented protocols are available in MiXiM protocol library.

#### 2.3.4.1 MAC Protocols

A Medium Access Control (MAC) protocol is devised to make decisions about the sharing of a medium for communication between nodes of a system. In the case of wireless systems, the shared medium is air. The main role of a MAC protocol is its needs to decide when a node should send out messages so that the messages do not interfere with messages of other nodes. Additionally, for low power devices (BLE devices) – the MAC protocol is responsible to determine at what times the radio can be switched off to avoid listening to the medium (which consumes power) when no nodes are sending. In MiXiM we have two base classes for building sensor network specific MAC protocols.

1. BaseMACLayer – basic MiXiM-style layering, providing en/decapsulation of packets, but no other functionality
2. EyesMACLayer – supplies a number of support functions for sensor network MACs, comprise support functions for low-power listening and sift inspired carrier sense period choosing, also generating statistical information about MAC protocol performance.

MiXiM implements standard MAC protocols for wireless Local Area Networks (WLAN) and Personal Area Networks (PANs). The IEEE 802.11b/g family and IEEE 802.15.4 standards can be ported from the Mobility framework to MiXiM.

#### 2.3.4.2 Network layer Protocols

MiXiM reinforces networking protocol with a wide variety of traffic paradigms, these are further supported by other simulation models, e.g. localization data for geographic routing, motion data derived from mobility module.

#### 2.3.4.3 Mobility models

MiXiM has an elaborate library of mobility modules which include simple modules like “constant speed mobility” and “Circle mobility”. It has a simplified way to create new mobility modules, by sub-classing from the base BaseMobility class. The BaseMobility class offers all the functionality needed for mobility handling in MiXiM.

#### 2.3.4.4 Localization

Localization is a feature to determine position information of the current node. This service is made available by the optional localization layer. Optional shows that it is not included in the standard software protocol stack, but must be added to the node module explicitly. This optional nature permits the placement of the localization layer beneath the network layer, such that geographic routing algorithms can utilize the localization layer for position information. The BaseLocalization layer aims to supply base functionality for a wide range of localization algorithms. The localization layer embeds position information in the header message from the upper layer, such that algorithms can be created that have little or no message overhead [7].

## 2.4 Bluetooth Low Energy - Overview

Bluetooth Low Energy is the power-version of Bluetooth that was built for the Internet of Things (IoT). BLE is the distinctive feature of the Bluetooth 4.0 specification. BLE-based sensors are able to operate on a coin cell for several months to several years, depending on its processing and communication demands and the parameterization of the (BLE) protocol. The protocol has been standardized in 2010 as the part of Bluetooth Core Specification version 4.0. Internet of Things (IoT) can be benefited largely by Bluetooth with low energy functionality. Bluetooth Low Energy is sometimes referred to as “Bluetooth Smart”. While there is some overlap with classic Bluetooth, BLE actually has a completely different origin and was started by Nokia as an in-house project called ‘Wibree’ before being adopted by the Bluetooth SIG. There are good-deal of wireless protocols out there for engineers and product designers. BLE remains in sleep mode constantly except for when a connection is initiated. It is vital for applications that only need to exchange small amounts of data periodically. Appealing factor about BLE is that it helps to easily design something that can talk to any modern mobile platform out there (iOS, Android, Windows phones, etc) and particularly in the case of apple devices it’s the only Hardware (HW) design option that doesn’t require any developers to bounce through interminable circlet. The Bluetooth SIG envisioned that by 2018 more than 90 percent of Bluetooth-enabled smartphones will up-hold Bluetooth Smart. The Bluetooth SIG officially proclaimed Bluetooth 5. Bluetooth 5 will double the speed, quadruple the range, and furnish an eight-fold increase in data broadcasting capacity of low energy Bluetooth transmission compared to Bluetooth 4.x, which could be important for IoT applications especially in smart industry, logistics, and also in smart homes.

### 2.4.1 Classic Bluetooth and Bluetooth Smart

Classic Bluetooth technology was originally designed for continuous streaming data application such as voice and was successful in getting rid of wires in many consumers as well as industrial and medical applications. Classic Bluetooth technology will continue to grant a robust wireless connection between devices ranging from headsets and cars to industrial controllers and streaming medical sensors. The IEEE standardized Bluetooth as IEEE 802.15.1, but no longer maintains the standard. Bluetooth Low Energy was introduced in 2011. As with classic Bluetooth technology, BLE operates in the 2.4 GHz ISM band and has similar radio frequency (RF) output power. However, because a BLE is in sleep mode most of the time and wakes up only when a connection is commenced, the power consumption can be kept to minimum. Power consumption is kept low because the actual connection times are of only a few ms. The maximum, or peak power consumption is only 15 mA, and the average consumption is only about 1µA. Classical Bluetooth operates in the 2400 - 2483.5 MHz range within the ISM 2.4 GHz frequency band. Table 3 describes the difference between classical Bluetooth and Bluetooth Low Energy.

| **Technical Specification** | **Classical Bluetooth** | **BLE Smart technology** |
|---|---|---|
| **Data rate over the air** | 1 Mbps | 1-3Mbps |
| **Range /Distance (Theoretical)** | ~10 – 100 meters | ~10 – 100 meters |
| **Frequency Channels** | 79 channels from 2.4 GHz to 2.483 GHz with 1 MHz spacing | 40 channels from 2.4 GHz to 2.48 GHz (3 advertising and 37 data channel) |
| **Throughput** | 0.7 – 2.1 Mbits/s | 0.27 Mbits/s |
| **Security** | 56/128 bit and application layer user defined | 128 bit AES with Counter Mode CBC-MAC and application layer defined |
| **Robustness** | Adaptive fast frequency hopping, FEC, fast ACK, FHSS | Adaptive frequency hopping, 24 bit CRC, 32 bit Message integrity check, FHSS |
| **Latency (from a non-connected state)** | 100ms | 6ms |
| **Link Layer** | TDMA | TDMA |
| **Network Topology** | Point-to-point, scatternet | Point-to-point, star, scatternet |
| **Modulation** | TDMA | TDMA |
| **Power consumption** | 1 (reference value) | 0.01 to .5 |
| **Message size (bytes)** | 358 (Max) | 8 to 47 |
| **Profile concept** | Yes | Yes |
| **Primary use cases** | Mobile phones, headsets, stereo, automotive, PCs | Mobile phones, gaming, PCs, Sport & fitness, medical, automotive, automation, home electronics |
| **Voice Capable** | Yes | No |

*Table 3: High level comparison between Classic Bluetooth and Bluetooth Low Energy*

## 2.5 Bluetooth Core Specification

The Bluetooth core specification defines the technology building blocks that act as a guiding manual for developers to create the interoperable devices that make up the thriving Bluetooth ecosystem. Bluetooth Special Interest Group (SIG) oversees the Bluetooth specification [10]. This section presents the BLE protocol stack and defines the main mechanism and features of each layer.

### 2.5.1 General Description

The Bluetooth wireless technology was introduced as a short-range communication system designated to replace the cables. Core specification has many features which are optional, allowing product differentiation. There are two modes of Bluetooth wireless technology systems: Basic Rate (BR) and Low Energy (LE).

Both the Basic rate (BR) and Classical Bluetooth and Low Energy systems include device discovery, connection establishment, and connection mechanism. The Basic Rate system includes optional Enhanced Data Rate (EDR), Alternate Media Access control (MAC) and Physical (PHY) layer extensions. The LE system includes features designed to enable products that demand lower current consumption, lower complexity, and lower data rates and has lower duty cycles depending on the use case or application. The Bluetooth core system consists of a Host and one or more Controllers. A Host is a logical entity enclosed of all of the layers below the non-core profiles and above the Host Controller Interface (HCI). A Controller is a logical entity defined as all of the layers below HCI. The Host Controller Interface (HCI) takes care of the communication between the Host and the Controller. The Controller includes the Physical layer and the link layer. The Host includes upper layer functionality which is Logical Link Control and adaptation Protocol (L2CAP), the Attribute Protocol (ATT), the Generic Attribute Profile (GATT), the Security Manager Protocol (SMP) and the Generic Access Profile (GAP) [11]. Figure 3 illustrates the BLE Protocol Stack.

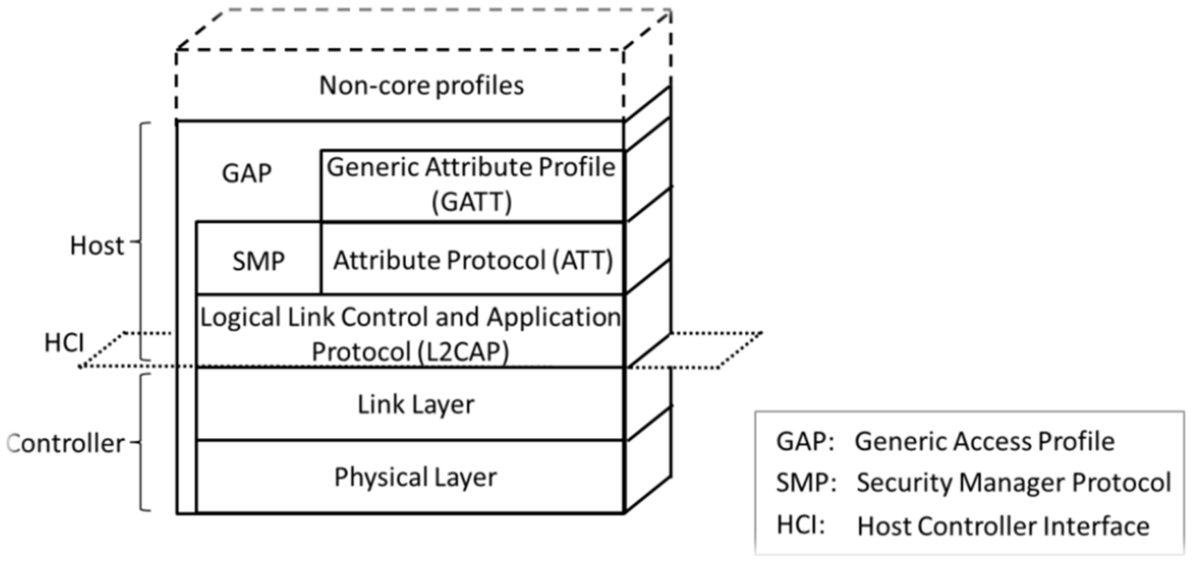

Non-core profiles
GAP
Generic Attribute Profile (GATT)
SMP
Attribute Protocol (ATT)
Logical Link Control and Application Protocol (L2CAP)
Link Layer
Physical Layer
Host
HCI
Controller
GAP: Generic Access Profile
SMP: Security Manager Protocol
HCI: Host Controller Interface

*Figure 3: BLE Protocol Stack [1]*

#### 2.5.1.1 Physical Layer

Bluetooth Low Energy (BLE) device operates in the 2.4 GHz Industrial Scientific Medical (ISM) band. The transceivers use frequency hopping mechanism [8] to fight interference and fading. The LE system uses 40 RF Channels. The center frequency of these RF channels is 2402 + k*2 MHz, where k = 0, ..., 39. There are two types of RF channels: advertising channel and data channels.

| Regulatory Range | RF Channels |
|---|---|
| **2.400 – 2.4835 GHz** | f = 2402 + k*2 MHz, where k = 0, ..., 39 |

*Table 4: Operating frequency Bands*

Advertising channels are utilized for device discovery, connection establishment and broadcast transmission. Data channels are used for bi-directional communication between connected devices. Three channels are defined as advertising channels. An adaptive frequency hopping mechanism is used over data channels to counter balance interference and wireless propagation issues, such as fading and multipath. This frequency hopping mechanism picks one of the 37 available data channels for communication during a given time interval.

All physical channels follow Gaussian Frequency Shift Keying (GFSK) modulation [8], which are simple to implement. The modulation index is in the range between 0.45 and 0.55, which permits reduced peak power consumption. The data rate in physical layer is 1 Mbps. The sensitivity of the receiver is defined in BLE as the signal level at the receiver for which a Bit Error Rate (BER) of $10^{-3}$ is reached. This sensitivity should be better than or equal to – 70 dBm according to Core Specification [11].

#### 2.5.1.2 Link Layer

The operations in the link layer can be expressed in terms of state machine with following five states

- Standby state
- Advertising state
- Scanning state
- Initiating state
- Connection state

The link layer in the Standby state does not transmit or receive any packets. The Standby state can be entered from any other state.

In BLE, if a device wants to broadcast some data, it can transmit the data in advertising packets through advertising channels. That device which transmits advertising packets is termed as advertisers. The transmission of packets take place in the interval of time called advertising event. This interval is known as the advertising interval and it is a multiple of 0.625 ms and it can range from 20 ms to 10.24 s. Inside an advertising event, the advertisers sequentially employ each advertising channel for data packet transmission. The advertising state can be entered from the Standby state.

The link layer in the Scanning state will be listening for advertising channel packets from advertisers. A device in the Scanning state is known as scanners. The link layer in the Initiating state will be listening for advertising channel packets from particular devices and responding to initiate a connection with that advertising device. The devices which perform scanning and initiating are called scanner and initiator respectively. Scanning is performed for a duration of scan window in a periodic interval. This interval is called scan interval. Scan interval and scan window are multiples of 0.625 ms and can range from 2.5 ms to 10.24 s. As shown in Figure 4, the Connection state can be entered from either Advertising state or the Initiating state [11].

The connection creation between two devices is an asymmetric procedure by which an advertiser disclose that it is a connectable device, at the same time the other device which is referred to as an initiator listens for such advertisements. As soon as an initiator discovers an advertiser, it can transmit a Connection Request message to the advertiser. This is the procedure to create a point-to-point connection between two devices. Link layer has only one packet format for both advertising channel packets and data channels packets. The packet format is shown in Figure 5. Each packet consists of four fields: the preamble, the Access address, the PDU, and the CRC. The preamble is 1 octet and the access address is 4 octets. The Range of PDU is from 2 to 257 octets.

The CRC is 3 octets. The packets in this connection can be recognized by a randomly generated 32-bit access code.

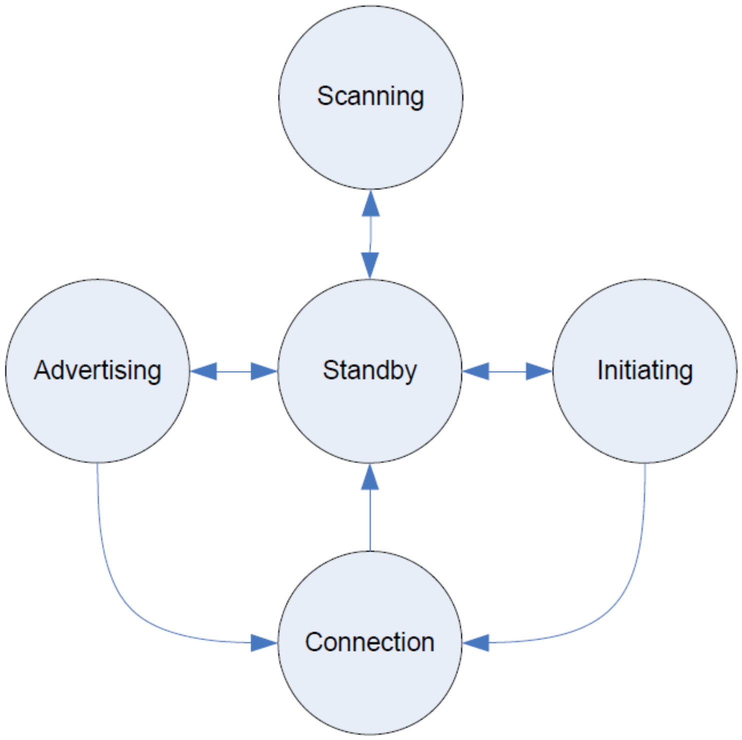
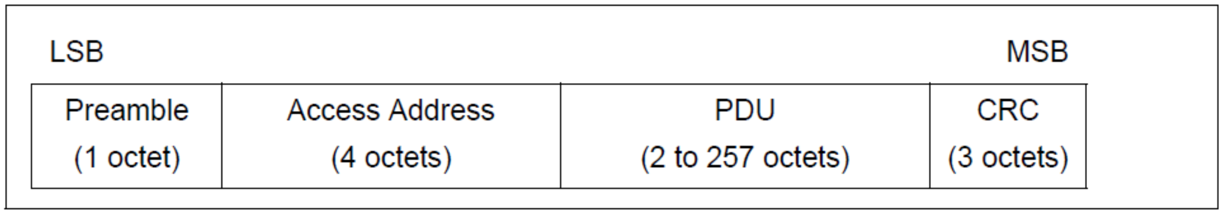


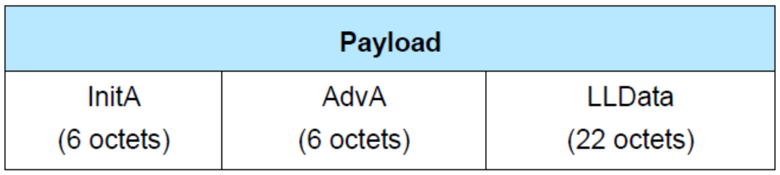
*Figure 4: State diagram of the link layer state machine [9]*



| LSB | | | MSB |
|---|---|---|---|
| Preamble (1 octet) | Access Address (4 octets) | PDU (2 to 257 octets) | CRC (3 octets) |

*Figure 5: Link Layer packet format [10]*

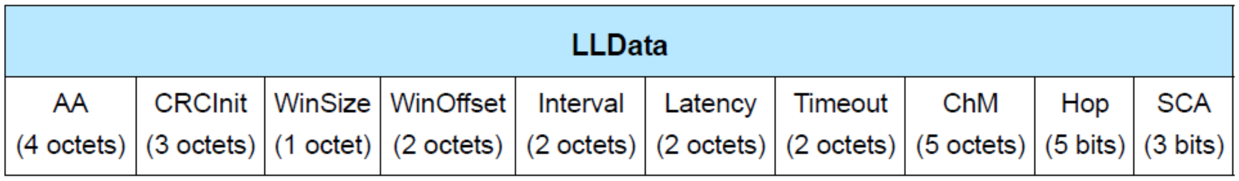
The master and the slaves are the devices in connection state that act as initiator and advertiser during connection creation, respectively. From [10] the master has the capability to manage multiple simultaneous connections with different slaves, whereas each slave can only be connected to one master. Thus, the network our constructed by a master and its slaves, which is called a piconet, follows a star topology. By default, slaves are in sleep mode and wake up periodically to listen for possible packet receptions from the master, to save energy on the slave side.



| **Payload** | | |
|---|---|---|
| InitA (6 octets) | AdvA (6 octets) | LLData (22 octets) |

*Figure 6: Connection Request PDU payload [1]*

The master coordinates the medium access b using a Time Division Multiple Access (TDMA) scheme. The masters also inform the slave about the frequency hopping algorithm (including the map of data channels to be used) and for the connection supervision. The connection management based parameters are transmitted in the Connection Request message. The connection message payload is shown in Figure 6 and Figure 7.



| LLData | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| AA (4 octets) | CRCInit (3 octets) | WinSize (1 octet) | WinOffset (2 octets) | Interval (2 octets) | Latency (2 octets) | Timeout (2 octets) | ChM (5 octets) | Hop (5 bits) | SCA (3 bits) |

*Figure 7: LLData field structure in Connection Request PDU Payload [10]*

Formally after the creation of a connection between a master and a slave, the physical channel is divided into non-overlapping time units called connection events. Every connection event starts with the transmission of a packet by the master. When the slave receives a packet, it must send a packet to master in response. At the same time, the master is not required to send a packet upon receipt of a packet from the slave. Minimum, an Inter Frame Space (IFS) pf 150 µs must pass between the end of the transmission of a packet and the start of the next one. The connection event is considered to be open, while master and slave continue to alternate in sending packets. Data channel packets (Figure 8 and Figure 9) include a More Data (MD) bit which signals whether the sender has information to transmit. If none of the devices has more data to transmit, the connection event will be closed and the slave will not be required to listen to the beginning of next connection event.

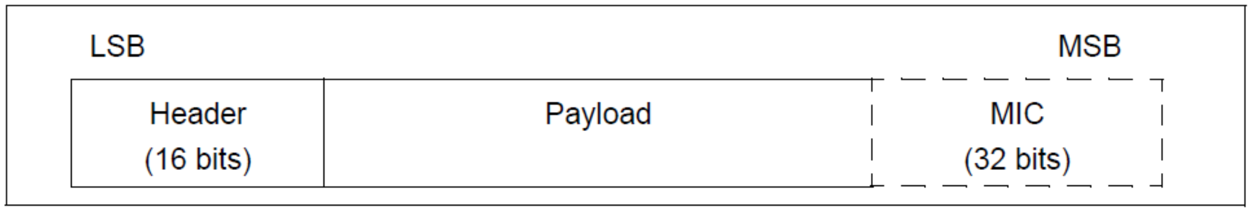
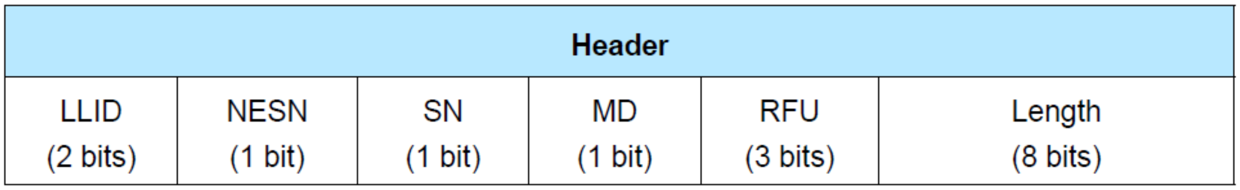


*Figure 8: Data Channel PDU [10]*

Other situations that force the end of a connection event include the reception of two consecutive packets with bit errors by either the master or the slave and the corruption of the access address field of a packet sent by any device.



| Header | | | | | |
|---|---|---|---|---|---|
| LLID (2 bits) | NESN (1 bit) | SN (1 bit) | MD (1 bit) | RFU (3 bits) | Length (8 bits) |

*Figure 9: Data channel PDU Header [9]*

For a new connection event, master and slave make use of a new data channel frequency. This frequency is computed by using the frequency hopping algorithm. The time between the start of two consecutive connection event is denoted by the connInterval parameter, which is a multiple of 1.25 ms in the range between 7.5 ms and 4 ms. The connSlaveLatency is also one of the important, which defines the number of consecutive connection events during which the slave is not required to listen to the master and thus can keep the radio turned off. This parameter is an integer between 0 and 499 and must not cause a supervision timeout. When the time since the last received packet exceeds the connSupervisionTimeout parameter, a supervision timeout will happen. This parameter can range from 100 ms to 32 s. To detect the loss of a connection due to severe interference or the movement of a device outside the range of its peer, is the purpose of this mechanism. The data channel packet header contains two one-bit fields: the Sequence Number (SN) and the Next Expected Sequence Number (NESN). The SN determine the packet, while the NESN indicates which packet from the peer device should be received next. Whenever a device successfully receives a data channel packet, the NESN of its next packet will be incremented, and that packet will serve as an acknowledgment. Alternatively, if a device receives a packet with an invalid CRC check, the NESN of the received packet cannot be relied upon. This force the receiving device to resend its last transmitted packet. This mechanism serves as a negative acknowledgment.

#### 2.5.1.3 L2CAP

The L2CAP of the classical Bluetooth is optimized and simplified to be used in BLE. In BLE, the main objective of L2CAP is to multiply the data of three higher layer protocols, ATT, SMP, and Link Layer control signaling, on top of a Link Layer connection. The maximum payload size of L2CAP is equal to 23 bytes in BLE, thus avoiding the use of segmentation and reassembly capability.

#### 2.5.1.4 ATT

The ATT describes the communication between two devices paying the roles of server and client respectively. The server manages set of attributes. An attribute is a data structure that stores the information controlled by the GATT, the protocol that operates on the top of the ATT. The client of server role is identified by the GATT and is independent of the master or slave role.

#### 2.5.1.5 GATT

The GATT characterize a framework that uses the ATT for the discovery of services and the exchange of characteristics from one device to another. The characteristics are nothing but a set of data which includes a value and properties. The data relevant to services and characteristics are stored in attributes. For example, a server that performs a ‘temperature sensor’ service may present with a ‘temperature’ characteristic that uses an attribute for describing the sensor, another attribute for storing temperature measurement values and a further attribute for the indication of the measurement units.

#### 2.5.1.6 Security

BLE incorporates various security services to protect the information exchange between two connected devices. The supported security services can be manifested in terms of two mutually-exclusive security modes called LE Security Mode 1 and LE Security Mode 2. These two modes provide security functionality at the ATT layer and Link Layer, respectively. In a connection, when encryption and authentication are used, a 4-byte Message Integrity Check (MIC) is appended to the payload of the data channel PDU (refer Figure 8). Encryption is then employed to the PDU payload and MIC fields.

#### 2.5.1.7 GAP and Application Profiles

GAP makes the highest level of the core BLE stack. This specifies device roles, modes, and procedure for the discovery of devices and services, the management of security and connection establishment. The BLE GAP modulates four roles with specific requirements on the underlying controller: Broadcaster, Observer, Peripheral and Central. When a device is performing Broadcaster role, it only broadcasts data (via the advertising channels) and does not support connections with other devices. The Observer is the counterpart of the Broadcaster, i.e., it has the responsibility of receiving the data transmitted by the Broadcaster. The Central role is formalized for a device that is in charge of initiating and managing multiple connections, while the Peripheral role is formalized for a simple device which uses a single connection with a device in the Central role. A device may approve various roles, but only one role can be adopted at a given time. A highest-level profile that establishes how applications can interoperate is called an application profile.

## 2.6 Literature Review

### 2.6.1 Simulation of wireless networks

A survey of various simulation tools for wireless networks available for the developers is done in [1]. Evaluating the behavior and performance of wireless protocols in a simulation environment, depends greatly on choosing the correct tool. An appropriate tool must have a good compromise between complexity in one hand and accuracy of results in other hand [1]. In [1], strengths and weakness of six most “widely used” network simulator is been discussed. In particular, the [7] introduces a new framework called MiXiM to provide direct support and a concise modeling chain for wireless communication. MiXiM provides a detailed model of wireless channels, wireless connectivity, mobility models and models for obstacles. This also has many communication protocols especially at the MAC level [7]. An IEEE 802.11g MAC and PHY simulation model in the OMNeT++ simulation environment is been introduced in [12]. This model also has a debugging solution using network protocol analyzer. The model in [12] also considers transmitter output power, modulation and data rates. Furthermore, in [13], an extensive measurement study of wireless networks are done and these results are used to validate the accuracy of various IEEE 802.11g model on OMNeT++. The experiments are conducted in highly controlled and almost error free environment [13]. The lack of the model for effect of interference from co-located devices using the different technologies is been addressed in [14]. Methodology of modelling external interference, by taking the realistic characteristics interference generated by co-located wireless standards, is been proposed in [14].

Complete simulation environment for Bluetooth is been developed in ns (network simulator), an open source simulation environment [15]. This Bluetooth simulation environment support dynamic topology construction, enhance the behavioral control of device, provide animated simulation results, mobility models and support scatternets [15]. The simulation environment in [15] is an extended package for BlueHoc of ns. In addition, in [16] a Bluetooth network simulator that let user to work both on application level and on link management level is been presented. This Bluetooth simulator has been implemented in SystemC [16]. In the paper [17], simulation and security analysis of Bluetooth pairing protocol for numeric comparison using Elliptic Curve Diffie Hellman in NS-2 is presented. The BLE protocol is implemented in OMNeT++ environment for the purpose of studying the network performance [18]. The results in terms of throughput and energy consumption is presented [18].

### 2.6.2 Evaluation of wireless technologies

The presence of numerous wireless technology each having their own advantages and complexity, made it a tough job to choose the right technology for one application. There is lot of research work focusing mainly on this issue, has been conducted. The evaluation is done in many levels and forms concentrating on different performance metrics. In [19], an analytical energy model for secure communication among multi-mode terminals has been introduced. The model in [19] describes the energy consumption of mobile terminals operating inside a dynamic network considering secure data exchange issues. Simulation in ns framework is used to validate the model introduced in [19]. Interestingly, [20] presents a new wireless standard which built to address the point that a complex and heavy networking protocol is not necessary for simple point-to-point and multi point-to-point applications like PC mice. To demonstrate the effectiveness of the proposed Wireless USB protocol, ns-2 based simulation is used in this work [20]. The paper [20] also claims the protocol outperforms Bluetooth in terms of packet delivery ratio, latency, and power consumption . Bluetooth and IEEE 802.11 which are the two main communication protocol standards has been compared in [21]. The survey and comparison in [21] is made in terms of various metrics, including capacity, network topology, quality of service support, and power consumption. Additionally, the study in [22] investigates BLE utilization for transmitting the occupancy data to server. The paper [22] focus on monitoring and comparing the energy consumption of mobile phones when performing data transmission via WiFi and BLE. In the paper [23], an analytical model of the time, takes a Bluetooth based moving observer to discover a Bluetooth device within a floating traffic object is explained. Bluetooth EDR physical layer is modelled and simulated in Mathworks MATLAB [24]. Simulation for 1, 2, and 3 Mbits/s data rates are presented. The paper [25], presents an analysis of the Bluetooth physical layer in office room environment, focusing mainly on the interference between piconets. The research work in [26], proposes an analytical model to predict the delay of the transmission in Bluetooth piconets employing Serial Port Profile (SPP). The document from Nordic semiconductors [27], gives a brief introduction on Bluetooth Low Energy wireless technology. In the paper [28], a comparison in terms of the maximum peer-to-peer throughput, the minimum frame turnaround time, and the energy consumption for three protocols, is studied. The protocols considered by [28] are BLE, IEEE 802.15.4 and SimplicitTI. SimpliciTI is a proprietary protocol developed by Texas Instruments. The results from [28] reveal that BLE can potentially support maximum LL (link Layer ) data throughput of around 320 lbits/s. The study in [28] also shows that the BLE technology is capable of providing a frame turnaround time of less than one millisecond. Further, in [29] focus is been given in improving throughput and efficiency of BLE in a multi node environment. This research paper [29] proposes methods to improve throughput and energy efficiency. In particular, [30] presents an energy model to predict the energy consumption of BLE – based wireless devices in all possible operating modes. The accuracy of the energy model in [30] is evaluated using both discrete event simulation and actual measurements. Furthermore, the research paper [31] presents an analytical model for the maximum throughput of BLE taking the impact of important BLE parameter into account. The analytical model in [31] is derived as function of BER and connection interval. Then in the research paper [32], an analytical model to investigate the discovery probability and the expected discovery latency, has been derived. The model in [32] is validated via experiments.

# 3 System Model & Problem Description

In this chapter system components and assumptions related to multi-device environment will be named, followed by a formal specification of the problem to be solved.

## 3.1 System Model

Multi-device environment in a simple sense describes the system with many “Smart Things” connected to a central brain. In specific, in our system, multiple devices or things communicate wirelessly with a central or master devices. Bluetooth Low Energy (BLE) is the wireless communication protocol in this system.

Modelling of the parameters of BLE environment for a multi-device environment will be achieved first. Secondly, the BLE sensor network model is simulated in OMNeT++ network simulator. The validation of the simulated system can be done by implementing the same system in real world hardware.

The system will consist of many BLE slaves or peripheral devices whose objective is to communicate the data with a BLE master or central device. Bluetooth Low Energy follows a certain protocol on how a slave can communicate data to the master. The Master and slave were not synchronized to one another.

Initially, all peripheral devices will be in advertising mode and the central device will be in scanning (or) initiating mode. Once the connection is made, central device will become the master and peripheral devices will become the slaves. The slaves will send the data to the master device via a secured data channel.

In BLE, during advertising mode, all the advertisers use only 3 advertising channel and their indexes are 37, 38 and 39. The scanner (or) initiator will be listening for any possible advertisements only in these advertising channels. For simplicity, this phase will be called advertising and device discovery phase. Device discovery is when the central device receives the advertisement and respond to the corresponding advertiser with a connection setup request.
Once the advertiser accepts the connection request, it sends a response to the initiator. During the connection phase, both slave and master communicate the data using 37 data channel. Frequency hopping mechanism is used to avoid fading or interference with other radio in the same channel.
Different configuration parameters are there in BLE core specification to control these 2 above-mentioned phases. During advertising and device discovery phase, the configuration parameters such as advertising Interval, scanInterval and scanWindow play a considerable role. During connection phase, the configuration parameters such as connection interval, supervision timeout,

and connection latency play a major role. These parameters will have their own sensitivity which impact the performance metrics of the BLE environment.

## 3.2 Problem Statement

- Designing the simulation model which can achieve an abstraction of the real BLE network behavior.
- Implementation of the system which fully accords with BLE core specification in OMNeT++ simulator environment. OMNeT++ is a module based discrete event simulator. In order to realize the BLE protocol MiXiM framework will be used in OMNeT++. MiXiM framework has many compound modules to represent various wireless components.
- The performance metrics of a system with 1 central device and with M number of peripheral devices can be calculated using the simulation model
- The validation of the simulation model requires a hardware implementation and comparing the result with the results obtained from simulation model
- The hardware implementation involves a BLE gateway to act as central or master device and a countable number of batteries operated BLE devices to act as peripheral or slave devices.
- The investigation on the possible relationship between a number of slave devices and their impact on system performance metrics will be done.
- Calculating the deviation in simulation model compared with hardware and investigating the reason for the deviation

# 4 Implementation

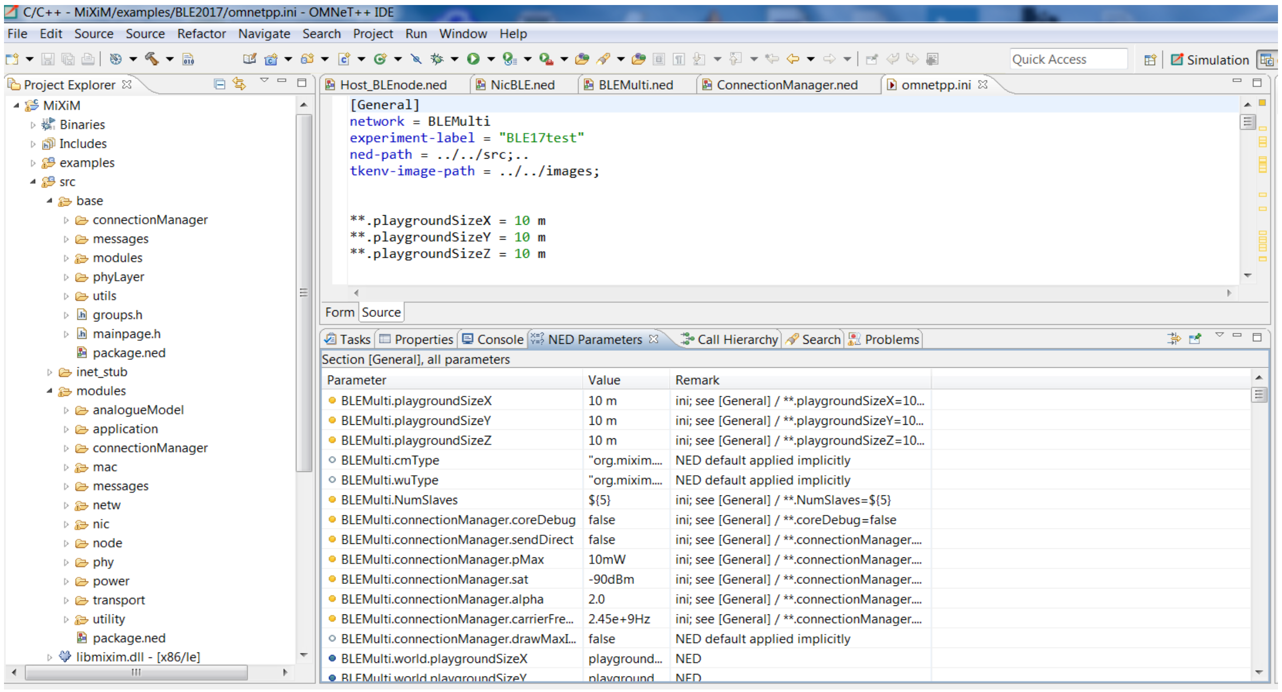
## 4.1 Simulation model in OMNeT++

The modeling of BLE in OMNeT++ with the help of MiXiM framework is the first goal of the thesis work. The simulation model in OMNeT++ is achieved by implementing the lower layers of the BLE communication protocol. To implement the lower layers, we use the existing base layer implementations in MiXiM framework. To represent the basic functionality of BLE, we decided to create models of Physical layer, Link layer, and Network layer specific to BLE.

### 4.1.1 Setting up OMNeT++ environment

The first step in the implementation would be to install OMNeT++ 4.0, open source framework for the network simulators. The document [33] describes how to install the IDE in various platforms. Figure 10 shows the OMNeT++ IDE with MiXiM installed. MiXiM framework can be installed in OMNeT ++ IDE by following few steps:

- Choose "File->Import" from the menu.
- Choose "General->Existing Projects into Workspace" from the upcoming dialog and proceed with "Next".
- Choose "Select archive file" and select the MiXiM archive file.
- "MiXiM" should appear in the "Projects" list below. Click "Finish".
- To build MiXiM, right-click on the project and choose "Build Project".



```
[General]
network = BLEMulti
experiment-label = "BLE17test"
ned-path = ../../src;..
tkenv-image-path = ../../images;


**.playgroundSizeX = 10 m
**.playgroundSizeY = 10 m
**.playgroundSizeZ = 10 m
```

Section [General], all parameters

| Parameter | Value | Remark |
|---|---|---|
| ● BLEMulti.playgroundSizeX | 10 m | ini; see [General] / **.playgroundSizeX=10... |
| ● BLEMulti.playgroundSizeY | 10 m | ini; see [General] / **.playgroundSizeY=10... |
| ● BLEMulti.playgroundSizeZ | 10 m | ini; see [General] / **.playgroundSizeZ=10... |
| ○ BLEMulti.cmType | "org.mixim.... | NED default applied implicitly |
| ○ BLEMulti.wuType | "org.mixim.... | NED default applied implicitly |
| ● BLEMulti.NumSlaves | ${5} | ini; see [General] / **.NumSlaves=${5} |
| ● BLEMulti.connectionManager.coreDebug | false | ini; see [General] / **.coreDebug=false |
| ● BLEMulti.connectionManager.sendDirect | false | ini; see [General] / **.connectionManager.... |
| ● BLEMulti.connectionManager.pMax | 10mW | ini; see [General] / **.connectionManager.... |
| ● BLEMulti.connectionManager.sat | -90dBm | ini; see [General] / **.connectionManager.... |
| ● BLEMulti.connectionManager.alpha | 2.0 | ini; see [General] / **.connectionManager.... |
| ● BLEMulti.connectionManager.carrierFre... | 2.45e+9Hz | ini; see [General] / **.connectionManager.... |
| ○ BLEMulti.connectionManager.drawMaxI... | false | NED default applied implicitly |
| ● BLEMulti.world.playgroundSizeX | playground... | NED |
| ● BLEMulti.world.playgroundSizeY | playground... | NED |

*Figure 10: OMNeT++ IDE with MiXiM installed*

### 4.1.2 Physical Layer Module

The physical layer is the lowest of all the layers in the BLE protocol from the figure (3). Physical layer module in OMNeT ++ has the task of initializing relevant Analogue models and Deciders. There are various Analogue models and Deciders modules available within the MiXiM module directory. Analogue models present within library are

1. SimplePathlossModel
2. LogNormalShadowing
3. JakesFading
4. BreakpointPathlossModel
5. PERModel

Similarly, Deciders present within library are

1. Decider80211
2. SNRThresholdDecider
3. Decider802154Narrow

For the BLE implementation, SimplePathlossModel is used. This model represents a Pathloss-function. In this simple path loss implementation, one attenuation value is assumed to be constant over the signals duration. With the help of a config.XML, we select the parameters and map values to them. For our SimplePathlossmodel, parameter alpha (minimum path loss coefficient) is selected as 4. The Carrier frequency of the signal is given in Hz. For BLE we choose the carrier frequency as 2.412e+9 Figure 11.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
      <AnalogueModels>
            <AnalogueModel type="SimplePathlossModel">
            <parameter name="alpha" type="double" value="4.0"/>
            <parameter name="carrierFrequency" type="double" value="2.450e+9"/>
          </AnalogueModel>
      </AnalogueModels>

      <Decider type="SNRThresholdDecider_Multichannel">
<parameter name="snrThreshold" type="double" value="1.12589254117942"/>
<parameter name="busyThreshold" type="double" value="3.98107170553E-9"/>
      </Decider>
</root>
```

*Figure 11: config.xml*

For BLE implementation, SNRThresholdDecider_Multichannel is used. This is an extension of the standard SNRThresholdDecider for enabling it to work with multiple radio channels. SNRThresholdDecider decides the channel state (idle/busy) at hand of the current received total power level (independent from signal or noise). If it is above the threshold defined by the "busyThreshold" parameter, it considers the channel busy. The RSSI value returned by this

Decider for a ChannelSenseRequest over time is always the RSSI value at the end of the sense. With the help of the same config.XML file, we map SNR threshold to a value of 1.12589254. The parameter “busyThreshold” is mapped to a value of 3.9810E-9.

The SNR-mapping for the Signal is created and checked against the Deciders SNR-threshold. Depending on that the received AirFrame is either sent up to the MAC-Layer or dropped.

### 4.1.3 MAC Layer Module

MAC layer module implementation provides an abstraction to the Link layer of BLE protocol. Unlike Physical layer module, this MAC layer module can only inherit very few properties from the existing base layer. The BLEMacLayer is an extension of the BaseMacLayer. The BaseMacLayer module provides functionalities such as decapsulation and encapsulation of messages using the standard address. It also provides basic handling of lower layer messages. The BaseMacLayer is an extension of BaseLayer in MiXiM library. This BaseLayer module provides a basic abstraction that eases development of a network or Mac layer.

The BLEMac.ned declares parameters such as bitrate, header length, transmission power, and state of the node (standby, advertising, initiating, and connection). It also declares some of the BLE link layer specific parameters such as length of the header for the data packet and advertising packet, Interframe spacing, maximum advertising PDU duration, maximum data PDU duration, minimum data PDU duration, and maximum data PDU payload bytes. We also declared parameters to include the hardware switching timings like time taken to switch from sleep mode to transmission mode, from sleep mode to receiving mode, from transmission mode to receiving mode, and from receiving mode to transmission mode. All these parameters can be mapped to a value defined in the core specification. The parameters for debugging and testing were also initialized in BLEMac.ned file.

BLE Mac layer’s C++ source has necessary classes to realize relevant functionality in Link layer and L2CAP. Initialization will be executed as soon as the control of the node enters the MAC layer. There are two stages of initialization. During stage 0 all the relevant timers and objects are initialized. This happens only once during the simulation. In stage 1, depending on the initState different function calls are made. For example, when the node is acting as an advertiser, the initState value will be equal to 1, then the MAC state gets updated to Advertising and a timer to start the advertisement will be triggered. Figure 12 shows the initialization of different parameters based on the value of initState.

During the connection state, a new data packet is generated with the function call prepareNewDataPkt, to use in transmission. During connection state, connection info plays a role in deciding when to close the connection event.

```
         if(initState==1){//act as advertiser
            updateMacState(ADVERTISING_2);
            updateMacSubstate(UNUSED);
           startTimer(TIMER_ADVERTISE_EVENTSTART);
        }
        else if(initState==2){//act as initiator
            updateMacState(INITIATING_4);
            updateMacSubstate(UNUSED);
           startTimer(TIMER_INITIATION_WAKEUP);
        }
        else if(initState==3){//act as a slave (starting from CON_REQUEST)
            updateMacState(CONNECTION_5);
            updateMacSubstate(CONNECTED_SLV_WAITCONNECTION);
            initVariablesAtStateStart();
            prepareNewDataPkt(1);
        }
        else if(initState==4){//act as a master (starting from CON_REQUEST)
            updateMacState(CONNECTION_5);
            updateMacSubstate(CONNECTED_MST_SLEEP);
            initVariablesAtStateStart();
            prepareNewDataPkt(1);
        }
        else if(initState==5){//act as a slave (already established)
            updateMacState(CONNECTION_5);
            updateMacSubstate(CONNECTED_SLV_SLEEP);
            myConnPars.ConnInfo.numMissedEvents=0;
   myConnPars.ConnInfo.lastunmappedChannel=myConnPars.ConnInfo.unmappedChannel;
            myConnPars.ConnInfo.transmitSeqNum=false;
            myConnPars.ConnInfo.nextExpectedSeqNum=false;
            myConnPars.ConnInfo.delayedGeneration=true;
            myConnPars.ConnInfo.moreData=false;
            startTimer(TIMER_CONNECTION_SLAVE_WAKEUP);
            startTimer(TIMER_CONNECTION_SLAVE_SUPERVISION);
            myConnPars.ConnInfo.timeLastAnchorReceived=0;
            prepareNewDataPkt(1);
            eventConnectionCompleted();
        }
        else if(initState==6){//act as a master (already established)
            updateMacState(CONNECTION_5);
            updateMacSubstate(CONNECTED_MST_SLEEP);
            myConnPars.ConnInfo.numMissedEvents=0;
myConnPars.ConnInfo.lastunmappedChannel=myConnPars.ConnInfo.unmappedChannel;
            myConnPars.ConnInfo.transmitSeqNum=false;
            myConnPars.ConnInfo.nextExpectedSeqNum=false;
            myConnPars.ConnInfo.delayedGeneration=true;
            myConnPars.ConnInfo.moreData=false;
            myConnPars.ConnInfo.timeLastSuccesfullEvent=0;
            startTimer(TIMER_CONNECTION_MASTER_WAKEUP);
            startTimer(TIMER_CONNECTION_MASTER_SUPERVISION);
            prepareNewDataPkt(1);
            eventConnectionCompleted();
        }
```

*Figure 12: Initialization of variables based on initState*

BLE protocol layers communicate between them using the message packets. These message packets can be defined by .msg files in OMNeT++. Figure 13 and Figure 14 describes those BLE advertising and data MAC packets. The advertising and data MAC packet is an extension of base MAC packet from MiXiM library. The MAC packet fields like access address, advertising PDU type, and transmitter address, receiver address, interval, latency, sequence number, next expected sequence number, more data, and etc, are described in the core specification [10].

```
packet BLE_Adv_MacPkt extends MacPkt
{
    long                    AccessAddress;    //preamble
    int                     Adv_PDU_type;
    bool                    TxAdd;
    bool                    RxAdd;
    int                     Length;
    LAddress::L2Type AdvA;
    LAddress::L2Type InitA;   //ADV_DIRECT, CONNECT_REQ
    long AA;
    int WinSize;
    int WinOffset;
    int Interval;
    int Latency;
    int Timeout;
    int MapCh0to7;
    int MapCh8to15;
    int MapCh16to23;
    int MapCh24to31;
    int MapCh32to39;
    int Hop;
    int SCA;
    LAddress::L2Type ScanA; // for SCAN_REQ
}
```

*Figure 13: BLE_Adv_MacPkt.msg*

```
packet BLE_Data_MacPkt extends MacPkt
{
    long AccessAddress;
    int      hdr_LLID;
    bool hdr_NESN;
    bool hdr_SN;
    bool hdr_MD;
    int      hdr_lgth;
    int Opcode  //Commands
    int      Instant;     //LL_CHANNEL_MAP_REQ, LL_CONNECTION_UPDATE_REQ
    int WinSize;    //link layer channel map request fields see spec v4.1
    int WinOffset;
    int Interval;
    int Latency;
    int Timeout;
int MapCh0to7;      //Link layer connection update request fields see spec v4.1
    int MapCh8to15;
    int MapCh16to23;
    int MapCh24to31;
    int MapCh32to39;
    int ErrorCode;   //LL_TERMINATE_IND

}
```

*Figure 14: BLE_Data_MacPkt.msg*

The BLE MAC layer handles the messages from both upper and lower layers. Based on different messages, different event will be executed. The BLE MAC layer also handles the message generated within the same layer. These message could be a timer or an event completion message or state transition event. Figure 15 shows the self-message handler of BLE MAC layer.

```
void BLEMacV2::handleSelfMsg(cMessage *msg) {

//advertiser
    if(msg==advertisementEventTimer){
        executeMac(EV_ADV_EVENT, msg);
    }
    else if(msg==advertisementNextPDUTimer){
        executeMac(EV_ADV_NEXTPDU, msg);
    }
    else if(msg==advertisementEndEvent){
        executeMac(EV_ADV_EVENTEND, msg);
    }

//initiator
    else if(msg==initiatingScanIntervalTimer){
        executeMac(EV_INIT_INTERVAL, msg);
    }
    else if(msg==initiatingScanWindowTimer){
        executeMac(EV_INIT_WINDOW, msg);
    }

//slave in connection
    else if(msg==slaveConnectionWakeupConnectionTimer){
        executeMac(EV_CON_SLV_WAKEUP_CONNECTION, msg);
    }
    else if(msg==slaveConnectionTransmitWindowTimer){
        executeMac(EV_CON_SLV_TRANSMIT_WINDOW, msg);
    }
    else if(msg==slaveConnectionNoBeaconTimer){
        executeMac(EV_CON_SLV_NOBEACON, msg);
    }
    else if(msg==slaveConnectionWakeupTimer){
        executeMac(EV_CON_SLV_WAKEUP, msg);
    }
    else if(msg==slaveIFSTimer){
        executeMac(EV_CON_SLV_WAITIFS, msg);
    }
    else if(msg==slaveWaitReplyTimer){
        executeMac(EV_CON_SLV_WAITREPLY, msg);
    }
    else if(msg==slaveEndConnectionEvent){
        executeMac(EV_CON_SLV_ENDEVENT, msg);
    }
    else if(msg==slaveConnectionSupervisionTimer){
        executeMac(EV_CON_SLV_DROPCONNECTION, msg);
    }
//master in connection
    else if(msg==masterConnectionWakeupTimer){
        executeMac(EV_CON_MST_WAKEUP, msg);
    }
    else if(msg==masterWaitReplyTimer){
        executeMac(EV_CON_MST_WAITREPLY, msg);
    }
    else if(msg==masterIFSTimer){
        executeMac(EV_CON_MST_WAITIFS, msg);
    }
    else if(msg==masterEndConnectionEvent){
        executeMac(EV_CON_MST_ENDEVENT, msg);
    }
    else if(msg==masterConnectionSupervisionTimer){
        executeMac(EV_CON_MST_DROPCONNECTION, msg);
    }
    //state switching stuff
    else if(msg==ctrl_switchState){
        stopAllTimers();
        if(msg->getKind()==INITIALIZING_to_CONNECTION){
            updateMacState(CONNECTION_5);
            updateMacSubstate(CONNECTED_MST_SLEEP);
      simtime_t FreeTime=phy->setRadioState(MiximRadio::SLEEP);
      if(FreeTime==0) eventRadioStateChanged();
            initVariablesAtStateStart();
            eventConnectionCompleted();
         }
       else if(msg->getKind()==ADVERTIZING_to_CONNECTION){
            updateMacState(CONNECTION_5);
            updateMacSubstate(CONNECTED_SLV_WAITCONNECTION);
             simtime_t FreeTime=phy->setRadioState(MiximRadio::SLEEP
      if(FreeTime==0) eventRadioStateChanged();
            initVariablesAtStateStart();
            eventConnectionCompleted();
         }
       else if(msg->getKind()==ANY_to_STANDBY){
           if(phy->getRadioState()==MiximRadio::SWITCHING){
                Flag_DelayedSwitchOff=true;
            else if(phy->getRadioState()!=MiximRadio::SLEEP){
              simtime_t FreeTime=phy->setRadioState(MiximRadio::SLEEP);
              if(FreeTime==0){
                     eventRadioStateChanged();
                     //inform host
               myEventData->Connection_Handle=myConnPars.ConnInfo.connectionHandle;
                     myEventData->ErrorCode=myCmdError->ErrorCode;
                     cMessage *m;
                     m = new cMessage("BLE_MACtoNWK_EVENT");
m->setControlInfo(BLE_MacToNwk::generate_DisconnectionCompleteEvent(myEventData));
                     sendControlUp(m);
                     Flag_DelayedSwitchOff=false;
                     updateMacState(STANDBY_1);
                     updateMacSubstate(UNUSED);
                    simtime_t FreeTime=phy->setRadioState(MiximRadio::SLEEP
       if(FreeTime==0) eventRadioStateChanged();
                     initVariablesAtStateStart();

                }
else if(msg->getKind()==STANDBY_to_ADVERTIZING){
            updateMacState(ADVERTISING_2);
            updateMacSubstate(UNUSED);
            initVariablesAtStateStart();
        }
        else if(msg->getKind()==STANDBY_to_INITIATING){
            updateMacState(INITIATING_4);
            updateMacSubstate(UNUSED);
            initVariablesAtStateStart();
        }
    }
    else if(msg==ctrl_terminateConnection){//terminate connection
        dropConnection();
    }
    else error("BLEMacV2::handleSelfMsg unknown message");
}
```

Figure 15: Handler for self-generated messages

Any BLE node based on their role (advertiser, initiator, master, slave), performs specific tasks which are defined in their MAC layer. An advertiser transmits an advertisement packet of a specific type in a specified time interval. After sending the advertisement packet, it turns its receiver on to receive any response from an initiator or a scanner. The advertiser then becomes the slave, by going into connection state, after receiving a connection request from Initiator. An Initiator will always listen for an advertisement packet. After receiving an advertisement packet, the initiator transmits connection request packet and becomes the master. During the connection state, an advertiser plays slave role and an initiator performs the master role. The function call updateStatusAdvertising executes all the events relevant to an advertiser. The function call updateStatusInitiating executes all the events relevant to an initiator. The function call updateStatusconnected executes all the events relevant to both master and slave. Figure (16) shows these function calls declaration in the BLE MAC header file.

```
    virtual void updateMacState(t_mac_states newMacState);
    virtual void updateMacSubstate(t_mac_substates newMacSubstate);
    virtual void updateStatusStandby(t_mac_event event, cMessage *msg);
    virtual void updateStatusAdvertising(t_mac_event event, cMessage *msg);
    virtual void updateStatusInitiating(t_mac_event event, cMessage *msg);
    virtual void updateStatusConnected(t_mac_event event, cMessage *msg);
```

*Figure 16: function call declaration in the BLE MAC header file*

MAC layer source file also has function definitions for the preparation of new data packet, connection terminate packet, channel map update packet, connection parameter update packet, and empty acknowledgment packet and also for the generation of scan response packet, scan request packet, and connection request packet. These message packets are used to describe the accurate depiction of BLE MAC layer in OMNeT++.

### 4.1.4 Network Layer Module

BLE network layer module is implemented as an extension of the Base network layer. Network layer commands the MAC layer module to initialize. This layer makes use of the HCI messages to handle messages coming from the lower layer. Figure 17 shows the graphical view of the BLE network topology. Source editor of the BLE_Nwk.ned has the initialization of parameter values like advInterval, scanInterval, scanWindow, connLatency, connInterval, supervision timeout, and connection channel map.

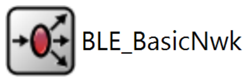


*Figure 17: Design view of the BLE_Nwk.ned*

Similar to the MAC layer module, network layer module makes the initialization of parameters and objects when the control of BLE node is in network layer. This layer also handles the message from lower and upper layers. It can also handle the self-generated message.

```
void BLE_BasicNwk::handleLowerControl(cMessage *msg){

    cObject* cInfo = msg->removeControlInfo();
    BLE_MacToNwk *const cEventInfo = dynamic_cast<BLE_MacToNwk *const>(cInfo);
    switch(cEventInfo->get_EventType()){
    case cBLEstructs_defs::BLECMD_HCICtrl_LE_Connection_Complete_Event:
        currentCmdConnectionPars->ConnHandle=cEventInfo->get_EventsParsPtr()-
>Connection_Handle;
        if(cEventInfo->get_EventsParsPtr()->Role==cBLEstructs_defs::MASTER_0){
if(useSuggestedFastIntervalSturtupMechanism==true){//modify the parameters
              currentCmdConnectionPars->Conn_Interval_Max=par("connInterval");
              currentCmdConnectionPars->Conn_Interval_Min=par("connInterval");
              currentCmdConnectionPars-
>Supervision_Timeout=par("supervision_Timeout");
```

```
               cMessage *m1;
               m1 = new cMessage("BLE_NWKtoMAC_CMD");
               m1-
>setControlInfo(BLE_NwkToMac::generate_LEConnectionUpdateCommand(currentCmdConnect
ionPars));
               sendControlDown(m1);
           }
        break;
    case cBLEstructs_defs::BLECMD_HCICtrl_LE_Connection_Update_Complete_Event:
        EV << "BLE_BasicNwk: BLECMD_HCICtrl_LE_Connection_Update_Complete_Event
Connection Handle:" << cEventInfo->get_EventsParsPtr()->Connection_Handle << endl;
        break;
    case cBLEstructs_defs::BLECMD_HCICtrl_Disconnection_Complete_Event:
        EV << "BLE_BasicNwk: BLECMD_HCICtrl_Disconnection_Complete_Event
Connection Handle:" << cEventInfo->get_EventsParsPtr()->Connection_Handle <<"
ErrorCode:" << cEventInfo->get_EventsParsPtr()->ErrorCode << endl;
        scheduleAt(simTime(), StartNode);//restart all
        break;
     }    delete msg;
}
```

Figure 18: Handler for the lower control message

Figure 18 shows the handler of the control message coming from the lower layer. Based on the HCI control command different function call will be executed. For example BLECMD_HCICtrl_Disconnection_Complete_Event restarts all the BLE node by executing StartNode. Figure 19 describes the function call StartNode. During the start of the node, depending on the role, the parameter will redefined.

```
void BLE_BasicNwk::startNode(void){
    int Role=par("NWK_DEBUG_ROLE");
     if(Role==1){//slave
         // set advertisement parameters
         int myIdx=FindModule<>::findHost(this)->getIndex();
         currentCmdAdvertisePars->Adv_Type=cBLEstructs_defs::ADV_IND_0;
         currentCmdAdvertisePars->Direct_Address=(LAddress::L2Type)(myIdx);
         currentCmdAdvertisePars-
>Direct_Address_Type=cBLEstructs_defs::DIRECT_ADDR_PUBLIC_0;
         currentCmdAdvertisePars-
>Own_Address_Type=cBLEstructs_defs::OWN_ADDR_PUBLIC_0;
currentCmdAdvertisePars->Advertising_Interval_Max=par("advInterval");
         currentCmdAdvertisePars->Advertising_Interval_Min=par("advInterval");
         currentCmdAdvertisePars->Advertising_Filter_Policy=cBLEstructs_defs::ANY_0;
         cMessage *m;
         m = new cMessage("BLE_NWKtoMAC_CMD");
         m-
>setControlInfo(BLE_NwkToMac::generate_LESetAdvertisingParametersCommand(currentCmdAd
vertisePars));
         sendControlDown(m);
        // set advertisement data
         cMessage *m1;
         m1 = new cMessage("BLE_NWKtoMAC_CMD");
         m1-
>setControlInfo(BLE_NwkToMac::generate_LESetAdvertisingDataCommand(currentCmdAdver
tisePars));
         sendControlDown(m1);
         // start advertisements
         currentCmdAdvertisePars->AdvEnabled=true;
         cMessage *m2;
         m2 = new cMessage("BLE_NWKtoMAC_CMD");
         m2-
>setControlInfo(BLE_NwkToMac::generate_LESetAdvertiseEnableCommand(currentCmdAdver
tisePars));
         sendControlDown(m2);
     }
     else if(Role==2){
         //set required channels
         currentCmdAdvertisePars->AdvPacketLgth=0;

         cMessage *m;
         m = new cMessage("BLE_NWKtoMAC_CMD");
         m-
>setControlInfo(BLE_NwkToMac::generate_LESetAdvertisingParametersCommand(currentCm
dAdvertisePars));
         sendControlDown(m);
         int myIdx=FindModule<>::findHost(this)->getIndex();
        // set scanning connection parameters
         currentCmdScanningPars->LE_Scan_Interval=par("scanInterval");
         currentCmdScanningPars->LE_Scan_Window=par("scanWindow");
         currentCmdScanningPars-
>Own_Address_Type=cBLEstructs_defs::OWN_ADDR_PUBLIC_0;
         currentCmdScanningPars->Peer_Address=(LAddress::L2Type)(myIdx);
         currentCmdScanningPars-
>Peer_Address_Type=cBLEstructs_defs::PEER_ADDR_PUBLIC_0;
         // set desired connection parameters
         currentCmdConnectionPars->Conn_Latency=par("connLatency");
   if(currentCmdConnectionPars->Supervision_Timeout<10)currentCmdConnectionPars-
>Supervision_Timeout=10;//see spec v4.1
         bool
useSuggestedFastIntervalSturtupMechanism=par("useSuggestedFastIntervalSturtupMecha
nism");
         if(useSuggestedFastIntervalSturtupMechanism==true){//set the minimum
interval
             currentCmdConnectionPars->Conn_Interval_Max=8;
             currentCmdConnectionPars->Conn_Interval_Min=8;
             currentCmdConnectionPars->Supervision_Timeout=10;
         }
         else{//set desired interval straight away
             currentCmdConnectionPars->Conn_Interval_Max=par("connInterval");
             currentCmdConnectionPars->Conn_Interval_Min=par("connInterval");
             currentCmdConnectionPars-
>Supervision_Timeout=par("supervision_Timeout");
         }
```

```
          //set used data channels
          cMessage *m0;
          m0 = new cMessage("BLE_NWKtoMAC_CMD");
          m0-
>setControlInfo(BLE_NwkToMac::generate_LESetHostChannelClassificationCommand(curre
ntCmdConnectionPars));
          sendControlDown(m0);
          //switch to initialize state
          cMessage *m1;
          m1 = new cMessage("BLE_NWKtoMAC_CMD");
          m1-
>setControlInfo(BLE_NwkToMac::generate_LECreateConnectionCommand(currentCmdScannin
gPars,currentCmdConnectionPars));
          sendControlDown(m1);
      }
}
```

Figure 19: Funtion call 'StartNode' of BLE network layer

### 4.1.5 BLE Node

The BLE node is a complex module which abstracts a BLE device. This BLE node accords with basic properties of a BLE device as defined by Bluetooth core specification [10].

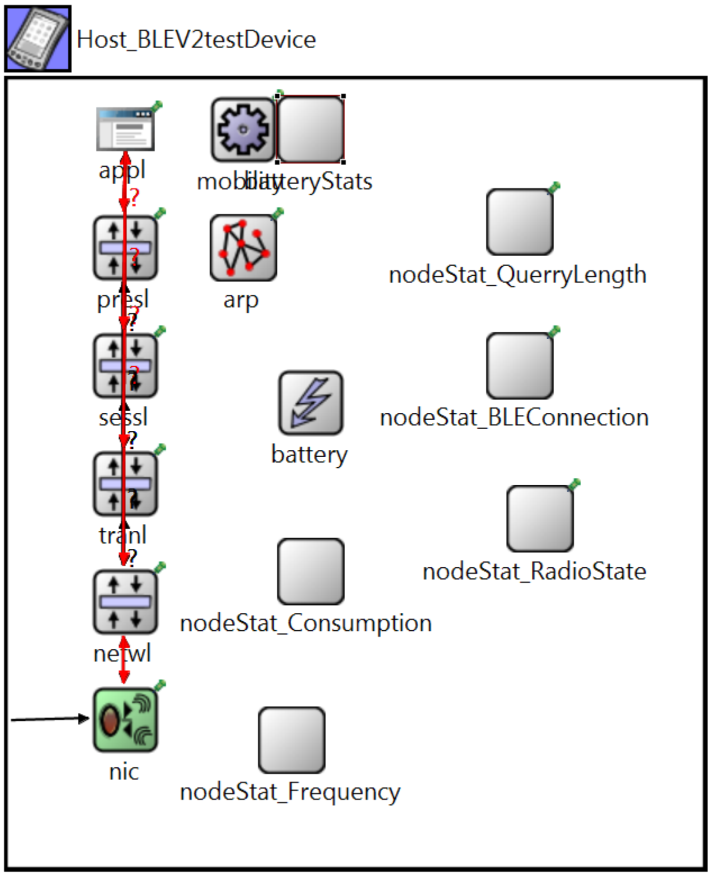


Figure 20: Graphical editor with the topology of BLE node, a complex module

While describing a BLE network, the master and slaves in the network are formed by BLE node which is a complex module defined in OMNeT++.Figure 20 shows the graphical editor with the topology of the BLE node module and shows the hierarchy of different protocol layers from NIC (Network Interface card) to Application layer. Except for NIC and Network layer, the default layers provided by MiXiM library are used. From the Figure 21, it is shown that Physical layer and MAC layer constitutes a NIC. The modules like nodeStat_BLEConnection, nodeStat_RadioState, nodeStat_Consumption, and nodeStat_Frequency are simple modules to log the information during the simulation Figure 20.

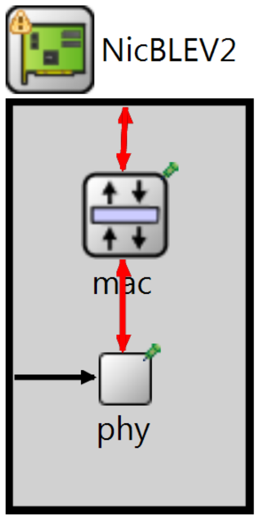


*Figure 21: NIC (Network Interface Card)*

### 4.1.6 BLE network module as in a multi-device environment

Designing a complex network module of a multi-device environment with many slaves and a single master is the last step in the OMNeT implementation. The BLE network module has defined an extension of Base network module. Figure 22 shows the graphical representation of BLE network module. Our BLE module has a single master with a number of slaves. The number of slaves can be defined with omnet.ini file. The connection manager controls all connection related functions. The connection manager periodically communicates with the mobility module and channel access. The world module provides a basic utility for the whole network. The utility includes the playground size in X, Y, and Z directions.

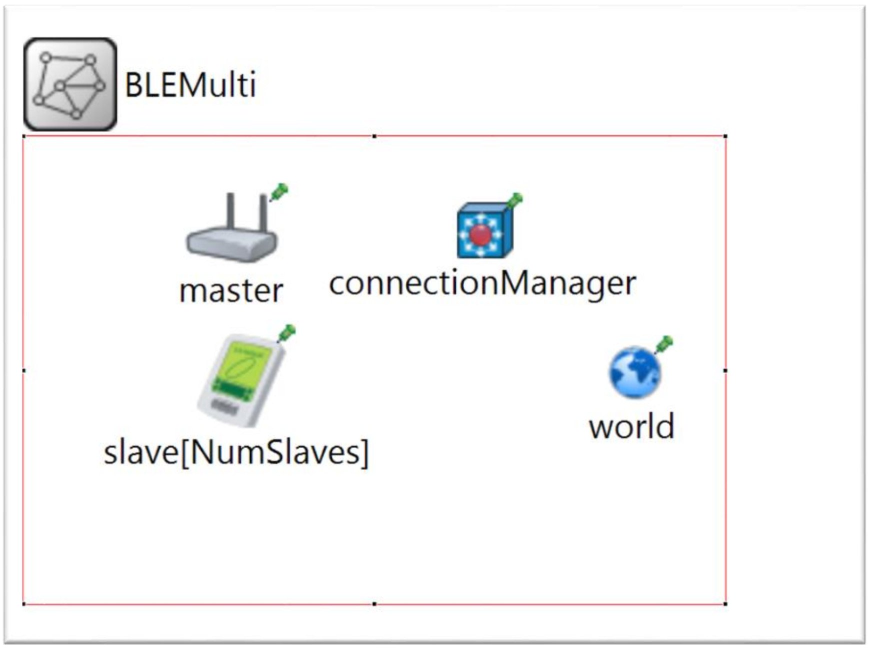


*Figure 22: Graphical editor with topology of BLE network module*

Finally the omnet.ini file describes the parameter settings and the configuration options for the simulation. Figure 23 describes the omnet.ini file for our implementation.

```
[General]
network = BLEMulti
experiment-label = "BLE17test"
ned-path = ../../src;..
tkenv-image-path = ../../images;

**.playgroundSizeX = 10 m
**.playgroundSizeY = 10 m
**.playgroundSizeZ = 10 m
################ MOBILITY #####################
**.mobilityType = "StationaryMobility"
#**.mobilityType = "ConstSpeedMobility"
###########################################################

**.NumSlaves = ${8}
**netwl.advInterval = 160 #in 0.625ms units
**netwl.scanInterval = 160
**netwl.scanWindow = 160
**netwl.connInterval = ${connIntervalVal=10}#in 1.25 ms units
**netwl.connLatency = 0
**netwl.supervision_Timeout = ${connIntervalVal} #8*connInterval
**nic.mac.advertisingAddr = -1 #-1 - advertising address equals to index
**nic.mac.transmitWindowOffset = 0
**nic.mac.transmitWindowSize = 1
**nic.mac.connAccessAddress = 777
**netwl.nodeStartupDelay = 0 s
**master.netwl.initialData = 0 byte
```

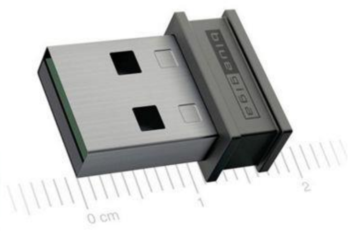
```
**slave[*].netwl.initialData = 40 byte

####################################################################################
############################
#[Config Channel_Hopping_Chart]
#"Channel Hopping chart"
**master.netwl.NWK_DEBUG_ROLE = 2 #initiator/master
**slave[*].netwl.NWK_DEBUG_ROLE = 1 #advertiser/slave

**master.netwl.connDataChannelMap_0to7 = 0x03
**master.netwl.connDataChannelMap_8to15 = 0x00
**master.netwl.connDataChannelMap_16to23 = 0x00
**master.netwl.connDataChannelMap_24to31 = 0x00
**master.netwl.connDataChannelMap_32to39 = 0x00


**netwl.connAdvertiseChannelMap_0to7 = 0x00
**netwl.connAdvertiseChannelMap_8to15 = 0x00
**netwl.connAdvertiseChannelMap_16to23 = 0x00
**netwl.connAdvertiseChannelMap_24to31 = 0x00
**netwl.connAdvertiseChannelMap_32to39 = 0xE0 #default 0xE0
#note - the TX-RX and RX-TX times should be equal to IFS (or at least not exceed
those!)
**.nic.mac.Time_llSLEEPtoTX = 0.000004s
**.nic.mac.Time_llSLEEPtoRX = 0.000004s
**.nic.mac.Time_llTXtoRX = 0.000150s
**.nic.mac.Time_llRXtoTX = 0.000150s
**.nic.mac.Time_llTXtoSLEEP = 0s
**.nic.mac.Time_llRXtoSLEEP = 0s


##########################################################
################ PhyLayer parameters #####################
**.nic.phy.usePropagationDelay = false
**.nic.phy.thermalNoise = -100dBm
**.nic.phy.useThermalNoise = true
**.nic.phy.analogueModels = xmldoc("config.xml")
**.nic.phy.decider = xmldoc("config.xml")

**.nic.phy.initialRadioState = 2 #i.e., sleep
**.nic.phy.sensitivity = -93 dBm #CC2540 datasheet (SWRS084F) p.5 - High-gain mode
**.nic.phy.maxTXPower = 10.0 mW #according to BLE standard & Ficora regulations
**.nic.phy.nbRadioChannels = 40 #according to BLE standard

##########################################################
#                           channel parameters                  #
##########################################################
**.connectionManager.sendDirect = false
**.connectionManager.pMax = 10mW
**.connectionManager.sat = -90dBm
```

```
**.connectionManager.alpha = 4.0
**.connectionManager.carrierFrequency = 2.45e+9Hz
###########################
####Some MAC consts ####
**.nic.mac.llDataHeaderLengthBits = 16 bits
**.nic.mac.llIFSDeviation = 0.0000002 s
**.nic.mac.llIFS = 0.000150 s
**.nic.mac.llmaxAdvPDUduration = 0.000376s #maximum duration of a advertisement
channel packet
**.nic.mac.txPower = 1mW #0dbm - desired TX power
```

*Figure 23: omnet.ini file for parameter settings and configuration options*

## 4.2 Hardware Implementation of BLE network

Implementing the BLE multi-device environment with real-life hardware will help in evaluating the accuracy of our simulation model. Identification of correct hardware to use for the experiment is very important. For the hardware implementation, there are many BLE hardware available in the market. Some of the hardwares are BLE400 and core 51822 wireless module produced by the Nordic semiconductor, BLED 112 BLE dongle by Texas Instruments, and BLEfruit LE friend produced by the Nordic semiconductor. Table 5 describes the available hardware with their properties.

| **Features** | | | |
|---|---|---|---|
| **Bluetooth chip manufacturer** | Nordic semiconductor | Texas Instruments | Nordic Semiconductor |
| **Hardware** | Waveshare Mother board BLE400 Core 51822 wirelss module | BLED 112 BLE dongle  | Adafruit Bluefruit LE friend |

| | | | |
|---|---|---|---|
| **Programmability** | Core 51822 wireless module can be programmed using mother board BLE500 | Compiled hex file can be uploaded inside dongle using DFU (Device firmware upgrade) mode | This cannot be flashed with our application. Can only be used as an extension of a host computer. |
| **Capability** | Core 51822 can act as both master and slave devices (Or) can both send and receive advertisements | BLE dongle can act as both master and slave devices (Or) can both send and receive advertisements | It can only perform as slaves. |

*Table 5: Available hardware in market with their properties*

A comparison is made based on the required properties in the hardware to choose the right one for our experiment. Table 6 shows the comparison results. From the comparison, it is evident that bluegiga BLE dongle has many advantages.

| **Comparison** | **Waveshare** | **Bluegiga** | **Adafruit** |
|---|---|---|---|
| Flexibility to flash the application in the device, to perform as a standalone device | + | + | − |
| The slave devices should be battery operable with a way to measure the power consumption | + | + | − |
| Availability of well documented API to control the BLE operations | + | + | + |
| Existence of relationship between the chip manufacturer and BOSCH | − | + | − |
| Cost of module along with the battery to power them | + | + + | |
| Along with other things the robustness and reliability in all environment | + | + + | + |
| Availability of support in the form of forum or a platform to ask and clarify the questions along the line of implementation | + + | + + | + |

*Table 6: Comparison of the available hardware modules for the experiment*

Bluegiga dongle is based on TI's cc2540 chip. The 128 kB flash block provides in-circuit programmable non-volatile program memory for the device. The master BLE dongle supports up to 8 connections. It also has an integrated Bluetooth Smart stack. The sensitivity of the receiver is -93dbm. On the dongle, applications are developed with BGScript scripting language, thus enabling stand-alone operation. It also has a PCB antenna. It has a SRAM of 8 kB.

The master works by integrating with host computer. BGAPI, which is a simple protocol over UART or USB interfaces, is installed in the host computer. BGLib is a python library for host processor implementing BGAPI is present. With the help of this library, a master python program is written. Figure 24 describes the part of python code, implementing the main functionalities of a master.

```
import library_ble_bled112
import  serial, time, optparse, sys
global ble, ser, deviceMACid,t1, count, latency,fn,fn2

#funtion to read the value from the relevant uuid
def my_ble_evt_attclient_attribute_value(sender, args):
   global latency, count, fn
   if args['atthandle'] == 24:
      ble.send_command(ser, ble.ble_cmd_connection_disconnect(connection_handle))
      t2 = time.time()
      diff = t2-t1
      latency = latency + diff
      count = count + 1
      fi = open("%s.txt" %fn, "a")
      fi.write("%s  " %count)
      fi.write("time taken till disconnection of one slave %s seconds\n" %diff)
      fi.close
#call back function executed after disconnection
def my_ble_evt_connection_disconnected(sender, args):
   global Connection, connection_handle, t1,count, latency, fn,fn2
   if count <26:
      ble.send_command(ser, ble.ble_cmd_gap_discover(1))

#call back function executed after receiving a scan response
def my_ble_evt_gap_scan_response(sender, args):
   global state, PayLoadData, skipCount, Advint,t1
   temp = args['sender']
   if deviceMACid[5] == temp[5]:
      t1 = time.time()
      ble.send_command(ser, ble.ble_cmd_gap_connect_direct(temp, 0, 6, 9, 1000, 0))
def main():
   global ble, ser, state,Parameter,temp,t1
   p = optparse.OptionParser(description='BGLib for ble communication')
   options, arguments = p.parse_args()
   p.set_defaults(port="COM4", baud=115200, packet=False, debug=False)
   ble = library_ble_bled112.BGLib()
   ble.packet_mode = options.packet
   ble.debug = options.debug
  ble.ble_evt_gap_scan_response += my_ble_evt_gap_scan_response
```

```
 ble.ble_evt_connection_status += my_ble_evt_connection_status
ble.ble_evt_connection_disconnected += my_ble_evt_connection_disconnected
ble.on_timeout += my_timeout
#ble.ble_evt_attclient_find_information_found += my_ble_evt_attclient_find_information_found
ble.ble_evt_attclient_procedure_completed += my_ble_evt_attclient_procedure_completed
ble.ble_evt_attclient_attribute_value += my_ble_evt_attclient_attribute_value
ser.flushInput()
ser.flushOutput()
ble.send_command(ser,ble.ble_cmd_flash_ps_erase_all())
time.sleep(0.5)
ble.ble_cmd_system_reset(0)
time.sleep(0.5)

ble.send_command(ser, ble.ble_cmd_gap_set_scan_parameters(0x1F40, 0x1F40, 1))
ble.check_activity(ser, 1)
ble.send_command(ser, ble.ble_cmd_gap_discover(1))

while (1):
   ble.check_activity(ser)
   time.sleep(0.00001)
```

*Figure 24: Part of python code implementing the important functionality of Master*

The slave has to act as a standalone device. Slaves periodically send advertisement and wait for a connection request from the master. BGScript is an XML based scripting language for writing applications. Enables very fast application development and allows programs to be executed directly on the BLED 112 without the need of an external MCU. The code can be developed with any text or source code editor. The code will be compiled with Bluegiga's free compiler. All BLE configuration parameters can be modified using the script file. The project is started by creating a project file. Figure 25 shows the project file of a complete project [10].

```
<?xml version="1.0" encoding="UTF-8" ?>
<project>
   <gatt in="gatt.xml" />
   <hardware in="hardware.xml" />
   <script in="bgdemo.bgs" />
   <usb_main in="cdc.xml" />
         <config in="config.xml" />
   <image out="out-bledcase_1.hex" />
  </project>
```

*Figure 25: Project file of a complete project*

From the Figure 25, the project configuration is described within the <project> tags

- <gatt> tag defines the .XML file containing the GATT data base
- <hardware> tag defines the .XML file containing the hardware configuration
- <script> tag defines the .BGS file containing the BGScript code. If the project does not contain a BGScript code, this tag can be simply left out.
- <usb_main> tag defines the .XML file containing the USB descriptors description. If the project does not use USB interface, this tag can be simply left out.
- <image> tag defines the output .HEX file containing the firmware image
- <device type> tag defines if the project is meant for BLE112 or BLE113 hardware
- <boot fw> tag defines which interface is enabled for DFU firmware upgrades

The .bgs file implements a standalone device without an external host processor. The application is created with BGScript scripting language and the code is shown in the Figure 26. BGScript uses an event based programming approach. The script is executed when an event takes place, and the programmer may register listeners for various events. The BGScript functions and events can be found from the [34] document. Every Bluetooth Smart device needs to implement a GAP service. The GAP service is a part of the "gatt.xml". Bluetooth Smart devices can have services defined by programmer. The 128-bit long UUIDs which are not standardized by the Bluetooth SIG, are used for these services. Figure 27 shows the service in our project.

```
<service uuid="00431c4a-a7a4-428b-a96d-d92d43c8c7cf">
      <description>Bluegiga demo service</description>
      <characteristic uuid="f1b41cde-dbf5-4acf-8679-ecb8b4dca6fe">
         <properties read="true" write="true"/>
         <value type="hex">000</value>
      </characteristic>
   </service>
```

*Figure 26: Service UUID defined for our project*

```
dim tmp(10)
dim ret_result      # USB CDC RX data operation return code
dim ret_data_size   # USB CDC RX data size
dim ret_data(8)     # USB CDC RX data value
event system_boot(major ,minor ,patch ,build ,ll_version ,protocol_version ,hw )
# enable watermark event on UART endpoint (REQUIRED for USB CDC-based DFU trigger)
   call system_endpoint_set_watermarks(3, 1, 0)

# set advertisement interval to 20-30ms, and use all advertisement channels
  call gap_set_adv_parameters(799, 801, 7)
#set to advertising mode
call gap_set_mode(gap_general_discoverable,gap_directed_connectable)
#start timer at 1second interval, handle 0, and repeating
call hardware_set_soft_timer(32768,0,0)
end
```

```
event hardware_soft_timer(handle)
                #start adc read
        call hardware_adc_read(15,3,0)
end
event hardware_adc_result(input,value)
        #adc read complete, write result to attribute
        call attributes_write(xgatt_battery,0,2,value)
end
event connection_disconnected(handle,result)
   call gap_set_mode(gap_non_discoverable,gap_non_connectable)
end
# catch remote write of characteristic value
# NOTE! This function or something similar MUST remain in the BGScript to
# allow access to DFU mode by triggering from a characteristic value update
event attributes_value(connection, reason, handle, offset, value_len, value_data)
   if handle = c_dfu_reboot_trigger then
      # remote client requested DFU reboot
      # NOTE: this will reset on ANY written value. You can check for a specific
      # byte or set of bytes in the "value_data" argument if desired.
      call system_reset(1)
   end if
end
```

*Figure 27: BGScript file for the slave application*

## 4.3 Communication process between master and slave

The communication of any data between master and slave undergoes two phases of operation. Device discovery phase and the connection phase are these two phases in BLE communication. During the device discovery phase, the advertiser sends advertisement packet of the particular type in all three advertising channels. The master can receive the advertisement from the slave when they are synchronized. Synchronization happens when master and slave are on the same channel. On receiving the advertisement packet, the master responds with either scan request or connection request. Once sending the advertisement packet, the slaves wait for any response from the master before going to next advertising event. Figure 28 shows the device discovery process and the connection process between a master and a slave. When master and slave in connection phase, data packets exchange happens through 37 data channel. At the packet reception, if the Access Address is incorrect, the packet shall be rejected. The packet shall also be rejected for an incorrect CRC bits. A packet with an incorrect CRC may still cause a connection event to continue, according to the specification. The master and slave always send a packet if it receives a packet to its peer device, regardless of a valid CRC match. In the scanner state, the scanner shall run a backoff procedure to minimize collisions of Scan_Req pdus from multiple scanners. As soon as entering the scanning state, the upper limit is set to one and the backoffcount shall be set to upper limit. On receiving Adv_Ind PDU or Adv_Scan_Ind PDU, the backoffcount will be decreased by one until it reaches zero. The Scan_Req PDU will be sent only when the backoffcount reaches zero. After sending the Scan_Req PDU the link layer shall listen for Scan_Rep PDU from Advertiser. If the Scan_Rsp PDU was not received, it is considered as failure. On every two consecutive failures, the upper limit of the backoffcount shall be doubled until it reaches the value 256. On every two consecutive successes, the upper limit shall be halved until it reaches the value one. For a successful discovery phase, the scanner should reply to Adv_Ind with a Scan_Req message. In the connection event, if the master doesn't receive a packet from the slave, it should close the connection event. The connection can break down due to other reasons like device moving out of the range, severe interference, or a power failure condition. If the connection supervision timer reaches 6*connInterval before the connection is established, the connection will be lost. The connsupervisiontimeout is a parameter that defines the time between two received data packet PDUs before it is considered as a lost connection. After this the link layer transits from connection state to standby state.

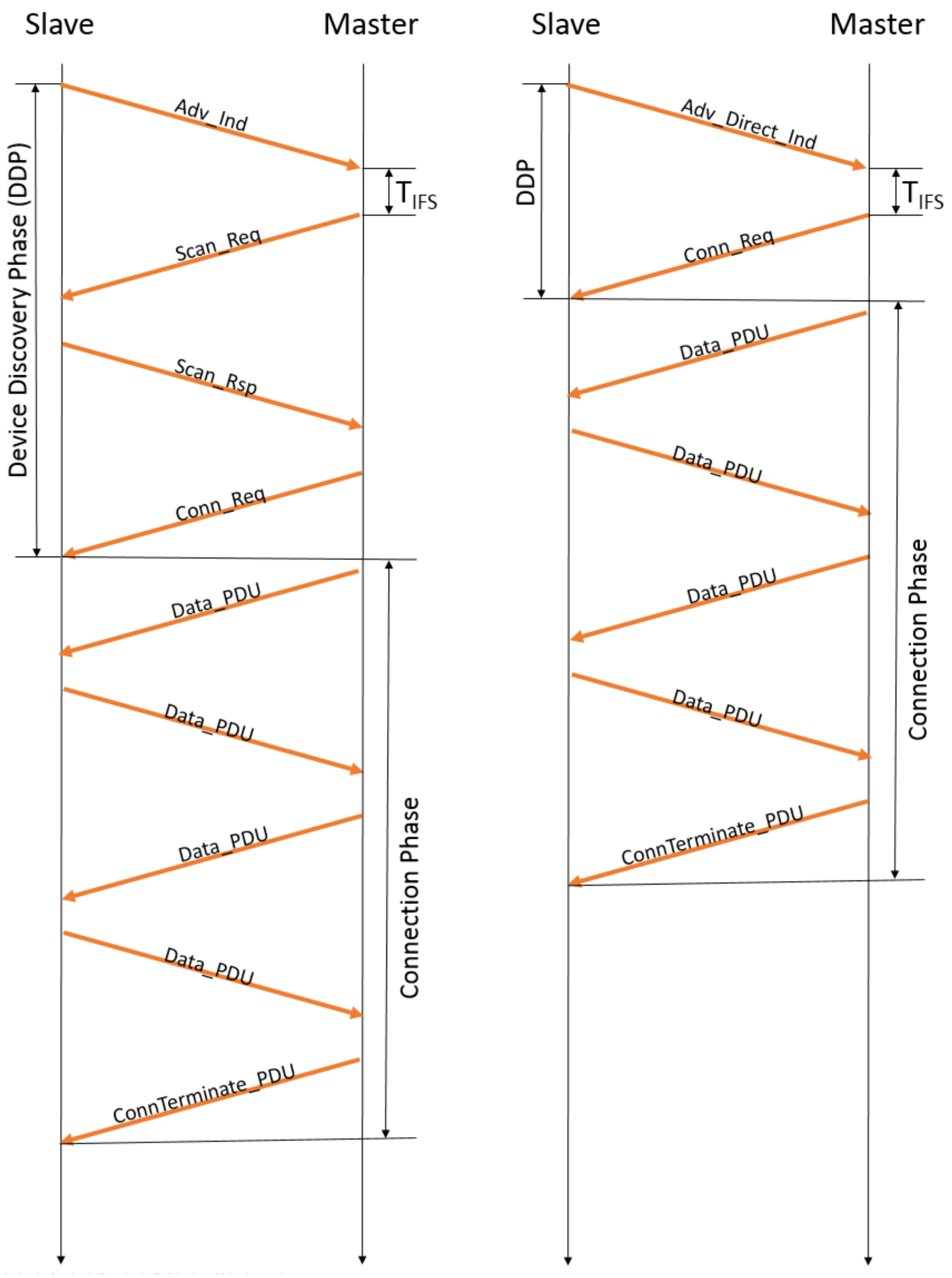


*Figure 28: Communication operation between a master and slave*

# 5 Evaluation

Validation of simulation model can be achieved by implementing the same configuration parameters in both simulation and hardware and by comparing the results. Creating the experimental scenarios to implement in simulation and hardware becomes the first step in the evaluation. The experiment scenarios are required to consider all the factors which will have an impact on the BLE multi-device environment. The configuration parameters to be considered in our experiments are advertising interval, scan interval, scan window, and a number of slaves in the network. The advertising interval is a multiple of 0.625 ms and it can range from 20 ms to 10.24 s. Similarly, scan interval and scan window is a multiple of 0.625 ms and can range from 2.5 ms to 10.24 s. Thus there is exponential number of configurations resulting from the above-mentioned parameters.

Table 7 shows the different experimental scenarios for the validation of our simulation model. For the experiments, Continuous Scanning mode is chosen, which means the scanning interval and scanning window will be of same value. Table 7 shows that some parameters are kept constant while changing other parameters. The experimental scenarios such as (1.1), (2.1), (3.1)…. (6.1) have only 4 slaves. The experimental scenarios such as (1.2), (2.2), (3.2)…. (6.2) have 14 slaves in their BLE network. The experimental scenarios such as (1.3), (2.3), (3.3)…. (6.3) have 30 slaves in their BLE network. At the same time, the experimental scenarios such as (1.1), (1.2), and (1.3) have same advertising interval, scanning interval, and scanning window. Similarly, (X.1), (X.2), and (X.3), where X=2, 3, 4, 5, and 6 have same advertising interval, scanning interval, and scanning window. While, the experimental scenarios (4.1), (4.2), (4.3) and (4.11), (4.21), (4.31) have same advertising interval and number of slaves but different scanning interval and scan window.

| **Experiment scenario** | **Advertising Interval (unit s)** | **Scanning Window (unit s)** | **Scanning Interval (unit s)** | **Number of Slaves** |
|---|---|---|---|---|
| **1.1** | 0.025 | 0.0025 | 0.0025 | 4 |
| **1.2** | 0.025 | 0.0025 | 0.0025 | 14 |
| **1.3** | 0.025 | 0.0025 | 0.0025 | 30 |
| **2.1** | 0.100 | 0.200 | 0.200 | 4 |
| **2.2** | 0.100 | 0.200 | 0.200 | 14 |
| **2.3** | 0.100 | 0.200 | 0.200 | 30 |
| **3.1** | 0.100 | 0.100 | 0.100 | 4 |

| | | | | |
|---|---|---|---|---|
| **3.2** | 0.100 | 0.100 | 0.100 | 14 |
| **3.3** | 0.100 | 0.100 | 0.100 | 30 |
| **4.1** | 0.200 | 0.100 | 0.100 | 4 |
| **4.11** | 0.200 | 0.200 | 0.200 | 4 |
| **4.2** | 0.200 | 0.100 | 0.100 | 14 |
| **4.21** | 0.200 | 0.200 | 0.200 | 14 |
| **4.3** | 0.200 | 0.100 | 0.100 | 30 |
| **4.31** | 0.200 | 0.200 | 0.200 | 30 |
| **5.1** | 2 | 2 | 2 | 4 |
| **5.2** | 2 | 2 | 2 | 14 |
| **5.3** | 2 | 2 | 2 | 30 |
| **6.1** | 5 | 5 | 5 | 4 |
| **6.2** | 5 | 5 | 5 | 14 |
| **6.3** | 5 | 5 | 5 | 30 |

*Table 7: Experimental Scenarios for Simulation and Hardware implementation*

During simulation in omnet.ini file, apart from the above mentioned configuration parameters, connection interval is given a value of 10ms, connection latency is given a value of 0, playground size is given a value of 10m$^2$, type of mobility is given a value of “stationary Mobility”, transmit window offset is given a value of 0, transmit window size is given a value of 1, physical layer sensitivity is given a value of -93dBm, maximum transmission power is given a value of 10.0mW, and number of radio channels is given a value of 40.

These values are chosen according to the core specification defined by Bluetooth SIG [10]. Figure 29 shows the visualization of BLE network with 5 slaves and 1 master in OMNeT simulator. After initializing all the above mentioned parameters, the simulation is made to run. The simulation is run till master finishes the communication for all slaves. Figure 29 is a slice of simulation. In the Figure 29, it is apparent that both master and slaves have a transceiver as a part of them. The position of the slaves and master in the playground are can be defined by topology description language (.ned file). Using the speed tab in the graphics window, the simulation graphics phase can be changed.

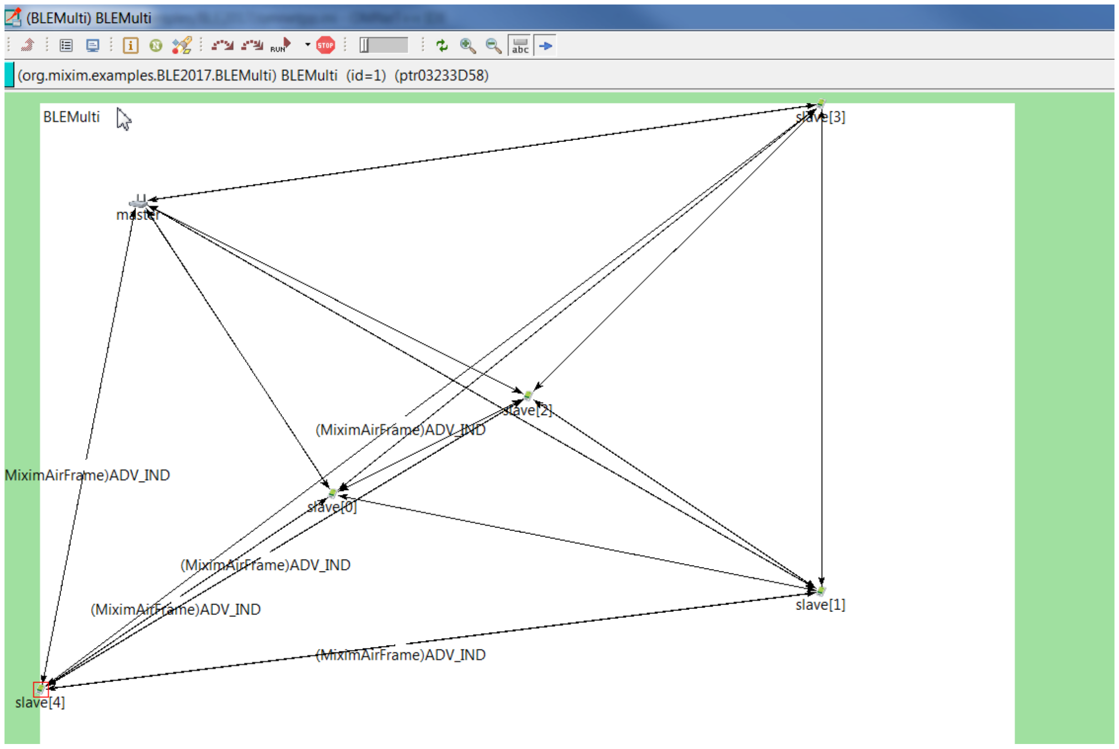


*Figure 29: Graphical visualization of BLE network in simulation*

For hardware implementation, in addition to the configuration parameters defined in the Table 7, connection interval is given a value of 10 ms, and connection latency is given a value of 0. The slaves are programmed with BGScript language and master is programmed with python code. Figure 30 shows the different devices necessary for the hardware experiment. The apparatus required for the experiments are BLED 112 BLE dongles, batteries, and USB hubs. Since stationary mobility is used in the simulation, we make sure the slaves and master are not moved during the experiment.

In both simulation and hardware, the total time taken for the master to finish the communication with all advertisers is calculated. The communication involves the advertiser to send advertisement packet to master, the master then send a connection request or scan request based on the type of the advertisement packet. For the experiments, directed advertisement packet (Adv_Direct_Ind) is used. Advertiser after receiving the connection request packet, goes to connection phase. During the connection phase the master and slave exchange three data packets before terminating the connection. The difference in the time measurement gives the deviation percentage of simulation model and hardware.

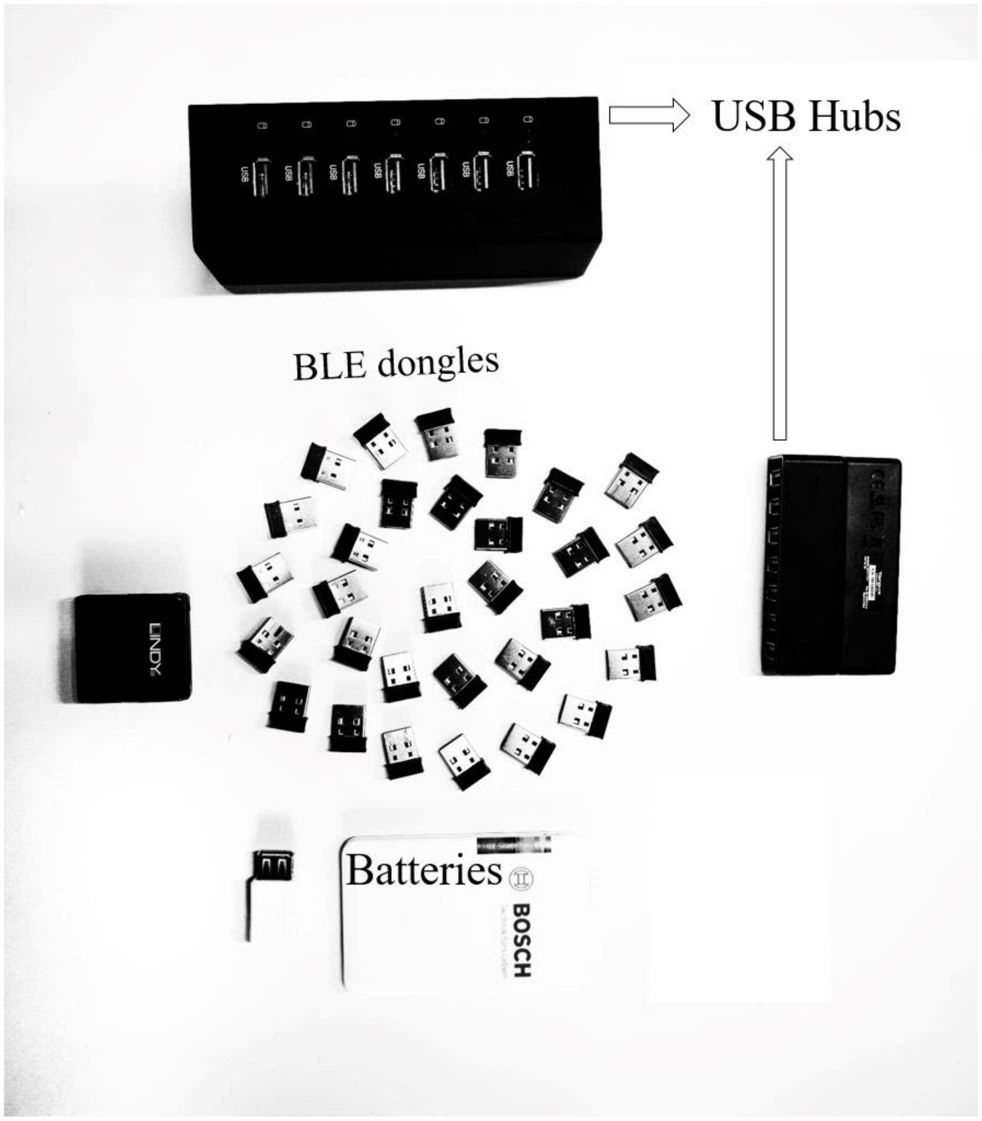


*Figure 30: Apparatus for hardware experiment*

The difference between the software and hardware results are in seconds. The time difference was distinct for diverse configuration parameters. A common unit is needed to measure the deviation for all the possible configuration. For this, we investigate on the main reasons for the difference in hardware result. When the same hardware experiment is repeated, a failure in connection is observed occasionally. Once the master or slave experience a connection failure, it will end up in repeating the device discovery phase and connection phase. Thus the main origin for the difference is that advertiser has to repeat the advertisement event. It is logical to measure the difference in terms of advInterval. The hardware experiment is repeated 30 times. Then the difference between hardware and simulation is calculated. Based on the difference, the experiments are grouped in to green zone, blue zone, and red zone. The green zone is the number of times the hardware experiments give the lowest difference. The red zone is the number of times the hardware experiments give the highest difference. The blue zone covers the hardware experiments with medium difference. Figure 31 shows the deviation of simulation results from hardware results while repeating the same hardware experiment for 30 times. The experimental case for the result in Figure 32 is 4.1 from Table 7. For the case 4.1, the advertising interval equals to 200 ms, the scanning window and the scanning interval are equal to 100 ms with 4 slaves in the network. During this hardware experiments, 23 out of 30 times resulted in difference less than 1*advInterval. It became the green zone. Secondly, 7 out of 30 times resulted in a difference less than 2*advInterval but greater than 1*advInterval and became the blue zone. Finally, no experiment resulted in a difference more than 2*advInterval. Figure 33 shows the deviation result for the case 3.2 from Table 7. For the case 3.2, the advertising interval equals to 100 ms, the scanning window and the scanning interval are equal to 100 ms with 14 slaves in the network. During this hardware experiments, 28 out of 30 times resulted in difference less than 3*advInterval. It became the green zone. Secondly, 7 out of 30 times resulted in a difference less than 6*advInterval but greater than 3*advInterval and became the blue zone. Finally, no experiment resulted in a difference more than 6*advInterval.

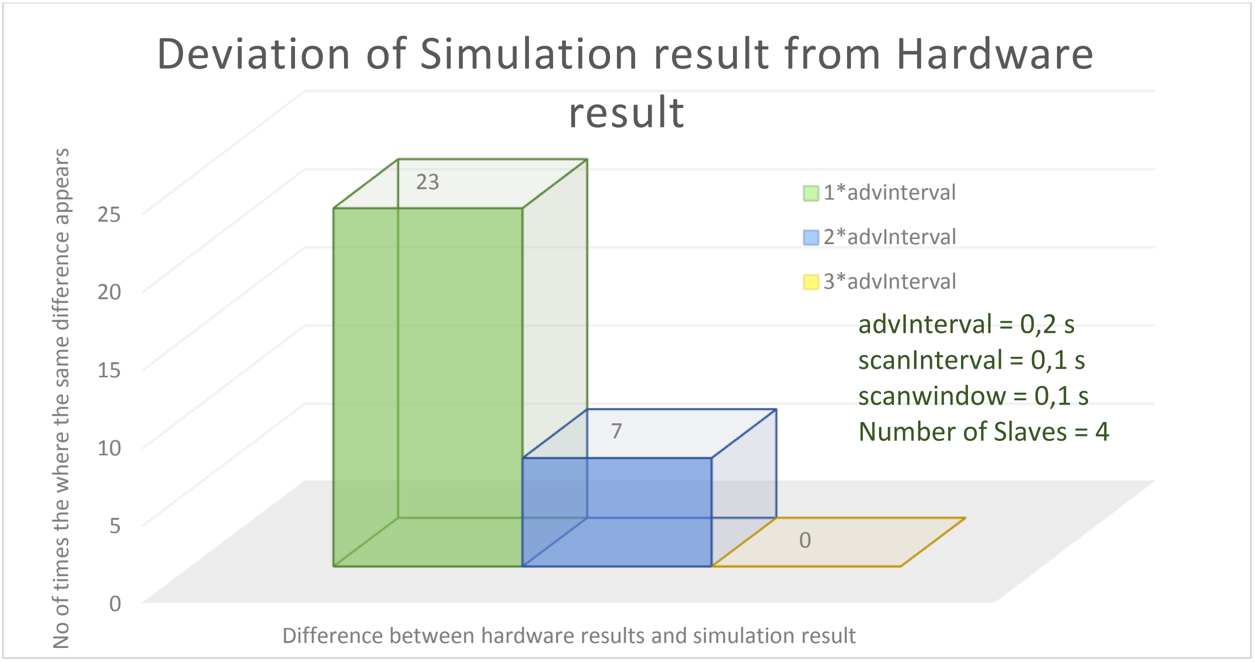


*Figure 31: Deviation of simulation result from hardware result*

Figure 32 shows the distribution of difference values with the reference to simulation value, for the experimental case 4.1 from Table 7. It can be observed from the Figure 32 that there are 13 experiments resulted in a difference value greater than -1*advinterval and less than 0, 10 out 30 experiments resulted in a difference value greater than 0 and less than 1*advinterval, 4 out of 30 experiments resulted in a difference value between 1*advinterval and 2*advinterval and 3 experiments resulted in a difference value between -1*advinterval and -2*advinterval. Figure 32 shows the distribution of difference values with the reference to simulation value, for the experimental case 4.1 from Table 7. It can be observed from the Figure 32 that there are 14 experiments resulted in a difference value greater than -3*advinterval and less than 0, 14 experiments resulted in a difference value greater than 0 and less than 3*advinterval, 2 out of 30 experiments resulted in a difference value between 3*advinterval and 6*advinterval and no experiments resulted in a difference value between -3*advinterval and -6*advinterval.

Thus from the experimental results, larger number of hardware experiments give very low deviation and thus creates a larger green zone Figure 35, Figure 36, Figure 37. This describes the accuracy of the software model of BLE multi-device environment.

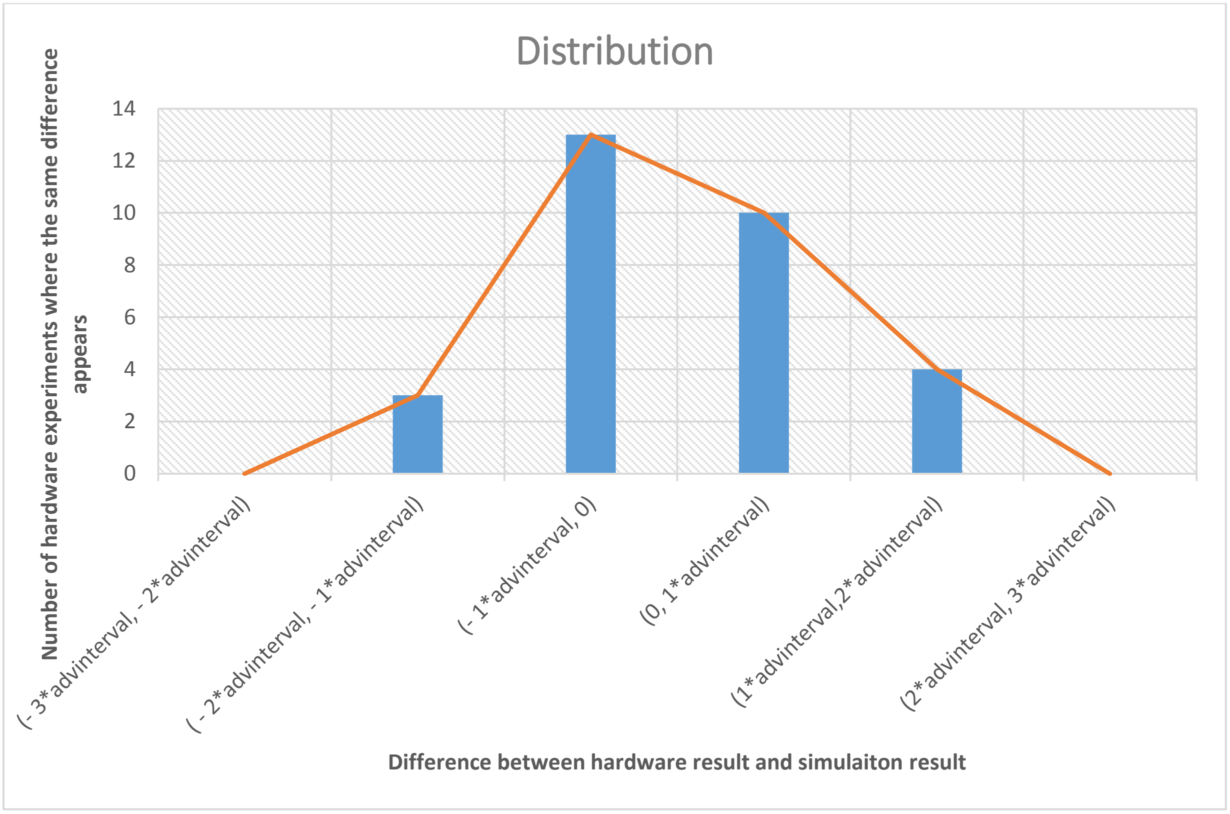


*Figure 32: Distribution of the difference between hardware and simulation result for case 4.1*

Figure 35 depicts an integrated chat for six experimental scenarios with 4 slaves in their BLE network. Figure 36 depicts an integrated chat for six experimental scenarios with 14 slaves in their BLE network. Similarly figure 37 depicts an integrated chat for experimental scenarios with 30 slaves in their BLE network. It is observed that when the number of slaves are very low i.e. 4, the difference between the hardware and simulation results does not exceed 3*advInterval. When the number of slaves grows above 8, the deviation could reach a maximum of 9*advInterval. It is evident that the difference does not increase above 9*advInterval even for 30 number of slaves.

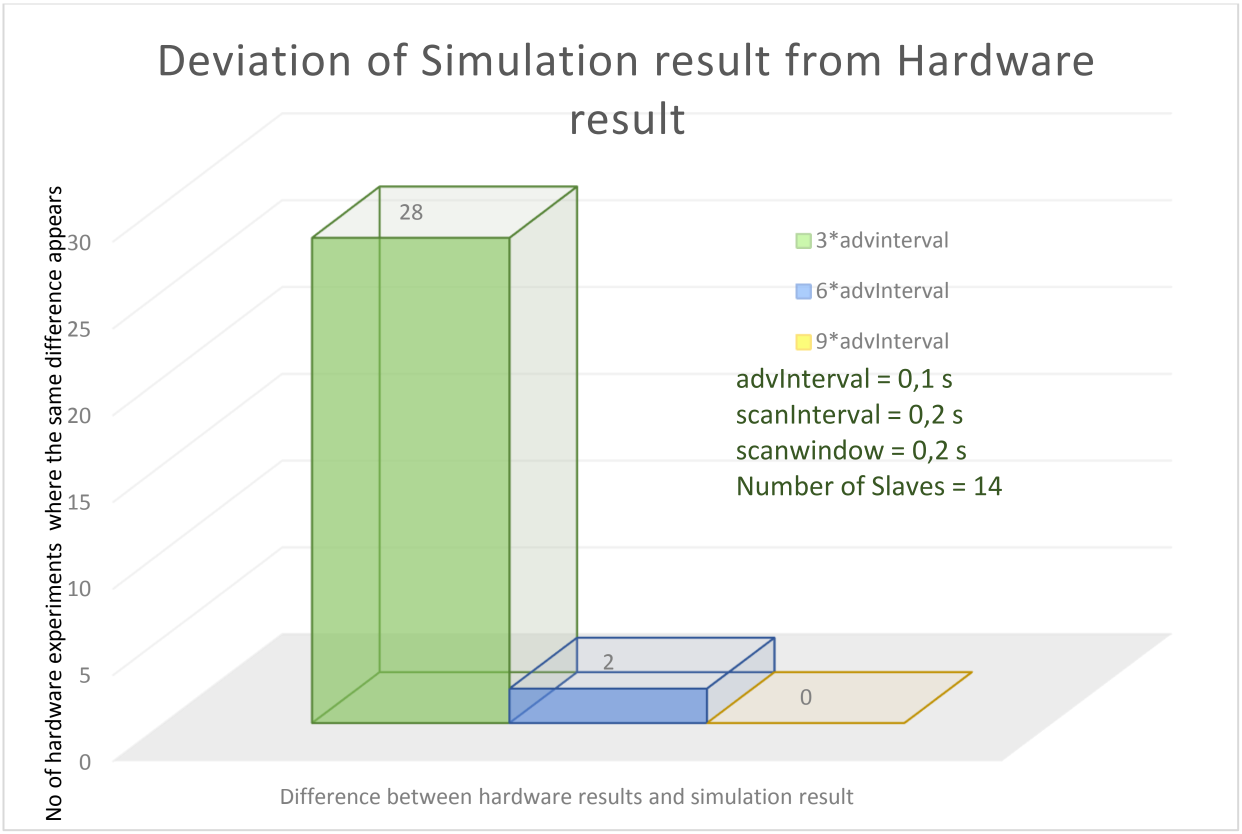


*Figure 33: Deviation for the experimental case 3.2*

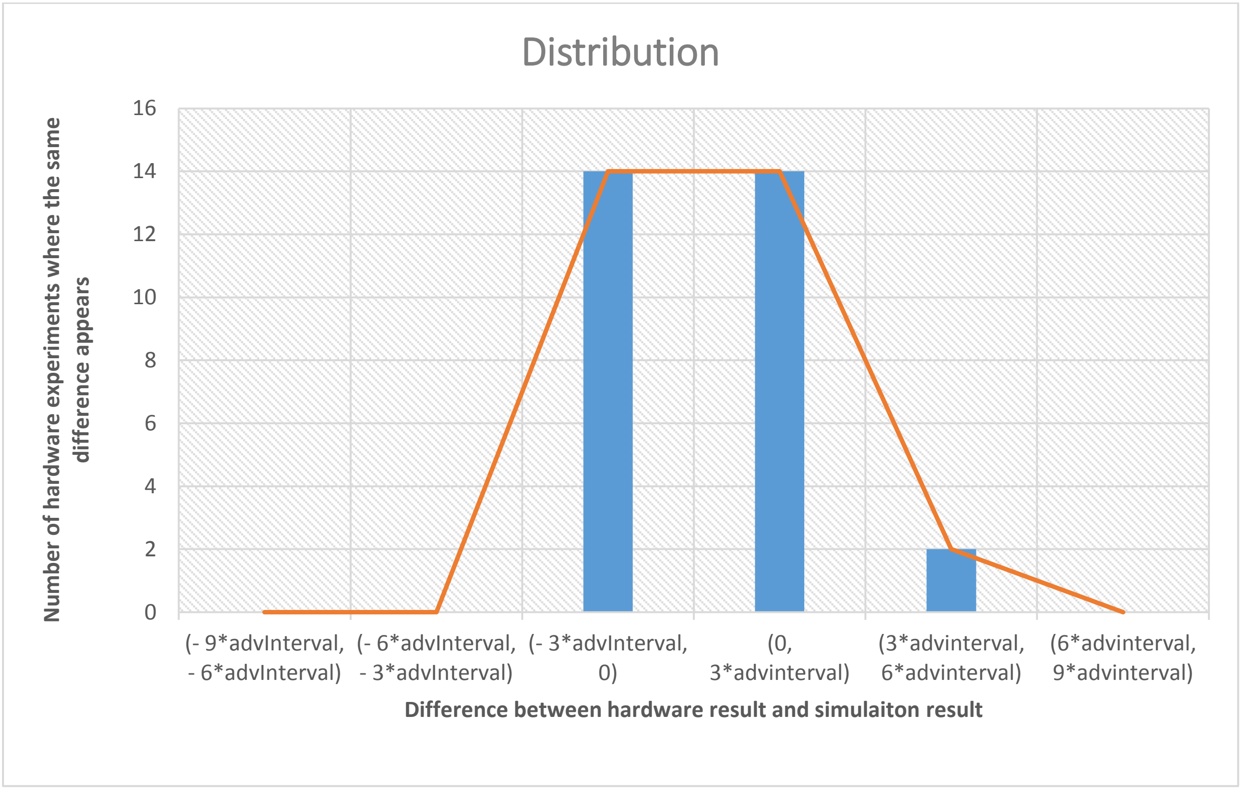


*Figure 34: Distribution of the difference between hardware and simulation result for case 3.2*

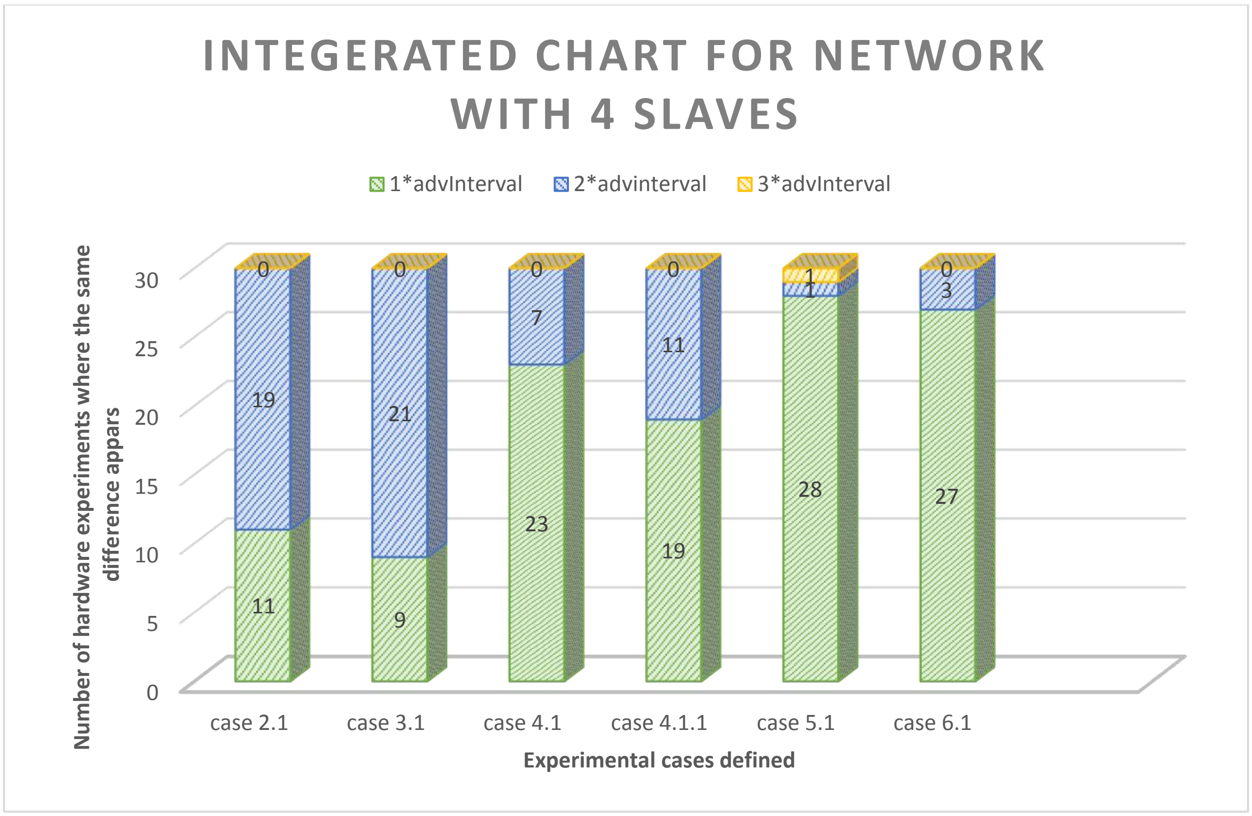


*Figure 35: Integrated chat for BLE network with 4 slaves*

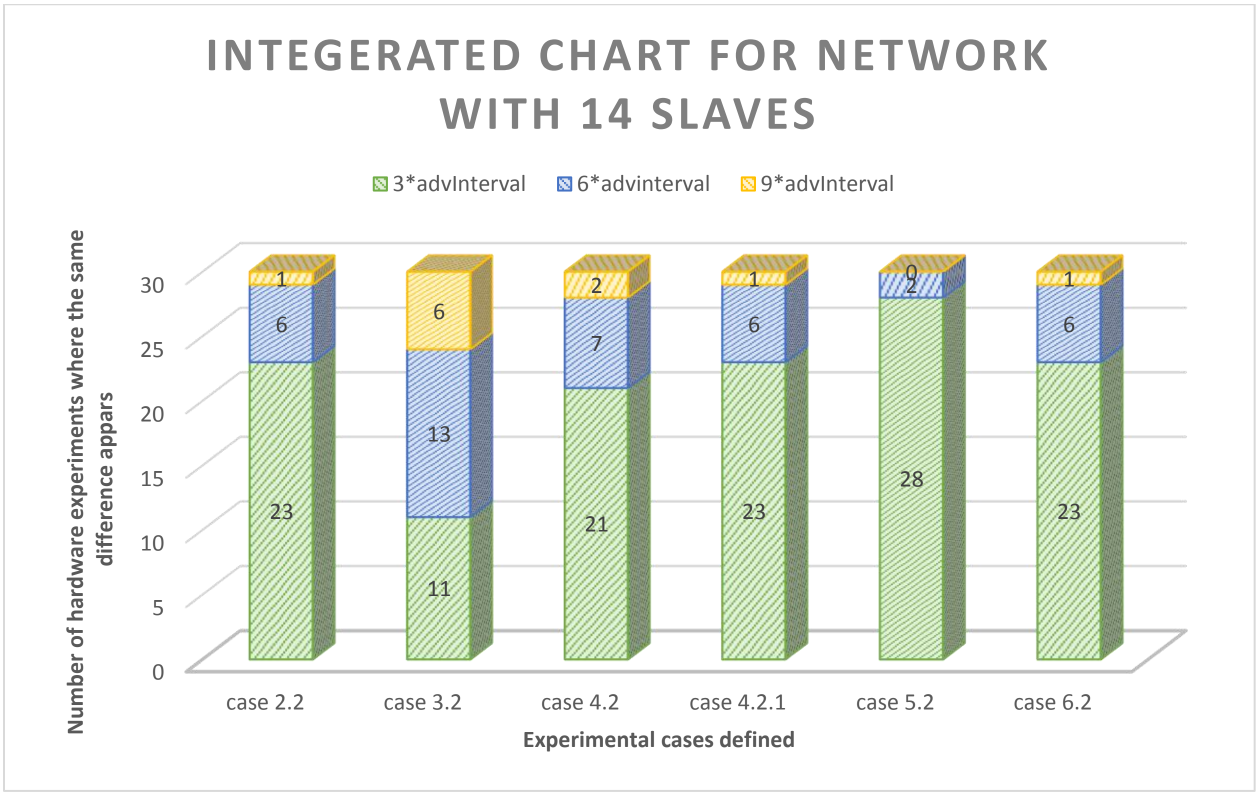


*Figure 36: Integrated chat for BLE network with 14 slaves*

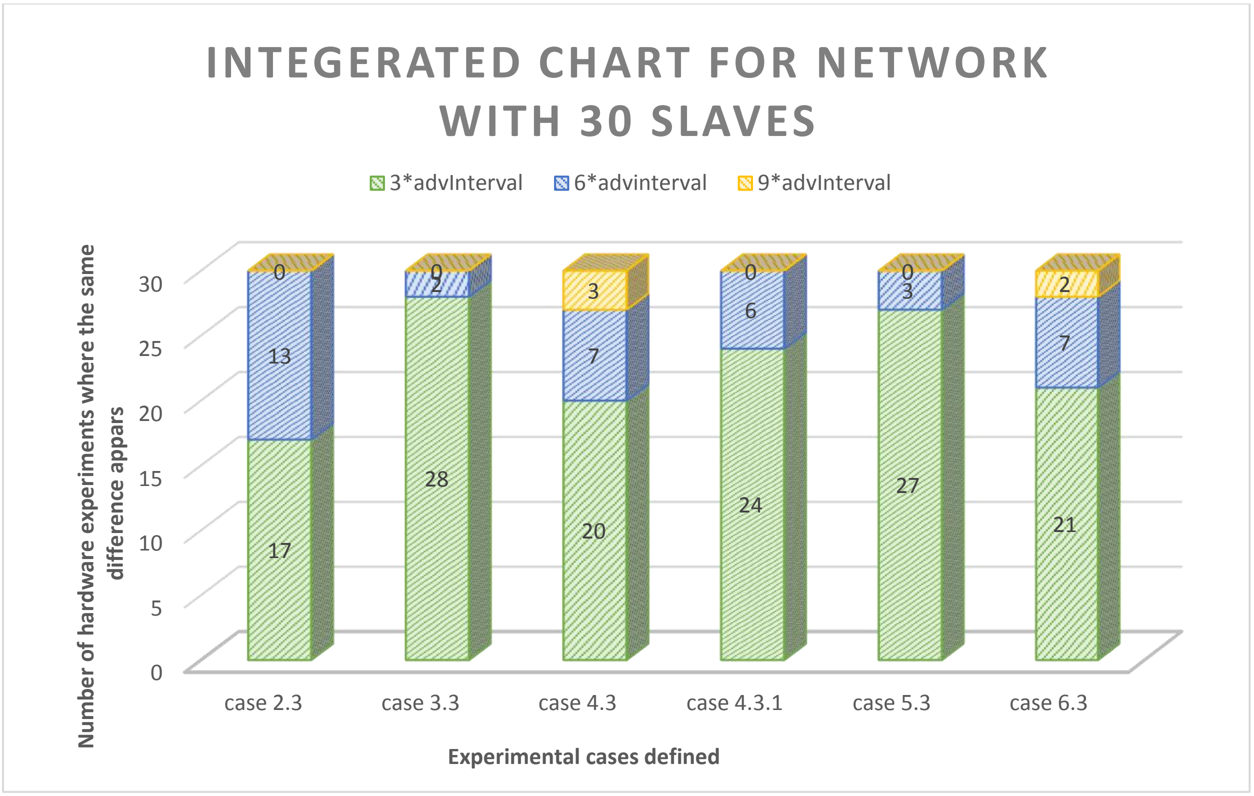


*Figure 37: Integrated chat for BLE network with 30 slaves*

The different experimental scenarios become the X axis for integrated chats. It can be observed from the results that for all the experimental scenarios the green zone maintains the majority. The simple path-loss model used in the physical layer simulation model, does not consider the presence of objects in the real world environment. It is also assumed that the simulation model does not consider the mobility of any BLE node.

# 6 Conclusion

In this thesis, the discrete event simulation tools that are available are studied based on its usability for the BLE simulation model designing. The configuration parameters which have an impact on the BLE network performance is identified. The simulation model of the BLE multi-device environment is created by having BLE nodes as building blocks. Abstraction of a BLE node is designed by modelling the lower layers of BLE communication protocol. The simulation model is designed in OMNeT++ with the help of MiXiM framework. The simulated model for the physical layer, MAC layer, and network layer for BLE node is designed. For the physical layer, a simple path loss model form MiXiM library is used. The MAC layer represents the link layer properties of BLE. The MAC layer in the simulation model performs the functionality depending on the state (standby, advertising, initiating, and connection) of the BLE node. The network layer performs the functionality of Host Control Interface between lower host layers and the upper layer. The validation of the simulation model requires the implementation the same BLE network system in hardware. The hardware implementation is done with the help of bluegiga BLE dongles. The results from the simulation model and hardware implementation are compared. The deviation between the hardware implementation and simulation model is presented. From the deviation chart, it can be inferred that when repeating the hardware system, the probability of getting a result close to simulation is very high.

Deviation in the results are because of the failure in the connection between master and slave. Reasons for the deviation between the simulation model and hardware implementation is identified. When there are lot of BLE devices operating independently within a multi-device environment there is a strong likelihood of two independent BLE devices having their transceiver tuned to the same RF channel resulting in a channel collision. Metal objects and electrical equipment emitting strong RFs can also interfere with Bluetooth or block it entirely. Apart from metal, Things like plaster, concrete, bullet proof glass, and etc. will still interfere with Bluetooth signal. If Bluetooth device is in contact with another wireless devices which uses same band, can be blocked.

The complexity of the multi-device environment is increased with presence of different real objects made of various materials and the presence of different wireless protocol like Wi-Fi. In future, the accuracy of hardware abstraction can be achieved by modelling all the physical objects in the real world.   The mobility of the BLE node

# 7 Bibliography

[1] S. Mehta, N. Ullah, M. H. Kabir, M. N. Sultana, and K. S. Kwak, “A case study of networks simulation tools for wireless networks,” in *Modelling & Simulation, 2009. AMS’09. Third Asia International Conference on*, 2009, pp. 661–666.

[2] A. Varga, “OMNeT++,” *Modeling and Tools for Network Simulation*, pp. 35–59, 2010.

[3] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, “Network simulations with the ns-3 simulator,” *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

[4] R. Barr, Z. J. Haas, and R. van Renesse, “Jist/swans,” *Wireless Networks Laboratory, Cornell University. http://jist. ece. cornell. edu*, 2005.

[5] A. Sobeih *et al.*, “J-Sim: A simulation and emulation environment for wireless sensor networks,” *IEEE Wireless Communications*, vol. 13, no. 4, pp. 104–119, 2006.

[6] J. Prokkola, “Opnet-network simulator,” *URL http://www. telecomlab. oulu. fi/kurssit/521365A tietoliikennetekniikan simuloinnit ja tyokalut/Opnet esittely*, vol. 7, 2006.

[7] A. Köpke *et al.*, “Simulating wireless and mobile networks in OMNeT++ the MiXiM vision,” in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2008, p. 71.

[8] K. Pahlavan and P. Krishnamurthy, *Principles of wireless networks: A unified approach*: Prentice Hall PTR, 2011.

[9] L. Ahlin, J. Zander, and S. Ben Slimane, *Principles of wireless communications*: Studentlitteratur, 2006.

[10] S. I. Bluetooth, “Bluetooth core specification version 4.0,” *Specification of the Bluetooth System*, 2010.

[11] C. Gomez, J. Oller, and J. Paradells, “Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology,” *Sensors*, vol. 12, no. 9, pp. 11734–11753, 2012.

[12] S. Cocorada, “An IEEE 802.11 g simulation model with extended debug capabilities,” in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2008, p. 81.

[13] M. Bredel and M. Bergner, “On The Accuracy of IEEE 802.11g Wireless LAN Simulations Using OMNeT++,” in *Proceedings of the Second International ICST Conference on Simulation Tools and Techniques*, Rome, Italy, 2009.

[14] S. Han and N. B. Abu-Ghazaleh, “A realistic model of co-located interference for wireless network packet simulation,” in *Mobile Adhoc and Sensor Systems (MASS), 2010 IEEE 7th International Conference on*, 2010, pp. 472–481.

[15] C.-J. Hsu and Y.-J. Joung, “An ns-based Bluetooth topology construction simulation environment,” in *Proceedings of the 36th annual symposium on Simulation*, 2003, p. 145.

[16] C. Scavongelli and M. Conti, “A systemC Bluetooth network simulator,” in *Ph. D. Research in Microelectronics and Electronics (PRIME), 2014 10th Conference on*, 2014, pp. 1–4.

[17] S. Gajbhiye, M. Sharma, S. Karmkar, and S. Sharma, “Design, implementation and security analysis of Bluetooth pairing protocol in NS2,” in *Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on*, 2016, pp. 1711–1717.

[18] K. Mikhaylov, “Simulation of network-level performance for Bluetooth Low Energy,” in *Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on*, 2014, pp. 1259–1263.

[19] A. Castiglione, F. Palmieri, U. Fiore, A. Castiglione, and A. de Santis, "Modeling energy-efficient secure communications in multi-mode wireless mobile devices," *Journal of Computer and System Sciences*, vol. 81, no. 8, pp. 1464–1478, 2015.
[20] R. Woodings and M. Pandey, "WirelessUSB: A low power, low latency and interference immune wireless standard," in *Wireless Communications and Networking Conference, 2006. WCNC 2006. IEEE*, 2006, pp. 1367–1373.
[21] E. Ferro and F. Potorti, "Bluetooth and Wi-Fi wireless protocols: A survey and a comparison," *IEEE Wireless Communications*, vol. 12, no. 1, pp. 12–26, 2005.
[22] G. D. Putra, A. R. Pratama, A. Lazovik, and M. Aiello, "Comparison of energy consumption in Wi-Fi and bluetooth communication in a Smart Building," in *Computing and Communication Workshop and Conference (CCWC), 2017 IEEE 7th Annual*, 2017, pp. 1–6.
[23] G. Gurczik and M. Behrisch, "Modelling and simulating Bluetooth-based moving observers," in *Models and Technologies for Intelligent Transportation Systems (MT-ITS), 2015 International Conference on*, 2015, pp. 223–228.
[24] J. Mikulka and S. Hanus, "Bluetooth EDR physical layer modeling," in *Radioelektronika, 2008 18th International Conference*, 2008, pp. 1–4.
[25] A. Karnik and A. Kumar, "Performance analysis of the Bluetooth physical layer," in *Personal Wireless Communications, 2000 IEEE International Conference on*, 2000, pp. 70–74.
[26] M. J. Morón, R. Luque, and E. Casilari, "Modeling of the transmission delay in Bluetooth piconets under serial port profile," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 4, pp. 2080–2085, 2010.
[27] N. Semiconductor, *Bluetooth low energy wireless technology backgrounder*: Version.
[28] K. Mikhaylov, N. Plevritakis, and J. Tervonen, "Performance analysis and comparison of Bluetooth Low Energy with IEEE 802.15. 4 and SimpliciTI," *Journal of Sensor and Actuator Networks*, vol. 2, no. 3, pp. 589–613, 2013.
[29] K. Mikhaylov and T. Hänninen, "Mechanisms for improving throughput and energy efficiency of Bluetooth Low Energy for multi node environment," *Journal of High Speed Networks*, vol. 21, no. 3, pp. 165–180, 2015.
[30] P. Kindt, D. Yunge, R. Diemer, and S. Chakraborty, "Precise energy modeling for the bluetooth low energy protocol," *arXiv preprint arXiv:1403.2919*, 2014.
[31] C. Gomez, I. Demirkol, and J. Paradells, "Modeling the maximum throughput of Bluetooth low energy in an error-prone link," *IEEE Communications Letters*, vol. 15, no. 11, pp. 1187–1189, 2011.
[32] K. Cho *et al.*, "Analysis of latency performance of Bluetooth low energy (BLE) networks," *Sensors*, vol. 15, no. 1, pp. 59–78, 2014.
[33] A. Varga and others, "Omnet++ user manual," *OMNeT++ Discrete Event Simulation System. Available at: http://www. omnetpp. org/doc/manual/usman. html*, 2010.
[34] "AN980 BLUETOOTH SMART SDK,"