HOCHSCHULE DER MEDIEN

Stuttgart Media University, Germany

Print and Media Faculty

Computer Science and Media

MASTER-THESIS

FruityMesh Implementation of a battery-efficient mesh network on top of Bluetooth Low Energy

Marius Heil

E-Mail: info@mariusheil.de

Stuttgart Media University, Germany

Submission date:	1st June 2015
Supervisor:	Prof. Dr. Ansgar Gerlicher
Co-supervisor:	M. Sc. Christian Feser

m-way solutions

Eidesstattliche Versicherung

Name: Heil **Vorname:** Marius

Matrikel-Nr.: 27509 **Studiengang:** Computer Science and Media

Hiermit versichere ich, Marius Heil, an Eides statt, dass ich die vorliegende Masterarbeit mit dem Titel „FruityMesh – Implementation of a battery-efficient mesh network on top of Bluetooth Low Energy“ selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der eidesstattlichen Versicherung und prüfungsrechtlichen Folgen (§ 26 Abs. 2 Bachelor-SPO bzw. § 19 Abs. 2 Master-SPO der Hochschule der Medien Stuttgart) sowie die strafrechtlichen Folgen (siehe unten) einer unrichtigen oder unvollständigen eidesstattlichen Versicherung zur Kenntnis genommen.

Auszug aus dem Strafgesetzbuch (StGB)

§ 156 StGB Falsche Versicherung an Eides Statt

Wer von einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

Ort, Datum Unterschrift

Abstract

This thesis introduces a mesh implementation for purely battery-powered nodes that works on top of the Bluetooth 4.1 specification and proposes the FruityMesh algorithm to solve the complicated task of autonomous meshing and maintenance. Current state-of-the-art scientific, BLE and other mesh networks are presented before discussing possibilities and limitations of Bluetooth Low Energy. A simulation is used to evaluate the algorithm's behaviour for a large amount of nodes. The final prototype uses the Nordic nRF51 platform in combination with the S130 software-based BLE stack that can concurrently manage a Central and Peripheral role.

Other topics in this thesis include an overview of the currently available implementations for smartphones, available chipsets, and corresponding development environments.

Both the simulation and prototypical implementation have shown that it is currently possible to build a mesh network using standard BLE connections that can work with limited power supplies and a high number of nodes without the need for a central coordinator.

Kurzfassung

Diese Arbeit stellt die Implementierung eines Mesh Netzwerkes vor, welches auf dem Bluetooth 4.1 Low Energy Standard aufbaut. Der vorgestellte FruityMesh Algorithmus übernimmt hierbei die Aufgabe der Vernetzung und Selbstheilung der Netztopologie. Als Untersuchungsgrundlage werden aktuelle Mesh Netzwerke aus dem Bereich der Wissenschaft und Industrie vorgestellt und auf ihre Schwächen und Vorteile hin untersucht. Im Vordergrund steht hierbei die Suche nach Techniken, die sich auf ein Bluetooth Low Energy Mesh Netzwerk übertragen lassen. Mithilfe einer selbstgeschriebenen Simulation wird der Algorithmus mit einer großen Anzahl an Netzknoten getestet und verbessert. Eine prototypische Implementierung auf Basis der nRF51 Plattform von Nordic Semiconductors, in Verbindung mit deren hauseigenem Bluetooth Stack – dem S130 SoftDevice – zeigt weitere Probleme und Lösungen auf.

Des Weiteren beinhaltet die Arbeit einen Überblick über die aktuell verfügbaren Implementierungen für Smartphones, sowie Chipsätze und deren Entwicklungsumgebungen.

Sowohl die Simulation als auch die prototypische Implementierung haben gezeigt, dass es mit aktuellen Mitteln möglich ist ein Mesh Netzwerk zu entwerfen, welches standardisierte BLE Verbindungen verwendet und mithilfe von diesen eine lange Batterielaufzeit, bei einer großen Anzahl an Netzknoten, erlaubt ohne dabei einen zentralen Koordinator zu benötigen.

Table of Contents

1 Introduction

1.1 Preface

The Internet of Things is already a $2 billion revenue business for Intel (Intel 1/15/2015) and as revenue and shipment of chipsets and devices grow, the world is on the brink to omnipresent interconnectivity. It is envisioned that every device ranging from the size of a fingernail to desktop computers will finally take part in an ever-growing network of electronics.

Standards are readily available for either the plain old wired connections or the wireless ones. Many modern consumer electronic devices communicate over radio transmitters with standards such as NFC, Bluetooth, LTE or Wi-Fi. Some smaller devices such as sensor tags, though, remain a special use-case because they are usually running on limited battery power. This is where protocols like ZigBee, ANT or Bluetooth Smart (alias Bluetooth Low Energy) come into play. These protocols can work with very little amounts of energy but do impose some restrictions such as their limited send-range and throughput.

1.2 Motivation

Connecting these small battery-powered devices is possible using a multitude of network topologies that are given by the protocol's capabilities. For this reason, Bluetooth LE is a strong competitor when it comes to smartphones in conjunction with sensor tags (Bluetooth SIG 2015b), while ZigBee is often used in home automation because of its meshing capabilities (ZigBee Alliance 2015). However, studies have shown that ZigBee uses a higher amount of energy than Bluetooth Smart (Dementyev et al. 2013). Additionally, ZigBee is not available in smartphones yet and requires additional hardware and setup effort. There are many other protocols in this field, but (Ammari 2013, p. 12) points out that "until today, most WSN deployments have a strong scientific background". ZigBee is currently one of the few mesh solutions that try to make their way into the hand of consumers.

The development of a meshing protocol on top of Bluetooth Low Energy would profit from the easy set-up procedure, encryption, the excellent low energy capabilities and the wide adoption in smartphones. Currently, there is only one major company that tried to implement a mesh network with Bluetooth Smart which they named CSRmesh (CSR 2/25/2014). There are two major shortcomings: Their protocol will only be available within 12-18 months (cf. Appendix 2) and will probably not work with devices that have small batteries, because the nodes must listen to the network's broadcasts at most times (AIRcable.net 2014).

The lack of a publicly available implementation of a mesh network on top of Bluetooth Smart leaves the unique chance to develop a solution and fill this gap. During the writing of this thesis, a press release was published, detailing that the Bluetooth SIG is interested in developing a mesh technology that builds on the current standard as well (Bluetooth SIG 2/24/2015).

1.3 Goals

After extensive research, it became clear that a mesh network on top of Bluetooth Low Energy could not fulfil all demands at the same time. It would, for example, not be possible to develop a network with little battery drain while having a very low hop latency at the same time. Without special hardware, a close to zero latency implies that the receiving devices must listen at all times. This is not possible when operating on little energy reserves. The desired network characteristics have therefore been chosen based on some use-cases that can be found in chapter 6.

In order to reach consumers as early as possible, it was decided to stay compatible with the most current Bluetooth specification, which as of writing is Bluetooth 4.2. This allows current devices to communicate with the mesh and will enable easy adoption of any changes that are made to the standard in the future.

This thesis presents the research that was needed for finding a good mesh solution which builds upon the current standard. It will also point out improvements that could be made to the standard without breaking backward compatibility. As a result, a simulation and a working prototype will be presented.

1.4 Structure

The first chapters introduce the reader to the Bluetooth Low Energy standard and presents a survey of the currently available devices and development environments. Next, an evaluation of existing mesh network technology is given and significant features are pointed out. This is followed by a chapter that is dedicated to the research of the final algorithm, which is then further explained and refined with a simulation and a prototypical implementation during the following chapters. Finally, a discussion of the expected performance and two showcases will conclude this work.

1.5 Terminology

This thesis assumes that the reader has a well understanding of computer science and therefore makes use of many common expressions and abbreviations without preceding explanation. The reader is advised to reference the glossary in chapter A if any of the used expressions are unknown.

1.6 Electronic Sources and Statements

Because this work has been written at a time when extensive research was still ongoing in many of the discussed topics, there were many non-scientific sources that had to be referenced because they were the only source of information. All of the internet resources have therefore been marked with their access times.

2 Bluetooth Low Energy Basics

Before we dive into the specifics of building a mesh network, the reader should be familiar with the fundamentals of the Bluetooth Low Energy protocol that are key to understanding the following chapters and the prototype implementation. This chapter cannot be considered a Bluetooth reference. It makes an attempt to give a short overview that is necessary to understand the research and its results. The following information has been compiled from the latest official Bluetooth Low Energy Specification (Bluetooth 4.2) and the book “Getting Started With Bluetooth Low Energy” (Townsend 2014).

2.1 General

Bluetooth is a wireless standard for exchanging data over short distances which has been developed by Ericsson in 1994. The specification has seen many updates over its lifetime and has gone through four major revisions. The latest being 4.2 which is the one that will be presented in this chapter. Bluetooth 4.0 introduced the new Bluetooth Low Energy (BLE, Bluetooth LE) protocol that is widely different from Classic Bluetooth.

Development of this new protocol was done by Nokia and began as early as 2001. It was then decided to add it to the existing Bluetooth standard in order to profit from the wide market adoption of the Bluetooth brand. This did not happen until 2010, which is when the final 4.0 specification was released. In the following years, Bluetooth LE slowly gained traction with its marketing name Bluetooth Smart and its integration into a wide range of smartphones and wearables.

Bluetooth Low Energy enables many use-cases that cannot be met with existing technologies like Wi-Fi, Classic Bluetooth or any cellular technology because it consumes a lot less power. Sending a small amount of data every second allows a device to work on a single coin cell battery for more than a year (cf. Appendix 1) depending on the chipset. And by using energy harvesting or photovoltaic cells it is possible to build devices with infinite time of operation. Another selling point of Bluetooth Low Energy are the cheap material costs for the hardware and the possibility of integrating both Bluetooth Classic and Bluetooth Low Energy on a single chip while sharing some of the circuitry and the antenna.

2.2 Protocol Layers

The BLE stack is a combination of several layers. The next sections provide an overview of the layers that play an important role when developing a mesh network. The best effort was made to separate the information according to the Bluetooth specification but it was decided to include certain information in different sections for better understanding.

The stack can be divided into three parts. The first being the **Controller** that is most of the time implemented in hardware. After that, we have the **Host** that is often implemented in software to account for changes in upcoming Bluetooth standards. The upmost layer is the **Application** layer that is programmed separately for each specific use-case.

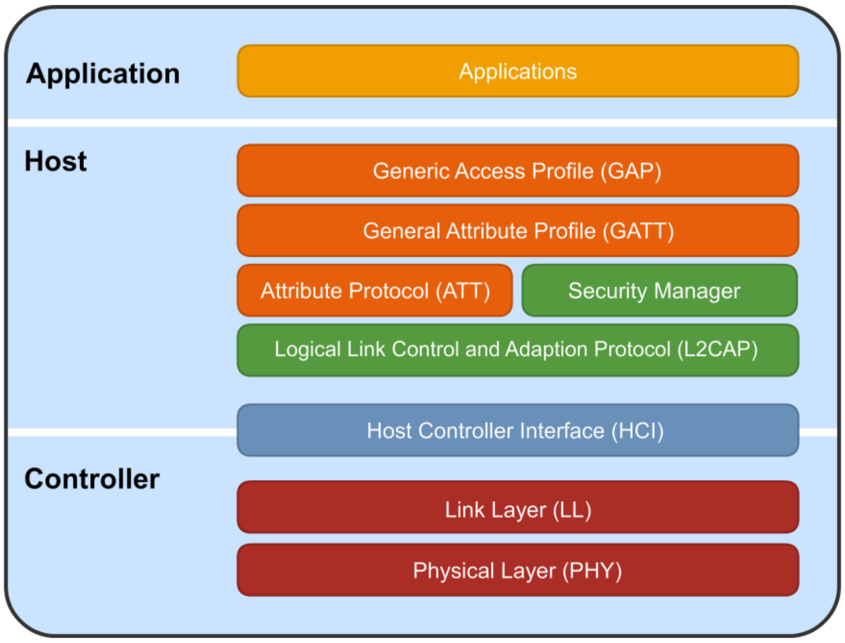

Application
Applications
Host
Generic Access Profile (GAP)
General Attribute Profile (GATT)
Attribute Protocol (ATT)
Security Manager
Logical Link Control and Adaption Protocol (L2CAP)
Host Controller Interface (HCI)
Controller
Link Layer (LL)
Physical Layer (PHY)

Figure 2.1: Layers of the Bluetooth Low Energy stack
Original source: https://developer.bluetooth.org/TechnologyOverview/Pages/BLE.aspx

2.3 Physical Layer (PHY)

Bluetooth Classic and its new sibling Bluetooth Low Energy both operate in the 2.4 GHz Industrial, Scientific and Medical Radio Band (ISM Band). The Bluetooth Low Energy specification divides the available bandwidth into 40 evenly distributed channels with a spacing of 2 MHz.

The transmit power can be varied from -20 dBm to a maximum of 10 dBm, with most devices being hardware limited to 0 dBm. This results in a usual range of about 100 m at 0 dBm and an extended range of up to 450 m at 8 dBm according to (Bluegiga 2015). These ranges can only be reached when a line of sight is available and will decrease immensely in buildings, with typical ranges between 5-30 m.

When data is transmitted over the air it is prefixed with a preamble – which is an alternating sequence of digital ones and zeros - and suffixed with a CRC checksum. The receiver can then use the preamble for automatic gain control and for synchronizing its timings on the signal. The CRC is used afterwards to check the message against corruption. Next, a process called whitening is used to alter long sequences of zeros or ones in the transmitted signal. This ensures that transmitter and receiver are able to stay in sync during the transmission of a packet. The signal is sent over the air at a symbol rate of 1 MHz with a Gaussian Frequency Shift Keying (GFSK) modulation. This modulation smoothes out the transitions between digital ones and zeros in the frequency spectrum and therefore reduces interference on neighbouring channels. By relaxing the parameters for the GFSK modulation from Classic Bluetooth, it was possible to reduce energy consumption during the transmission. These relaxed parameters allow the use of low-cost hardware such as cheap oscillators with a high drift. This drift is usually measured in parts per million (ppm) and a drift of ±50 ppm means that the oscillator will at most drift 50 µs per second in either direction.

Keeping two devices in sync during a transmission is essential to maintain a stable link and requires a much higher accuracy than the one that is needed during the long sleep intervals between transmissions. Most BLE SoCs contain an active clock with a high frequency and a sleep clock with a typical 32 KHz. The active clock is used to generate the transmission timings at 1 MHz which are required to be within ±50 ppm, and can additionally be used to clock a CPU. Between transmissions, the device is allowed to use a sleep clock accuracy with a much higher maximum drift of ±500 ppm. The exact timing parameters can be found in (Bluetooth 4.2, Vol. 6, Part A, 3.3) and (Bluetooth 4.2, Vol. 6, Part B, 4.2).

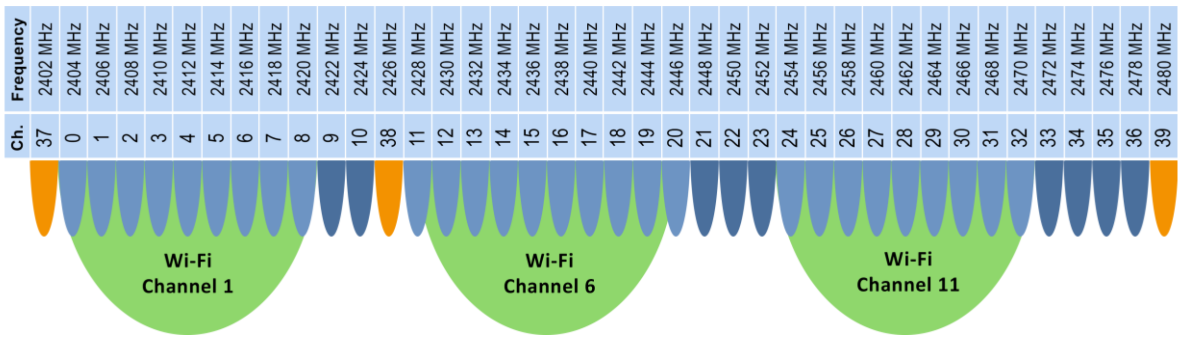

Frequency
2402 MHz 2404 MHz 2406 MHz 2408 MHz 2410 MHz 2412 MHz 2414 MHz 2416 MHz 2418 MHz 2420 MHz 2422 MHz 2424 MHz 2426 MHz 2428 MHz 2430 MHz 2432 MHz 2434 MHz 2436 MHz 2438 MHz 2440 MHz 2442 MHz 2444 MHz 2446 MHz 2448 MHz 2450 MHz 2452 MHz 2454 MHz 2456 MHz 2458 MHz 2460 MHz 2462 MHz 2464 MHz 2466 MHz 2468 MHz 2470 MHz 2472 MHz 2474 MHz 2476 MHz 2478 MHz 2480 MHz
Ch.
37 0 1 2 3 4 5 6 7 8 9 10 38 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 39
Wi-Fi Channel 1
Wi-Fi Channel 6
Wi-Fi Channel 11

Figure 2.2: BLE channel distribution in the 2.4 GHz spectrum
Original source: http://fcs.futureelectronics.com/future-connectivity-solutions-smart-about-bluetooth-smart

The 40 channels are divided into advertising and data channels. The advertising channels are used for connection establishment and short data broadcasts. Any connection traffic is sent through the data channels. There are three dedicated advertising channels and 37 data channels. This is a lower number of advertising channels than the 32 used in Classic Bluetooth and comes with some advantages. Because the radio can only operate on one channel at a time it has to switch channels frequently. And with BLE, only three channels need to be scanned to find a connection partner. This reduces the scanning time significantly and enables both fast connection setup and low energy consumption. In order to minimize collisions with Wi-Fi devices, these advertising channels have been placed in between the three most commonly used Wi-Fi channels. An advertising packet should be transmitted on all advertising channels one after the other. This increases the chance of proper reception.

The data channels, on the other hand, use Adaptive Frequency Hopping (ADFH). Both connection partners agree on a hopping scheme and on a list of blacklisted channels when initiating the connection. During a connection, they will frequently change channels, which allows for great robustness against frequencies with high signal interference and effectively prevents continuous collisions.

2.4 Link Layer (LL)

The link layer is responsible for maintaining stable connections to all of its connection partners and uses a state machine to control its functions. There are five different states: **Standby**, **Advertising**, **Scanning**, **Initiating** and **Connection**. The link layer can only be in one state at a time but some implementations allow the use of multiple concurrent state machines.

One link layer packet format is used for both advertising and data channel packets. The **Access Address** is used to uniquely address a device. For data channel packets it is generated randomly for each new connection while advertising channel packets use a fixed address that is defined in the standard.

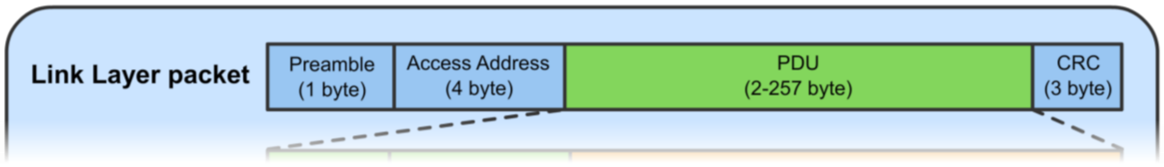


Figure 2.3: Link layer packet format
Original source: (Bluetooth 4.2, Vol. 6, Part B, 2.1)

The **Packet Data Unit (PDU)** that is contained in each link layer packet is different for advertising and data packets. With BLE 4.2, the PDU that was previously limited to 39 bytes now has a maximum size of 257 bytes. This feature, dubbed **LE Data Packet Length Extension**, greatly reduces the protocol overhead when sending large amounts of data as explained in section 9.2.

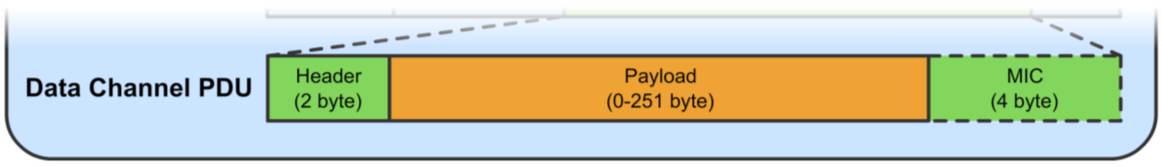


Figure 2.4: Data Channel PDU
Original source: (Bluetooth 4.2, Vol. 6, Part B, 2.4)

The **Data Channel PDU** is split into a header, the user-definable payload and, for encrypted messages, a Message Integrity Check (MIC) field. The header is used to specify whether the packet contains data or a Link Layer Control Message. Link Layer Control Messages are used for management tasks like feature negotiation, maintaining stable connections or encryption control. One of the control messages that is needed for a stable mesh is described in detail in section 6.3.3.3.

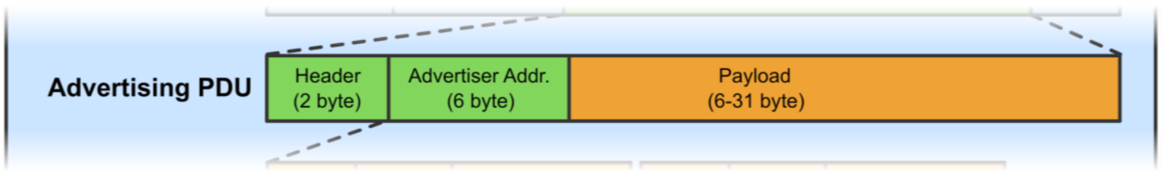


Figure 2.5: Advertising PDU
Original source: (Bluetooth 4.2, Vol. 6, Part B, 2.3)

The PDUs that are sent on the advertising channels use a different header. There are PDUs for broadcasting data (**Advertising PDUs**), requesting and delivering additional data (**Scanning PDUs**) and for initiating a connection (**Initiating PDUs**).

Broadcasting data on the advertising channels is limited by the small size of the PDUs and their structure. With Advertising PDUs it is possible to broadcast around 20 bytes of useful information to surrounding devices. The Scan Response PDU can then be issued to devices that chose to request more data with a **Scan Request**. Connections should be used if bigger amounts of data are exchanged between devices. The previously mentioned size limitations for advertising channel packets originate from the maximum packet length that is defined in the standard and the data structure of Advertising PDUs which consist of a series of different **AD Types** as shown in **Figure 2.6**. Several AD Types exist for common information like the TX power that was used to send a packet or the device type. There is a special AD Type for manufacturer specific data that can be used to include custom binary data that does not fit in another AD Type.

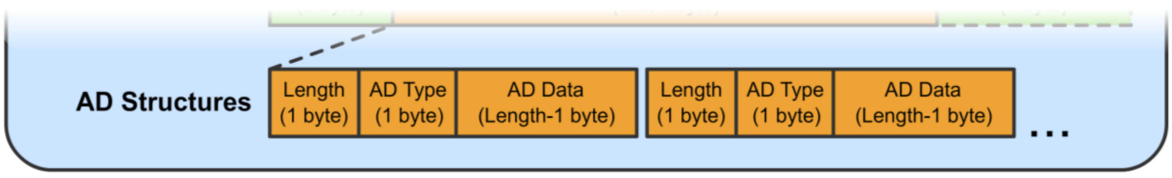


Figure 2.6: AD Structures
Original source: (Bluetooth 4.2, Vol. 3, Part C, 11)

Following is a list of the four different Advertising PDU types that exist according to the current standard and are sent on the advertising channels. Depending on the PDU type, other devices can request more information or connect to the advertising device.

ADV_IND

Used to broadcast data and allows other devices to either request more data or to respond with a connection initiation request.

ADV_DIRECT_IND

This PDU includes the address of a target device but has no payload data. It is used to reconnect to known devices. It can be sent at a very high interval (higher than other advertising channel PDUs) but only for a total of 1.28 seconds, which has been restricted by the standard in order to conserve energy (Bluetooth 4.2, Part B, 4.4.2.4).

ADV_NONCONNECT_IND

Other devices may not initiate a connection or request more data. The purpose of this PDU is to broadcast data to surrounding devices.

ADV_SCAN_IND

This PDU is sent when the device does not allow connections but has more data that it will share when it receives a scan request.

Advertising packets are broadcasted at an interval which can vary from 20 ms to about 1 second. A random delay of up to 10 ms at the end of each interval reduces the likeliness of continuous packet collisions from two devices. The time between two intervals is called **Advertising Event**. At the beginning of each **Advertising Interval**, the device will consecutively broadcast an advertising packet on each of the advertising channels. After each packet, it will listen for a short amount of time on the same channel. If it does not receive a valid response within this timeframe, it will continue to the next channel. This process is illustrated in **Figure 2.7**.

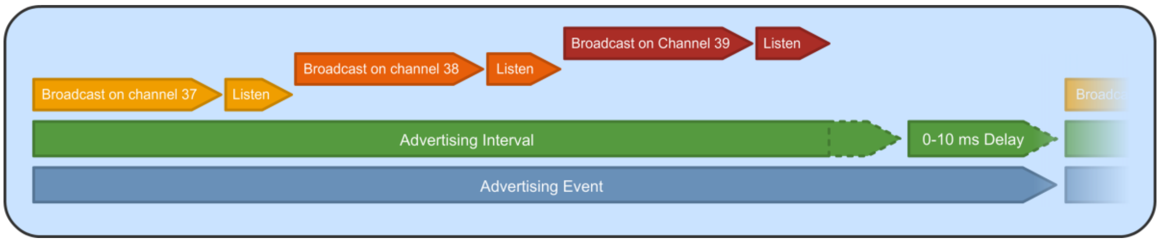


Figure 2.7: Advertising on three different channels
Original source: (Bluetooth 4.2, Vol.6, Part B, 4.4.2.2)

Other devices can respond either with a **Scan Request PDU (SCAN_REQ)** or a **Connection Initiating PDU (CONNECT_REQ)**. Valid responses depend on the preceding Advertising PDU. The response must be sent 150 μs after receiving the advertising packet. This time is known as the Inter Frame Space (T_IFS), a space that must be maintained between any two subsequent packets.

A Scan Request is answered on the same channel with a **Scan Response PDU (SCAN_RSP)** that contains additional data while a Connection Request will be negotiated on the data channel that was received in the Connection Request packet.

2.5 Generic Access Profile (GAP)

The Generic Access Profile provides higher level functionality on top of the link layer and is responsible for dictating device interaction. Most BLE implementations provide an API to interface with its functionality.

There are four different roles that a device can assume with GAP:

Broadcaster

A device that broadcasts data on the adverting channels. A good example is a temperature sensor that broadcasts the measured temperature.

Observer

An Observer listens for data on the advertising channels. This could be a temperature measurement of the above mentioned temperature sensor.

Central

A Central is a device that is capable of initiating connections to multiple other devices. It is responsible for managing these connection and is therefore the **Link Layer Master** that dictates the connection parameters. Because of this, the Central needs more resources in terms of memory, processing power and battery capacity when compared to a Peripheral.

Peripheral

A Peripheral is not able to initiate connections and must advertise its presence in order to be detected by a Central. If a Central sends a Connection Request, the Peripheral will assume the **Link Layer Slave** role in which it must obey the connection parameters that have been chosen by the Central.

Bluetooth Low Energy provides two possibilities of data exchange. A Broadcaster can reach multiple Observers by sending data on the advertising channels or two devices can use a connection to exchange data. It is important to mention that the Central and Peripheral role have nothing in common with the data flow direction. Data flow is possible in either direction after a connection has been established.

A typical connection setup begins with a Peripheral that wants to be discovered. It will then broadcast advertising packets with a flag that makes it discoverable to others. At the same time, there must be a Central that is either in the Scanning or the Initiating state. If it is in the Initiating State, it can respond to one of the advertising packets with a connection request.

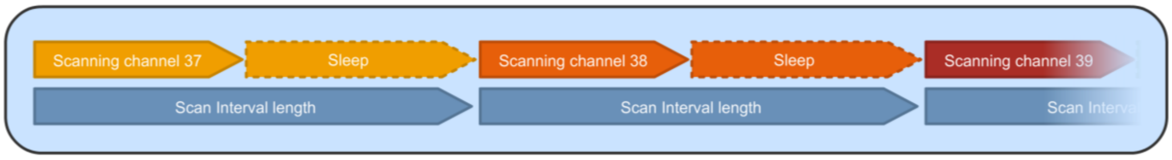


Figure 2.8: Scanning or Initiating state
Source: Original work

Figure 2.8 shows that in both the Scanning and Initiating state, there are three parameters that define how often and how long the advertising channels are observed. The **Scan Interval** defines how often a scan is started while the **Scan Window** regulates the scan duration. An optional timeout will specify the time after which the scan will be stopped.

A Central in the Scanning state does not immediately connect to a Peripheral. It only observes the advertising channels and reports all packets to the application, which can then choose - or let the user choose - to connect to any of the advertisers. Either the address of a single connection target or a whitelist is then handed to the controller after which the Initiating state is entered.

In the Initiating state, the Central must periodically activate its receiver and wait for an advertising packet from any of the whitelisted devices. It will then immediately react and send a Connection Request. Because of the very strict timing requirements, it is not possible for the application to react on an advertising packet in the Scanning state. This task must be handled by the controller. But the link layer will not forward any packets to the application in the Initiating state.

The Central includes some parameters in the **Connection Initiating PDU** which the Peripheral has to abide to.

The **Transmit Window Offset** and **Transmit Window Size** allow the Central to specify the time and length of the first data transmission. This is essential because the Central can have other scheduled connection events which cannot overlap with a new connection.

The connection request packet also include a **Channel Map** and a **Hop Increment**. Together, these two parameters define the data channel hopping scheme that is used for subsequent connection events (ADFH).

Connection Interval and **Slave Latency** set the anchor points in time where master and slave are able to exchange data. The Connection Interval must be between 7.5 ms and 4.0 s. The **Connection Supervision Timeout** is used to declare a connection lost if one of the devices does not receive a valid packet from its partner within this timeframe. A very unique feature of BLE is **Slave Latency** which allows the Peripheral to sleep for a number of connection events if it does not need to receive or send data.

Depending on the communicated **Master Sleep Clock Accuracy** and its own sleep clock accuracy, the slave will need to widen its reception window at every connection event in order to receive packets. It will then resynchronize to the master's clock upon reception of each received packet. This is further explained in section 6.3.3.2.

The Peripheral can provide the Central with its own set of **Preferred Connection Parameters** in order to achieve e.g. a low power usage or a low latency, but the Central is free to ignore these parameters.

In a connection, master (LL Master) and slave (LL Slave) can alternate sending packets each connection event. If both have no more data available, they can sleep for the rest of the connection interval. A header flag in each packet indicates if more data is available.

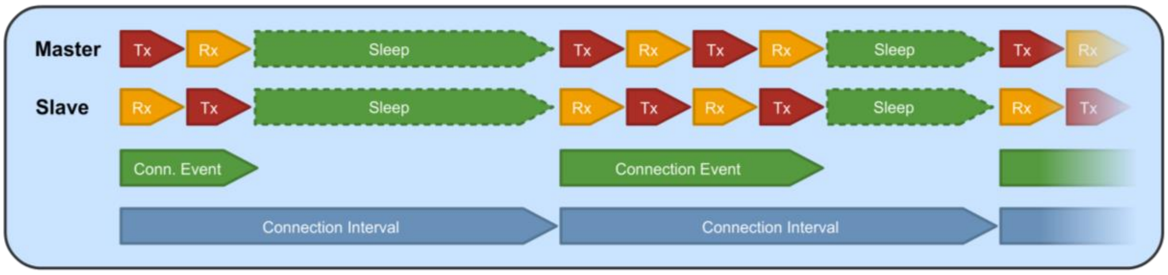


Figure 2.9: GAP Connection Intervals
Original source: (Townsend 2014, § 2)

This chapter only discusses direct connections between two devices. Other connection topologies are discussed in more detail in section 6.3.3.6.

2.6 Security Manager

The Security Manager is responsible for providing encryption and authentication in connections. When two devices in a connection are **paired**, a temporary security key is generated but not stored in long-term memory. Both devices save the exchanged security key persistently after they **bond**. Connections are then re-established with the stored key.

There are three different pairing algorithms. For first one, called "Just Works", both devices will pick the same seed of 0 for generating an encryption key. A 6 digit passkey will be displayed on both devices if more security is required. This has to be compared by the user. The only method that provides actual protection against man-in-the-middle attacks must use an out of band method. Bluetooth Low Energy connections can be easily compromised when one of the first two methods is used (Mike Ryan 2013). Additional information on security can be found in section 9.5.

2.7 Attribute Protocol (ATT)

The Attribute Protocol is a simple stateless protocol for exchanging data between two devices.

The protocol uses **Universally Unique Identifiers (UUIDs)**, defined in an ISO standard, with a size of 128 bit. They are used to identify Services, Characteristics, Descriptors and more throughout the protocol. Additionally, the BLE specification allows shortened UUIDs with 16 or 32 bit. To derive the full UUID, these bits are inserted into the BLE base UUID which is "xxxxxxxx-0000-1000-8000-00805F9B34FB". UUIDs that are not specified by the Bluetooth SIG are vendor specific and must be used in their 128 bit representation.

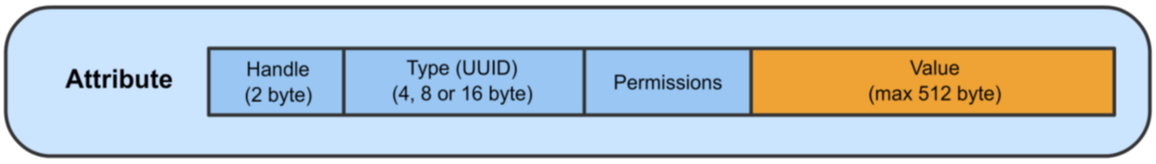


Figure 2.10: Structure of an Attribute
Source: Original work based on (Townsend 2014, § 2)

An Attribute consists of four parts:

Handle

A device-unique 16 bit identifier that is used to access the Attribute during data exchange.

Type / UUID

A UUID is used to specify the type of information represented by the Attribute. There are predefined UUIDs that classify an entry as a Service or Characteristic. Then, there are other UUIDs for certain Profiles that are specified by the Bluetooth SIG. These are for example the Heart Rate Measurement Profile or Temperature Profile which define collections of Services and Characteristics. Finally, there are vendor specific UUIDs that are not defined in the official specification.

Permissions control how the Attribute data can be accessed and how it should be handled. There are three types of Permissions:

The **Access Permission** can be either set to no access, read, write, or read & write. This setting applies to all devices that want to access the value.

Some values can be accessed over non-encrypted connections but it is possible to specify **Encryption Requirements** so that data can only be accessed over an encrypted connection or over an encrypted connection with additional authentication.

The last one is the **Authorization** permission which regulates whether access requires user consent or not.

Value

With a maximum size of 512 bytes, this is where the actual data is stored. Some values are human readable or contain sensor readings while others are used for structural purpose, which is explained in the next section.

2.8 Generic Attribute Profile (GATT)

The Generic Attribute Profile is built on top of the Attribute Protocol and provides a structured way of dealing with data. GATT describes two different roles, the **server** and the **client**, which are independent from GAP roles and should not be confused with the Peripheral or Central role. After a GAP connection has been set up, data transmission is possible in both directions and each device can act as a GATT server or a GATT client simultaneously.

The server stores a GATT table with the information that it can make available to a client upon request. It does have a concept of hierarchical order although it is internally structured as a table. Each entry in this table is an Attribute with a handle, a UUID, permissions, and a value. These entries are exchanged between server and client if granted by the permissions.

A GATT table contains **Services**:

A Service is a collection of Characteristics that are grouped together to serve a common purpose. For example a measurement Characteristic and the corresponding body sensor position Characteristic in the Heart Rate Measurement Profile.

A Primary Service is declared with a UUID of 0x2800. The value must be read-only and must be set to a unique UUID that is then used to identify this Service.

Each Service contains 0-n **Characteristics**:

> Characteristics are containers for data that can be read, written or both. They consist of at least two Attributes. The first Attribute declares the Characteristic and contains the value while the second Attribute contains a set of additional properties.

Each Characteristic can contain 0-n **Descriptors**:

> Descriptors are used to add metadata to a Characteristic. There are multiple GATT defined Descriptors. Examples are the Characteristic User Descriptor, which is used to add a human readable description to a Characteristic or the Client Characteristic Configuration Descriptor (CCCD) which is used to enable Notifications or Indications for a Characteristic.

In order to read or write a value, the client must first acquire the Attribute handle. This is done in a process called Service and Characteristic discovery. It is a challenge and response scheme where the client requests a range of Services and the server responds with a list of its Attribute handles and UUIDs. After evaluating these, the client can request the Characteristics of one or multiple Services. Because this is a power and time consuming process, Attribute handles can be cached between connections if two devices are bonded. However, this means that Attribute handles must persist between connections. If this is not the case, the Service Changed Characteristic, explained later in this chapter, should be used.

There are several possibilities for reading and writing Attributes. A simple read can read up to 20 bytes of an Attribute value. If the value is bigger, multiple long reads can be issued with an offset to query the full value. Writing can be either acknowledged or not. An acknowledged write **(Write Request)** will guarantee that the value has been written while a write without response **(Write Command)** does not have a guarantee that the client received and accepted the message. A third option, the **Reliable Write**, enables atomic modifications and issues a final commit after many writes to multiple Characteristics.

The above mentioned commands can only be initiated by a GATT client. Server initiated updates, on the contrary, allow the server to send unacknowledged **Indications** or **Notifications**. Indications are used to tell the client to fetch a value that has changed. Notifications are used to directly send the updated value in the same message. Both must at first be enabled by the client by writing the CCCD of the corresponding Characteristic. This ensures that he is in possession of the required Attribute handle and that he is interested in receiving these updates. When enabled, the server can send update messages at any time during an active connection.

There are two Services that must be implemented in every device. The GAP Service with its UUID of 0x1800 and the GATT Service, which uses 0x1801.

The **GAP Service** is freely accessible to all devices over an unencrypted connection. It is used to communicate the device name and appearance (e.g. phone, computer, smartwatch) as well as the **Slave Preferred Connection Parameters**. The last-mentioned parameters can be read by a Central before establishing a connection with a Peripheral in order to select suitable connection parameters.

The **GATT Service** can only contain the previously mentioned **Service Changed Characteristic**. This Characteristic is optional and cannot be read or written, but can be sent in Notifications if these have been activated by the client. A Notification is sent after internal changes to the GATT table structure have occurred and contains a range of handles that the client should rediscover.

3 Devices and Platforms

3.1 Smartphones

During the initial evaluation, one of the key differentiators for Bluetooth Low Energy was its availability on smartphones. It is therefore necessary to assess the state of current implementations on major smartphone operating systems and see which roles mobile phones could play in a mesh network. A full list of Bluetooth Smart Ready products is available at (Bluetooth SIG 2015a).

3.1.1 Android

Bluetooth Low Energy has been introduced with Android 4.3, which was released to the public in mid-2013. Most major handset manufacturers include a BLE compatible chipset in their phones nowadays. One feature that was notably absent from the Android 4.3 API was the ability to use the phone in the Peripheral or Broadcaster role. This has been solved with the introduction of Android 5.0 where a device is now able to advertise and act as a Peripheral (Android Open Source Project 2015). As a result, this should enable devices to access a mesh topology and maybe even take part. There are some prominent API restrictions, for example the limitation to only set the advertising or scan intervals to three predefined values without having fine granular access to the timings. But it is possible to let apps advertise while running in the background. As of February, the market share for Android 5.0 is still small at 1.6% of all Android devices (c|net 2015).

3.1.2 iOS

Apple has been the first to introduce Bluetooth LE capabilities to its smartphones with the iPhone 4S and iOS 5 (Bluegiga 12/1/2011). Since iOS 6, they also support the Peripheral role with support for background advertising. But as with the Android API, it is not possible to gain full access to advertising or scanning timings as these are controlled by the operating system. The BLE design guidelines (Apple Inc. 2013) do also not disclose the intervals used by the implementation and these can change with an update of the operating system.

3.1.3 Windows Phone

A Bluetooth LE API has been made available to developers since Windows Phone 8.1. It includes the ability to communicate with devices that have been connected through the system settings. This does not respect the intentions of the BLE specification and does not allow an app to easily interface with other devices. Currently, there is no support for the Broadcaster or the Peripheral role (Microsoft 2015).

3.1.4 Summary

Most major smartphone platforms have support for BLE which will at least allow them to access the mesh. If the Peripheral role is needed to gain access to the mesh or if an app must be able to initiate connections to unknown devices, Windows Phone will not be able to access the mesh. Implementing a mesh logic as an app on iOS or Android could be possible but may be restricted by some API limitations.

3.2 Chipsets

There are many different chipsets available include the Nordic Semiconductors nRF51 series, the Texas Instruments (TI) CC2540 and CC2640, and the Dialog Semiconductor DA1458x series. The base price per chip usually varies between 1€ and 3€ and has not been taken into consideration. The primary decision focus was the availability of development tools and platform features.

3.3 Development Platforms

When developing a prototype, it is important to choose a development platform that enables fast results and is flexible enough to not pose any restrictions in the process. I will now present a short overview of some available platforms that were available at the time of writing.

Platforms that need an external CPU have not been considered because of the added complexity. Quoted prices have been gathered from (Mouser Electronics 2015) at the time of writing.

3.3.1 Nordic Semiconductor nRF51 Dongle

The nRF51 series is Nordics latest offering for BLE, ANT and custom 2.4 GHz protocols. It includes an ARM Cortex-M0 with 16 MHz, flash memory up to 256 KB and random access memory with up to 32 KB. The 2.4 GHz radio uses GFSK modulation and can do hardware-based on-the-fly encryption. The Bluetooth Low Energy Stack is written in software and is available in the form of different binary files that are called SoftDevices. Each of these support different functionality and the latest offering does even support multiple BLE roles at the same time. There is a C API available with support for IAR, KEIL and GCC toolchains.

The maximum transmit power is +4 dBm and peak currents during transmissions are 13 mA for RX and 10.5 mA for TX.

There are two development kits available, the nRF51-DK and the nRF51-Dongle. Both have a J-LINK SEGGER debugger chip on board which allows flashing and debugging over USB.

They are available at a price of about 70€ or respectively 50€. There are many additional software tools available as well.

3.3.2 TI CC2540 Dongle

The TI CC2540 has an integrated 8051 core and a flash memory up to 256 KB. The ram size is 8 KB. The BLE stack is written in software and is available royalty-free and open source. Programming is only possible with the commercial IAR Embedded Workbench. There are many additional development tools available.

The maximum transmit power is +4 dBm. The TX peak current is 27 mA at 0 dBm and 20 mA for RX.

There are some development kits available, including a CC2540 USB Dongle for about 50€. An external CC Debugger has to be bought as well for flashing and debugging the dongle which adds an additional 50€.

As of February 2015, after programming had started, there was a new development kit available that is based on the CC2650 chipset which now includes an ARM processor and an improved stack.

3.3.3 Bluegiga BLED112

The Bluegiga USB dongle is built on the basis of the TI CC2540 chipset. This means that is identical in terms of technical specifications. The development workflow is however different because Bluegiga distributes a proprietary stack which is closed source and is interfaced via a custom API. There is a scripting language available which has been deemed too restrictive and is very slowly executed.

3.3.4 Dialog Semiconductors

The DA1458x series is based on an ARM Cortex-M0 core as well and is available in a variety of RAM and ROM configurations in a small package size. It supports all BLE roles but only one at a time and not concurrently (Dialog Semiconductors 2015). The current consumption is very low at about 4 mA for both RX and TX which makes it very attractive for small battery-powered devices.

3.3.5 Summary

After some evaluation, the nRF51 Dongle has been chosen for prototype development because of the benefits that it provides over its competitors. Moreover, a contact person was already familiar with the nRF51 development workflow, which made the final decision easy. Some of its standout features are:

- The dongle is small and can be used easily with a power source attached, while also enabling debugging and flashing over USB.
- Multiple dongles can be used at once without requiring an external debugger.
- The dongle has a relatively low price point.
- It allows multiple toolchains, one of these being open source and free.
- The power consumption is good.
- The extensive C API should not pose many limitations.
- The Bluetooth Low Energy stack has support for concurrent roles.

4 The Development Platform

This chapter provides a closer look at the development environment of the nRF51 series. This includes the used hardware, the manufacturer-provided Bluetooth Low Energy stacks, programming and development environment, available documentation, and the available tools.

4.1 Hardware

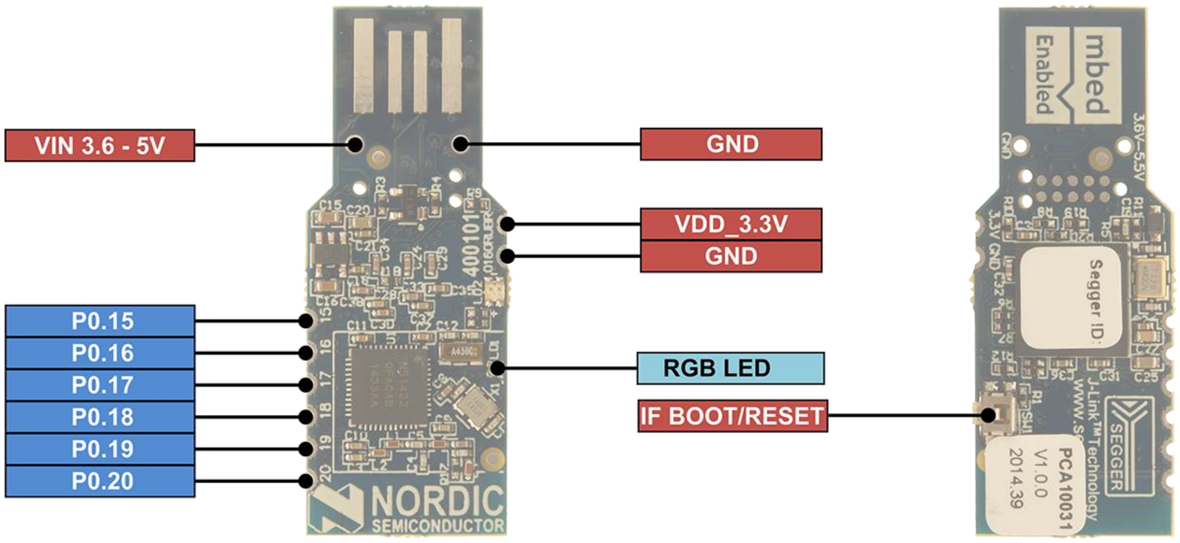


Figure 4.1: nRF51 dongle
Source: https://www.nordicsemi.com/eng/Products/nRF51-Dongle

The nRF51 Dongle is based on the nRF51422 chipset. The board itself is called PCA10031 and two board versions, V1.0.0 and V1.1.0, were used during development. The development board is very simple with only a hard-wired reset button, a power LED, a user-programmable RGB LED and some solder pads. It contains the nRF51422 chipset which is connected to an antenna and an external 16 MHz quartz oscillator. The board features a USB port for programming and enables flashing either via the mbed platform[1] or by using the on-board SEGGER J-Link debugger chip.

The nRF51422 SoC is available in several configurations and revisions. One of the used chipsets had the exemplary product code nRF51422_QFACAB:

- QF stands for QFN packaging, which determines the chip size and how it is soldered to the board.

[1] https://mbed.org

- AC is the variant code, which in this case is the version with 32 Kb ram and 256 KB flash memory.
- AB is the chip revision which increments with every new version. This is important to know because different chip versions have different known anomalies. Three different revisions have been used during development.

The nRF51 series SoC includes an ARM Cortex M0 processor clocked at 16 MHz. All types of memory (ram, flash and registers) can be accessed through a 32 bit address range that unifies access to all device functions. The chip integrates several so called peripherals. These are hardware circuits that target a specific purpose. (The term should not be mixed with the Peripheral role in the Bluetooth standard which I have decided to write with a capital “P”.) Some of the included peripherals are:

- The radio peripheral with an integrated whitener
- AC/DC converter
- AES hardware encryption engine
- General Purpose Input and Output (GPIO) ports

The peripherals can be controlled through the unified address range by setting the appropriate registers. Some of them, like the radio, will have access to the random access memory for transferring larger amounts of data.

The on-board SEGGER J-Link debugger is connected to the debug pins of the nRF51422 chip and enables flashing, has support for up to four hardware debug breakpoints, allows debug stepping, supports UART pass-through, and much more.

4.2 SoftDevices

Nordic offers a handful of so called SoftDevices. These are precompiled software libraries that implement the BLE and ANT protocol stacks. They register some hardware interrupt handlers in order to be able to support a real-time and deterministic behaviour. Furthermore, they restrict access to certain registers and peripherals which they need to handle the protocols.

The SoftDevices are provided as binary files together with a set of header files that are utilized to interface with their functionality. The handlers and library functions are used to control the entire Bluetooth Low Energy stack. Some peripheral functions are accessible through the provided header files as well. The SoftDevice then guards this access and can determine at which time it does not need the peripheral itself.

Some SoftDevices additionally provide a **Timeslot API** which allows to take control over all hardware peripherals in times where the SoftDevice is not active. The application is notified of available timeslots in advance. It must have finished its tasks before the end of that timeslot to not crash the system. This API can be used to implement a custom communication protocol that runs concurrently with BLE. Close attention has to be paid to maintain the right timings so that the two protocols do not collide.

Currently, the following SoftDevices are available:

- The **S110** is currently available in version 7. It is the best developed SoftDevice as it was the first-to-market that implemented a BLE Peripheral stack. In its latest version, it does also support non-connectable advertising while being in a connection.
- With the **S120**, Nordic introduced support for the Central side of the BLE specification. The latest available production release is version 2.
- The **S130** was available in version 0.5-alpha during development and combines the functionality of the S110 and S120. The alpha version had some problems that affected the development but these were solved with a later version. At the end of development, it became available as a 1.0 production release.
- The **S210** provides an ANT stack to developers.
- Finally, the **S310** combines a BLE Peripheral and an ANT stack.

Additionally, there are some modified SoftDevices available that have been bundled with applications by Nordic Semiconductors:

- **IoT SoftDevice**: Enables IPv6 Routing over BLE as introduced in the Bluetooth 4.2 specification. The application is founded on the S110 and can only act as a Peripheral. A preconfigured Raspberry Pi image that can be used as a gateway router is available as well. With a connected nRF1 Dongle, the Raspberry Pi is then able to route IP packets from its network interface through the Bluetooth Low Energy protocol.
- **nRF Sniffer:** Can sniff advertising and connection packets and is explained in more detail in section 4.5.4.
- **Master Control Panel Firmware:** This firmware is based on the S120 SoftDevice and interfaces with the Master Control Panel desktop client which is further explained in section 4.5.1.

4.3 Programming Language and IDE

The **nRF51 SDK** includes header files for the C programming language and comes preconfigured for the Keil toolchain. Keil has an integrated C++ compiler and linker and makes it possible to combine both languages with some effort. As of the latest SDK versions, support for the GNU GCC toolchain has been improved. Additional libraries are included to provide some frequently needed functionality such as a FIFO buffer, a library for UART communication and a board support package which abstracts the layout of the development board from the implementation.

There are several possible workflows which do require a different amount of setup effort. Early development was started with Keil but the project was later converted to work with a GCC toolchain in combination with Eclipse because of its extended feature set.

Keil is an IDE that is currently owned by ARM. A free version is available with support for a limited code size. ARM was very kind to provide me with a licence for the time of my thesis.

Developing with **Eclipse** and an **embedded GCC toolchain** is possible and very comfortable but the debugger is still very error-prone and the setup effort is very high.

A third option would have been to develop the code on any x86 platform and then link it against a **serialization library** that became available during middle of development. This library uses the same header files as the SoftDevice, which means that the written code should be easily portable to the nRF51 platform. But this would have ruled the usage of the S130 because at that time, serialization libraries were only available for the S110 and S120 SoftDevices.

4.4 Documentation

While there is a lot of information and documentation available, it is not always easy to find. The documentation is separated in different documents and online resources and some of them are structured in a way that make the first steps difficult. After many problems, it was decided to cross-read all of the available documentation in order to gain an understanding of what is available and where it can be found. The following resources are all mandatory during development:

- The **nRF51 SDK Documentation** is available online[2] for each SDK version and is split into the different SoftDevices. The SD130 is a merge between the S110 and the S120 and did not have a dedicated documentation during development. The structure of the documentation is often complicated and seldomly provides links between topics.
- The **S130-SDS** SoftDevice specification details the functionality and some implementation details for the S130 SoftDevice. Because of the alpha state of the SoftDevice, the documentation is still unfinished and much of the specification is not yet finished.
- **S130 Release Notes:** Bug fixes and changes from previous SoftDevice versions, limitations and known bugs can be found separately with the release of a new version of the SoftDevice.

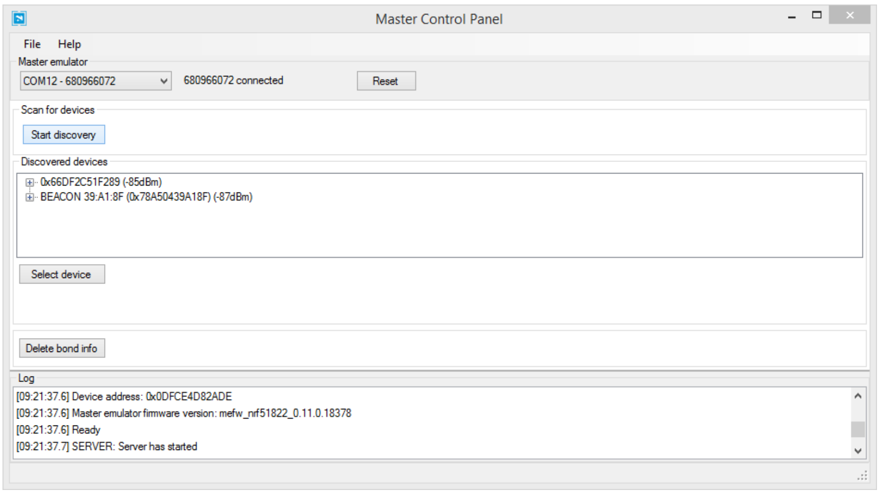
[2] http://developer.nordicsemi.com/nRF51_SDK/doc/8.0.0

- Low level functionality that only applies to the CPU can be found in the **ARM Cortex M0 Processor Documentation**. This is important for aligned memory access and other low level functionality.
- **nRF51422 Product Specification:** The peripherals and the available hardware in the chipset are specified in the product specification. This includes the operating conditions, voltage, current draw, pinout description and much more.
- **nRF51 Series Reference Manual:** Detailed description of the chipset with its peripherals and functionality from a programmer's point of view can be found in the manual.
- **nRF51422 Product Anomaly Notice:** Depending on the chip version, there may be different hardware anomalies. Sometimes these affect large portions of the functionality. The expected behaviour and workarounds are collected in this documentation.
- The available **Example Projects** are an invaluable source for information that often cannot be found in any of the manuals. This includes the event handling, best coding practices and more that is not documented elsewhere.
- Up to date information can be found in the **Nordic Developer Zone**[3] which is filled with user questions and answers.

4.5 Tools

The next sections are intended to provide a short overview of the available tools that have been very helpful in the development of the prototype.

4.5.1 Master Control Panel



Master emulator: COM12 - 680966072 — 680966072 connected

Discovered devices:
```
0x66DF2C51F289 (-85dBm)
BEACON 39:A1:8F (0x78A50439A18F) (-87dBm)
```

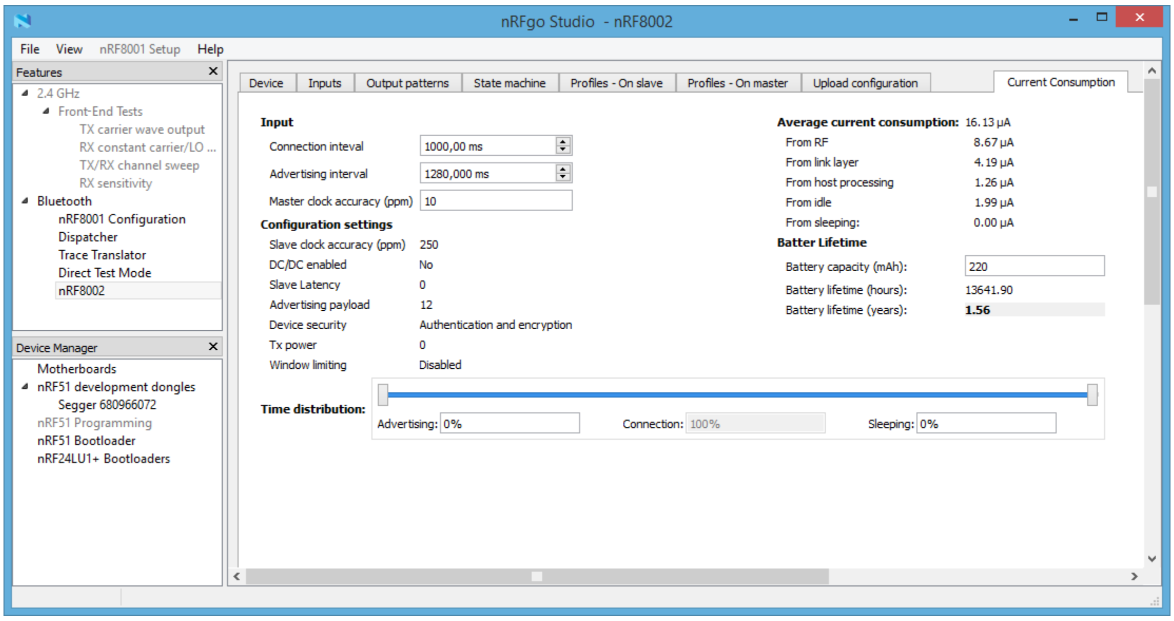
Log:
```
[09:21:37.6] Device address: 0x0DFCE4D82ADE
[09:21:37.6] Master emulator firmware version: mefw_nrf51822_0.11.0.18378
[09:21:37.6] Ready
[09:21:37.7] SERVER: Server has started
```

Figure 4.2: Master Control Panel
Source: Application screenshot

[3] https://devzone.nordicsemi.com

The Master Control Panel is a helpful tool for testing Peripherals. The desktop client can flash an interfacing firmware image on a development dongle which will then act as a Central. It can do all common tasks like scan for advertising devices, connect or bond to them and enumerate their Services and Attributes. It is also possible to create a GATT table with Services and Characteristics that are then accessible to connected Peripherals. One of the downsides is, that it is based on the S120 SoftDevice and can therefore not be configured for advertising or to accept incoming connections.

4.5.2 nRF Go Studio



Input

Connection inteval	1000,00 ms
Advertising interval	1280,000 ms
Master clock accuracy (ppm)	10

Configuration settings

Slave clock accuracy (ppm)	250
DC/DC enabled	No
Slave Latency	0
Advertising payload	12
Device security	Authentication and encryption
Tx power	0
Window limiting	Disabled

Average current consumption: 16.13 µA

From RF	8.67 µA
From link layer	4.19 µA
From host processing	1.26 µA
From idle	1.99 µA
From sleeping:	0.00 µA

Batter Lifetime

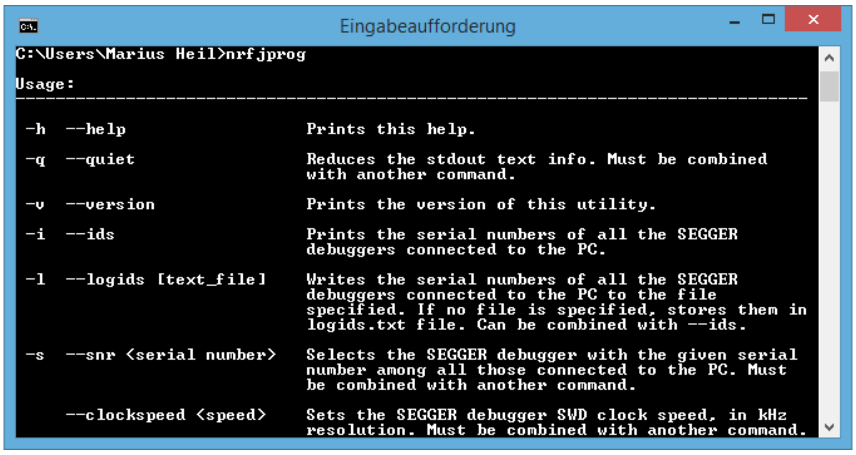
Battery capacity (mAh):	220
Battery lifetime (hours):	13641.90
Battery lifetime (years):	**1.56**

Time distribution: Advertising: 0% Connection: 100% Sleeping: 0%

Figure 4.3: nRFgo Studio
Source: Application screenshot

A typical firmware for the nRF51422 dongles consists of a bootloader, a SoftDevice and the application. There are some special registers that serve as a jump table and are required by the chipset to correctly initiate the boot sequence. The nRFgo Studio is one possibility to flash the dongles in an easy manner and fill these registers. It does additionally provide a simulation for the energy consumption of two older Nordic chipsets that can be helpful to estimate the current draw for different advertising or connections scenarios.

4.5.3 Nrfjprog Utility

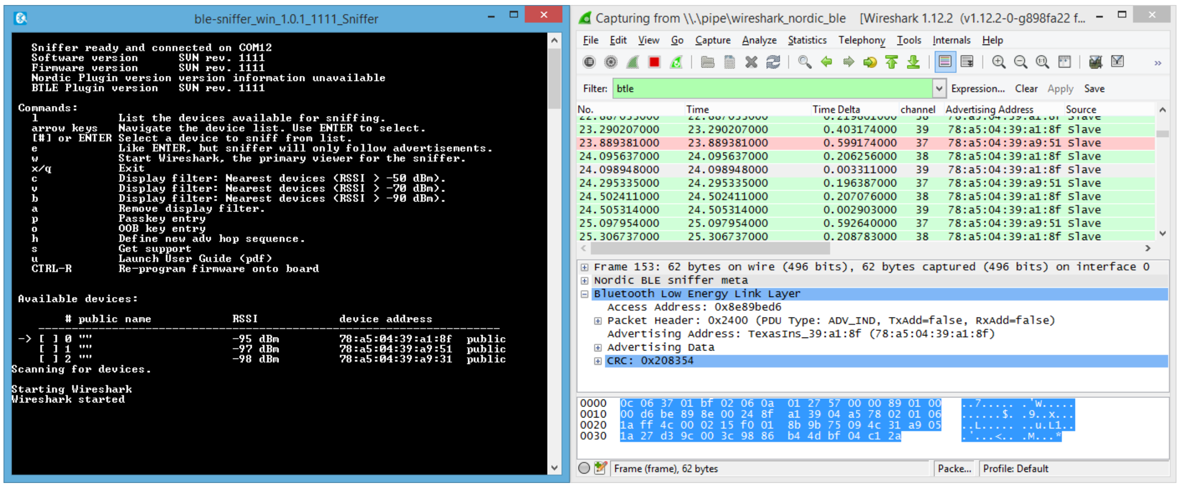


```
C:\Users\Marius Heil>nrfjprog

Usage:
-------------------------------------------------------------------------------

 -h  --help                   Prints this help.

 -q  --quiet                  Reduces the stdout text info. Must be combined
                              with another command.

 -v  --version                Prints the version of this utility.

 -i  --ids                    Prints the serial numbers of all the SEGGER
                              debuggers connected to the PC.

 -l  --logids [text_file]     Writes the serial numbers of all the SEGGER
                              debuggers connected to the PC to the file
                              specified. If no file is specified, stores them in
                              logids.txt file. Can be combined with --ids.

 -s  --snr <serial number>    Selects the SEGGER debugger with the given serial
                              number among all those connected to the PC. Must
                              be combined with another command.

     --clockspeed <speed>     Sets the SEGGER debugger SWD clock speed, in kHz
                              resolution. Must be combined with another command.
```

Figure 4.4: Nrfjprog
Source: Application screenshot

The nrfjprog command line utility can be used for simultaneous flashing of multiple nRF51 dongles. Two batch scripts were written to flash all connected dongles and to reset them upon request. This allowed for a fast development workflow.

4.5.4 nRF Sniffer



```
  Sniffer ready and connected on COM12
  Software version       SVN rev. 1111
  Firmware version       SVN rev. 1111
  Nordic Plugin version version information unavailable
  BTLE Plugin version    SVN rev. 1111

 Commands:
  l            List the devices available for sniffing.
  arrow keys   Navigate the device list. Use ENTER to select.
  [#] or ENTER Select a device to sniff from list.
  e            Like ENTER, but sniffer will only follow advertisements.
  w            Start Wireshark, the primary viewer for the sniffer.
  x/q          Exit
  c            Display filter: Nearest devices (RSSI > -50 dBm).
  v            Display filter: Nearest devices (RSSI > -70 dBm).
  b            Display filter: Nearest devices (RSSI > -90 dBm).
  a            Remove display filter.
  p            Passkey entry
  o            OOB key entry
  h            Define new adv hop sequence.
  s            Get support
  u            Launch User Guide (pdf)
  CTRL-R       Re-program firmware onto board


 Available devices:

       # public name          RSSI            device address
    -----------------------------------------------------------------
 -> [ ] 0 ""                  -95 dBm         78:a5:04:39:a1:8f  public
    [ ] 1 ""                  -97 dBm         78:a5:04:39:a9:51  public
    [ ] 2 ""                  -98 dBm         78:a5:04:39:a9:31  public
Scanning for devices.

Starting Wireshark
Wireshark started
```

Capturing from \\.\pipe\wireshark_nordic_ble [Wireshark 1.12.2 (v1.12.2-0-g898fa22 f...
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

Filter: btle

```
No.            Time           Time Delta   channel Advertising Address Source
23.290207000   23.290207000   0.403174000  39      78:a5:04:39:a1:8f   Slave
23.889381000   23.889381000   0.599174000  37      78:a5:04:39:a9:51   Slave
24.095637000   24.095637000   0.206256000  38      78:a5:04:39:a1:8f   Slave
24.098948000   24.098948000   0.003311000  39      78:a5:04:39:a1:8f   Slave
24.295335000   24.295335000   0.196387000  37      78:a5:04:39:a9:51   Slave
24.502411000   24.502411000   0.207076000  38      78:a5:04:39:a1:8f   Slave
24.505314000   24.505314000   0.002903000  39      78:a5:04:39:a1:8f   Slave
25.097954000   25.097954000   0.592640000  37      78:a5:04:39:a9:51   Slave
25.306737000   25.306737000   0.208783000  38      78:a5:04:39:a1:8f   Slave

⊞ Frame 153: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0
⊞ Nordic BLE sniffer meta
⊟ Bluetooth Low Energy Link Layer
    Access Address: 0x8e89bed6
  ⊞ Packet Header: 0x2400 (PDU Type: ADV_IND, TxAdd=false, RxAdd=false)
    Advertising Address: TexasIns_39:a1:8f (78:a5:04:39:a1:8f)
  ⊞ Advertising Data
  ⊞ CRC: 0x208354

0000  0c 06 37 01 bf 02 06 0a  01 27 57 00 00 89 01 00   ..7..... .'W.....
0010  00 d6 be 89 8e 00 24 8f  a1 39 04 a5 78 02 01 06   ......$. .9..x...
0020  1a ff 4c 00 02 15 f0 01  8b 9b 75 09 4c 31 a9 05   ..L..... ..u.L1..
0030  1a 27 d3 9c 00 3c 98 86  b4 4d bf 04 c1 2a         .'...<.. .M...*

Frame (frame), 62 bytes    Packe...    Profile: Default
```

Figure 4.5: nRF Sniffer (left) with Wireshark (right)
Source: Application screenshot

In the early stages of developing it is crucial to have a method for observing the radio behaviour. It is important to know which and how many packets are sent and what information are contained. This is where the nRF Sniffer comes in handy. Just as the Master Control Panel, it is a combination of desktop software and a firmware image for the nRF51 dongle. The dongle will pick up adverting packets and is also able to follow the hop sequence of two connected devices by sniffing the initial connection request packet. It is not able to follow a connection without receiving the connection request like crackle.

5 Evaluating Existing Technology

Many terms are used to describe wireless mesh networks. The term wireless ad hoc network for example describes a network "where all devices have equal status […] and are free to associate with any other ad hoc network device in link range" (Wikipedia 2015). These devices, called nodes, make up the network without the need for further infrastructure. Data is transmitted from node to node until it reaches its destination; each data transmission is called a hop. In most scenarios, the available data is collected at central points or there is a central controlling instance that is connected to another network technology. This device is then called **sink** or sometimes gateway.

Other technologies use controlling devices or hubs to manage the network. These are not called wireless ad hoc networks but the network they form is usually called a mesh network as well. One example is ZigBee with its routers and nodes.

An extension of the wireless ad hoc network are the wireless sensor network (WSN) and wireless sensor/actuator network (WSAN) that use nodes with additional hardware to measure or influence the environment.

There are a myriad of mesh technologies available and one thing that can be observed is that most are targeted at one specific use-case. This chapter will introduce the reader to the uniqueness of some mesh technologies available to date. They each have different characteristics and target varying use-cases. Many mesh networks have been evaluated and some of the notable ones are listed in conjunction with their distinctive features. Features that could help in the design process of a mesh network on top of Bluetooth Low Energy have been collected and considered for the final design. While building upon an accepted and widely used standard helps to avoid compatibility problems with different devices and future versions it can pose problems and restrictions because the protocol was not originally built to support a mesh network. These problems have to be solved.

5.1 Scientific WSNs

5.1.1 Research Platforms

The Telos platform and its successor TelosB are widely used in researching wireless sensor networks. Together with MicaZ, these are hardware modules that typically include multiple sensors, a processor and an IEEE 802.15.4 radio with the TinyOS operating system. The average cost is around 80 Euro (Advanticsys 2015). The nRF51 platform already includes a BLE stack and the required hardware, so there was no need to use an additional operating system.

5.1.2 Dozer

Dozer is a data gathering protocol for sensor networks that has been proposed in 2007 (Burri et al.). Despite its age, it has some interesting proposals. The network is able to stay alive for 5 years on two AA batteries when data is sent on a 2 minute schedule. The network forms a tree topology and also uses TinyOS, which is a frequently used real-time operating system in sensor networks. It uses time multiplexing but does not support different frequency channels. The used clocks have a drift of ±30-50 ppm. Because the timings are only synchronized between directly connected nodes, packets from different nodes might collide. To counter this problem, a seed is distributed between parent and child nodes which is used to initialize a pseudo random number generator on both sides. These pseudo random numbers are then used to add a mutually known offset at the end of every transmission event. Because this seed is different for every connection, the probability of multiple collisions in a row is very low. The network does only support routing to one sink but has a simple backchannel which works by broadcasting to all nodes, even if only one node is to be reached. In order to transmit data, each node buffers the received messages and waits for an explicit ACK message before clearing the transmission cache. But if a node is lost, the unsent messages will be lost as well.

In order to construct a data gathering tree, each node must know its hop distance from the sink and its number of directly connected children (load). These values are broadcasted in regularly scheduled beacon messages that act as an entry point for disconnected nodes. A joining node can then use a rating function to determine the best node to connect to. The constructed tree topology is very beneficial for message routing as it does not pose a problem with infinite cyclic routing. It is however limited to exactly one sink that is used as a reference for the algorithm.

The dozer technology has been used for gathering data from glacial movement and long-term behaviour in the Permasense project (Permasense) and many other networks have borrowed from its fundamental ideas.

5.1.3 Pegasis

There are different topologies for building an ad hoc network. Pegasis (Lindsey, Raghavendra), for example, forms a single chain of nodes in order to complete its task. It assumes that all nodes in a sensor network typically have to send data. Starting from the end of the chain, each node will buffer all the messages that it received from its previous nodes and will add its own data before sending it to the next node. Sending data in batches reduces the number of connection events and improves the overall performance.

The study showed that it reached a better efficiency than a previous effort dubbed LEACH (Heinzelman et al. 2000) in terms of energy depletion in single nodes. But at the same time that means, that the average latency to reach an arbitrary node will be a lot higher than in a tree topology. Furthermore, nodes that do not have to send any data will experience battery depletion as well. This topology is therefore not usable for unknown data traffic and different use-cases other than data collection.

5.2 Consumer Technology

5.2.1 Wirepas Pino

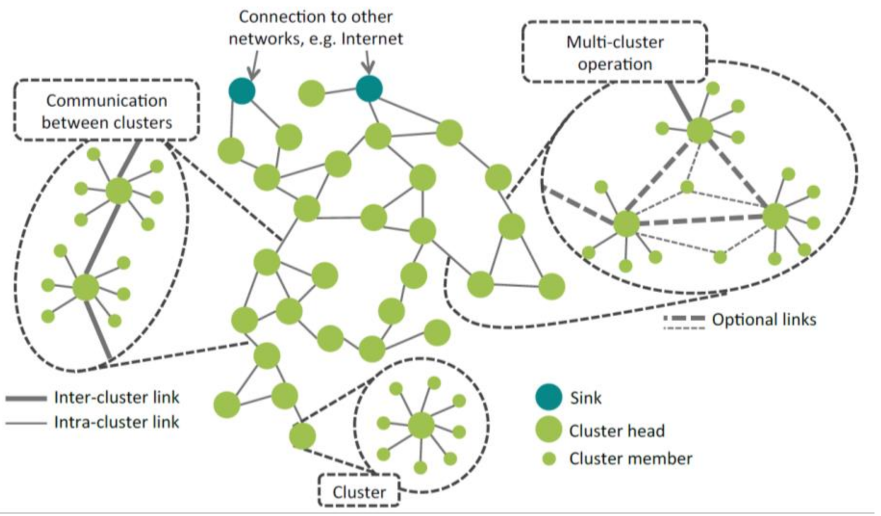


Figure 5.1: Wirepas Pino clustering
Source: (Wirepas Oy 2014)

The core of Wirepas Pino has been developed since 2000 at the Tampere University of Technology in Finland and can now be licenced from Wirepas Oy (Wirepas Oy 10/14/2014). It supports a wide range of devices and according to (Nordic Semiconductor ASA 12/8/2014) there is now an implementation available for the nRF51822 as well. Wirepas Pino is a mesh network that uses a custom protocol that is said to support zero-configuration networks which can be run with battery powered devices alone. The network is self-organizing and self-healing with every node being equal in terms of hardware and capabilities. Apart from marketing material, public information on Wirepas Pino is scarce. The technological whitepaper (Wirepas Oy 2014) mentions a few interesting facts about the technology itself.

The network is organized in clusters. Cluster heads are responsible for routing, while other nodes can only communicate with the network through their cluster heads. The network is able to support multiple sinks and dynamic nodes.

The Media Access Layer (MAC) uses time and frequency multiplexing to avoid collisions. Frequencies are observed and blacklisted in case of high traffic. Each cluster head is responsible to manage the channel access for its connected nodes. Packets are buffered in each node and reliable transfer is possible with acknowledged packets.

Node discovery is done by scanning the network for reachable nodes and is enhanced by using neighbour's neighbour information - each node provides its neighbours to others upon request to reduce the needed scan time. Together with the timing information, it is then possible to directly connect to these nodes.

A cost-based routing system with support for different routes based QoS criteria like low-latency or low energy is part of Wirepas Pino. The routing algorithm seems to be optimized towards sinks and not for inter-node routing.

The marketing material suggests that Wirepas offers a good mesh solution based on a custom protocol but this could not be evaluated from a technical point of view because of its closed source nature which may also be one of its biggest disadvantages.

5.2.2 ZigBee & Thread

ZigBee has a long history. It was standardized in 2003 and a revised standard from 2007 is now known as ZigBee PRO. Mesh technology is deeply integrated into the roots of the technology. It has been "designed to meet the increasing demands for low-powered and efficient wireless networking between sensory and control network applications" (Ammari 2013, § 3). It can operate in the 2.4 GHz ISM spectrum and some other bands. The data rate can be up to 250 Kbit/s with a transmission distance between 10-100 m.

The ZigBee protocol is a very mature protocol and is used in many products. It does, however, have some drawbacks. A typical ZigBee network contains exactly one coordinator that is responsible for forming the mesh. Afterwards, it will function as a router. Depending on the size of the network, multiple routers are needed. And as (Ammari 2013, §§ 3.4.2) notes, routers cannot sleep. This means that they should be connected to the power grid. Additionally, setting up a ZigBee network requires some configuration effort and is not always intuitive to consumers. Furthermore, studies have shown that ZigBee uses a higher amount of energy than Bluetooth Low Energy (Dementyev et al. 2013). One last limitations is the restrictive licencing agreement that states that "[t]he ZigBee Specification is available to individuals, companies and institutions free of charge for all non-commercial purposes […]. No part of this specification may be used in development of a product for sale without becoming a member of ZigBee Alliance" (ZigBee RF4CE Specification 1.01) – this involves membership costs.

A new protocol, named Thread, is an effort lead by many companies such as Samsung and the Google-acquired company Nest Labs to revolutionize home automation. The Thread protocol is said to be compatible with existing ZigBee hardware. The standardizing efforts are still ongoing and availability is planned in 2015. There is currently little public information available to non-members.

5.2.3 ANT / ANT+

ANT is an ultra-low-power wireless protocol that makes the same promises as Bluetooth Low Energy. ANT+ builds on top of ANT to provide a commonly agreed set of profiles. With Bluetooth Low Energy, both the protocol and the profiles are covered under the same specification. ANT is currently available in many Sony Xperia handsets and Samsung devices. It is primarily used in fitness trackers but is absent from any other smartphone makes (ANT+ Alliance 2015).

ANT has basic mesh capabilities but relies on custom implementations to solve tasks like routing or connection setup.

5.2.4 BLE Mesh Networks

There are ongoing efforts to develop a good mesh solution that is compatible with the current BLE standard but not much public information is available. Most information is available about CSRmesh, but two companies Zuli[4] and Seed[5] at least claim that they have their own solution as well (EETimes 3/12/2014).

5.2.4.1 CSRmesh

Figure 5.2: CSRmesh development kit
Source: http://www.csr.com/products/csrmesh-development-kit

CSRmesh is available as a developer kit from CSR, a company that is currently being bought by Qualcomm (Forbes 10/21/2014). Their solution is an implementation of a rebroadcasting mesh with a flooding scheme to pass a message through the network by using the advertising channels of BLE. It has primarily been developed for lightning automation but does allow other use-cases as well. Messages are AES encrypted with a network key, which prevents any attacker from forging messages or deciphering them. Replay attacks are prevented by using sequence numbers. Every device has a unique 2 byte device id and can take part in multiple networks at

[4] http://www.zuli.io
[5] http://www.seedlabs.io

once by storing several network keys. It is also possible to access devices in groups by assigning group ids. This is useful when e.g. all lights in one room should be switched on at once. Two methods of message passing are available, which are either reliable or best effort.

When inquiring about the underlying technology, CSR stated that it would take about 12-18 months until their protocol would be made available to consumers (cf. Appendix 2).

The first analysis of CSRmesh only provided some assumptions about the performance. (AIRcable.net 2014) states that each device in the mesh must constantly listen for new messages. If devices only listen sporadically, messages are lost or the latency increases. Using this technology is therefore only practical for devices that are connected to power sources or high capacity batteries. Because all traffic is sent on the advertising channels, these will get polluted with messages when the device density increases. These effects are discussed in (Ammari 2013, §§ 6.1). The next section presents a study that confirms these thoughts based on an actual implementation.

5.2.4.2 nRF51 BLE Rebroadcasting Mesh

The work conducted by (Trond Einar Snekvik 2014) analyses a custom rebroadcasting mesh solution[6] based on nRF51 hardware and draws several interesting conclusions.

On the positive side, since a rebroadcasting mesh does not depend on a specific topology, it can support dynamic nodes without any impact. Furthermore, if the node density is high enough, failing nodes will also not affect the performance of the network. Latency can be kept low because a packet will always travel the biggest possible distance to other nodes. Because there is no management overhead for discovery and routing, the network can theoretically reach an unlimited size.

However, the study states that "[t]he most notable weakness is the excessive power consumption in the radio, which makes the implementation practically unsuitable for mobile applications", noting that the provided implementation will only last a few hours on a coin cell battery. Another problem that is mentioned is channel pollution. Because of the high number of messages sent, the advertising channels are polluted with a lot of traffic, which leads to packet collisions and poor performance.

This still leaves use-cases like home automation and lightning, but excludes any use-case that depends on battery-powered nodes.

[6] https://github.com/NordicSemiconductor/nRF51-ble-bcast-mesh

5.3 Bluetooth Classic Scatternets

Bluetooth Classic and Bluetooth Low Energy differ in the way in which they support different topologies. The Bluetooth 4.0 specification did not allow the formation of scatternets by restricting each device to a single role. This made it impossible for a device to take part in more than one Piconet (explained in section 6.3.3.6) at a time. Bluetooth Classic does allow every device to assume a slave role in multiple Piconets, which is achieved by having the device constantly in standby state (Laird 2013). Bluetooth Low Energy devices on the other hand cannot listen longer than necessary and will switch off their receiver whenever possible. They can also not share a common channel. While being a slave in multiple Piconets has been allowed with the Bluetooth 4.1 specification, the scheduling is left to the implementation which is a non-trivial task and has not been implemented in the S130 SoftDevice. There is currently no known implementation that allows this. Having a concurrent slave and master role is however possible.

A survey shows that a multitude of scatternet formation algorithms exist (Mišić, Mišić 2006, § 9) and they employ a various number of techniques to efficiently form different topologies. None of the found algorithms do however deal with the limitations imposed by the current Bluetooth Low Energy stacks like the S130 with the connection limitation being the most problematic one. It is unknown at this point if a future implementation will support multiple connections as a slave but it is likely that more than three connections as master will be possible.

5.4 Conclusion

There is a lot to be learned from other technologies that should be considered when implementing a mesh with Bluetooth Low Energy. This knowledge was incorporated in the design process and some techniques are now discussed in more detail.

5.4.1 Wake-up Techniques

Long sleep times can drastically reduce energy consumption in nodes. According to (Boukerche 2009, § 2), there are multiple wake-up technologies that can be grouped as seen in **Figure 5.3**. All these have been considered in regard to Bluetooth Low Energy. Because we must stay compatible with the standard, some of these techniques will not work due to missing hardware or protocol limitations.

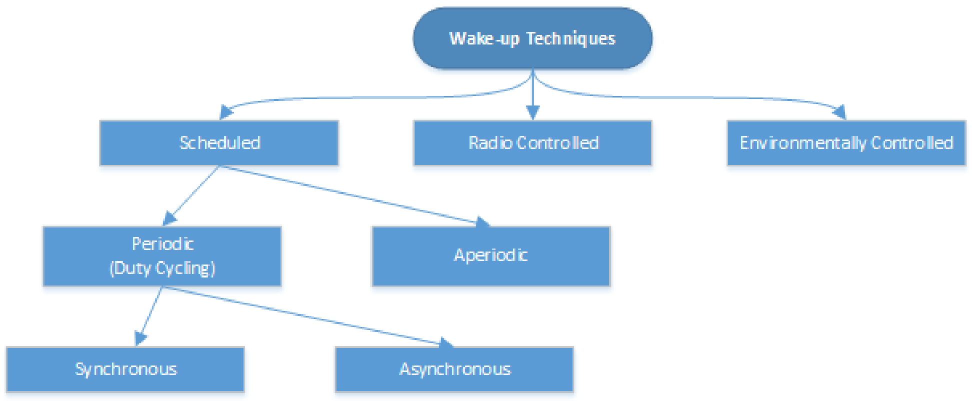


Figure 5.3: Wake-up techniques
Original source: (Boukerche 2009, § 2)

Scheduled wake-up is implemented in BLE connections. It allows two nodes to wake up **periodically** and **synchronously** to exchange information. By changing the connection parameters during a connection or by using Slave Latency, we can achieve an **aperiodic** wake-up cycle. When nodes wake up **asynchronously**, in contrast, they might need to stay awake for a long time until they receive a signal from another node. This is only beneficial for certain traffic types.

Radio controlled wake up is a technology that enables low latency and low power consumption at the same time. Radio controlled wake up uses dedicated hardware Wake-up Receivers (WuRx) that are able to listen continuously for specific peaks in the surrounding radio signal while using little power. If a wake-up signal is detected, the main radio is woken up for communication. This technology has been researched in conjunction with Dozer in glacial environments (Jelicic et al. 2013). With an increased power usage of only 10%, it was possible to reduce the latency to close to zero. BLE does not have this dedicated hardware and the 2.4 GHz band is full with interference that could cause many incorrect wake ups in busy environments, so this technique can be ruled out.

The **environmentally** controlled wake-up technique can use sensors or events from a power source such as an energy harvester. An earthquake or the first sunrays can then wake up all nodes at once from a long sleep.

In summary, there are no other known wake-up techniques apart from scheduled wake ups that are possible without additional hardware. This means that these other techniques can be ruled out from further research.

5.4.2 Node Diversity

ZigBee has three device types that possess different capabilities. A router is always in listening mode and therefore uses more power. This can be very convenient for other devices in order to save power because they can send data whenever they please. But installing a router is sometimes cumbersome and makes the setup process more complex. Additionally, it requires a power connection which is not always available. We cannot rely on the existence of a hub or certain devices with different capabilities because there are too many use-cases with different needs. Therefore, the network must be able to work when all nodes are equal. If possible, devices with more power or bigger memory should be able to work higher duty cycles and accept additional tasks in order to save resources of other network nodes.

5.4.3 Topology

The presented technologies rely on different topologies. While it is possible to aggregate packets and build a single line topology as done with LEACH (Heinzelman et al. 2000), this is not feasible for other tasks where only some devices have to send and receive data or where low latency is desired. Creating a network with multiple cluster heads as done in Wirepas on the other hand is currently impossible because of restrictions in the S130 SoftDevice. Using a tree or broadcast topology should therefore be considered.

6 Designing a BLE Mesh

6.1 Use-cases

Use-cases for a BLE mesh are countless and I can only present a few selected ones. But they serve as a good starting point for determining the requirements.

Deploying a configurable **indoor navigation** installation with battery-powered nodes is easy and can be done without any wiring, but the maintenance is not trivial because it requires monitoring the functionality and battery levels of many different nodes. This is a lot simpler if all nodes are connected and can be managed from a central place. Thereby, it is also possible to update the broadcast messages of each node with different data. Adding data can also be useful when indoor navigation should be used for other use-cases such as guiding robotic vacuum cleaners across a location and telling them where to clean.

By installing security certificates on individual nodes and by synchronizing their time with a server, it is possible to generate authenticated messages. These can even include a user's id, received from a smartphone, and could then be used for **bonus campaigns** where a user is rewarded for being at a specific place at a specific time.

Home automation works best when it can be refitted to existing devices. Power plugs, light bulbs, heating regulators, and many more can be outfitted with Bluetooth Low Energy chipsets to monitor and control them. Because a user is not always close to the appliances he wants to control, they must be connected to relay data to each other. Values from multiple sensors like thermostats can be collected at a central station and can be included in the solution.

Alarm systems consist of many sensors that must report to a central unit. Some of these sensors could also be moving around dynamically and their position should be tracked. Monitoring whether a window is open or closed is a good use-case in which the user is then able to check the current status with his phone whenever he wants. Practically, these sensors do not rely on the power grid and will not fail if there is an outage. Combined with a GSM module for example, they could broadcast their state over the internet.

Wireless sensor networks have been used for a long time to monitor big areas like forests or glaciers for changes in temperature, humidity and much more. **Monitoring** big areas like **tea plantations or farms** is equally possible as **monitoring potted plants** in a home. This is equally interesting for facility management. Objects like toilet paper dispensers can be monitored easily.

Finally, using a mesh network to provide an **off-the-grid network** in case where no other network service is available has been successfully demonstrated by the mobile application FireChat during protests in Hong Kong (The Guardian 9/29/2014).

6.2 Requirements

The above mentioned use-cases have been used to determine the requirements:

- Every node in a network must be able to operate battery-powered in the order of years.
- Nodes should be able to route traffic to other nodes or to a sink. Routing from a sink to a node should also be possible. Multiple sinks should be supported.
- The number of supported devices should be in the order of thousands with a typical scenario of dozens.
- The throughput will be very low in the order of bytes per second.
- Latency is mostly not an issue but should be kept lower than one second per hop.
- The network must be self-healing and should continue operation in case of node failure.
- The implementation shall be compatible to the current Bluetooth standard.
- The implementation should require little amounts of flash and RAM memory.
- Current smartphones should be able to access the mesh by sending and receiving data.
- Nodes must be able to report their current status. (battery, connections, …)
- The setup process should be very easy with little configuration needed.
- It should be possible to protect the network by encrypting all messages.
- While most nodes will be stationary, there should be support for moving nodes.

6.3 Advertising vs. Connections vs. Custom Protocol

This leaves us with three options for implementing the mesh. We could either use a custom protocol to implement the mesh communication with the Timeslot API, we could rely on Bluetooth LE connections or use BLE advertising messages. A combination of these is possible as well.

6.3.1 Using a Custom Protocol with the Timeslot API

While implementing a custom protocol would have fewer restrictions, it has been ruled out because of the high implementation effort. Using the Timeslot API requires very low-level programming. The protocol will then be limited by the timing restrictions that the API imposes. Not building on top of the BLE standard is in contrast to the original idea and custom protocols are already available. One example is Wirepas that has multiple years of research background.

6.3.2 Using BLE Advertising Messages

6.3.2.1 Flood Mesh

Advertising messages can be used to implement a flood mesh. Trying to send messages through a BLE flood mesh has already been evaluated in the work of (Trond Einar Snekvik 2014) and did not provide the desired results. Message collision and battery-depletion due to long scan-times are unavoidable in flood meshes.

6.3.2.2 Scheduling Advertising Messages

It is possible to avoid the complexity of a custom protocol and use scheduled advertising packets that are sent at known times. But this means, that we restrict our protocol to a subset of the available frequency spectrum.

Initial tests with a sniffer showed that scheduling advertising messages is not possible with the current standard because of some problems. (Bluetooth 4.2, Part B, Vol. 6, 4.4.2.2) states that "the advertising events are perturbed in time using the **advDelay**" which is a "pseudo-random value with a range of 0 ms to 10 ms generated by the Link Layer for each advertising event". Because this number is generated by using the hardware-based random number generator in the nRF51, it is not possible to distribute a seed that connection partners could use to synchronize their advertising and scanning efforts. Additionally, distribution of the advertising packets within the advertising interval is managed by the link layer implementation. It is only regulated that the start of two consecutive packets shall be less than or equal to 10ms (Bluetooth 4.2, Part B, 4.4.2.3). This makes scheduling very complicated.

Many BLE stacks do not allow the implementation to select the advertising channels that are used to transmit a message. And the BLE specification mandates that the selected channels must be used cyclically. With the nRF51 API, it is also not possible to access the AES-CCM encryption mode because access is blocked by the SoftDevice. Only the basic AES-ECB mode is available and a software implementation must be used to provide a secure encryption for advertising messages.

While advertising messages allow devices to use a common channel for broadcast messages, they do not solve the complexity of synchronizing devices. Using advertising messages is in many terms similar to implementing a custom protocol. It has therefore not been chosen as an appropriate way to implement a mesh protocol.

6.3.3 Using BLE Connections

Using BLE connections alleviates a lot of work needed for synchronizing devices. With the S130 it is possible for a device to assume a master and slave role simultaneously. This feature had been introduced with the Bluetooth Standard 4.1. The current S130 alpha build supports being master in a total of three connections and acting as slave in up to one connection. I will refer to these connections as outgoing or incoming connections which can be determined by the direction of the initiation packet. A master is always the connection initiator and uses an outgoing connection to connect to a slave which receives the connection request. Just as a reminder, this does not have any implications on the data transmission direction.

BLE connections can be used to build a scatternet configuration that spans all reachable devices. Furthermore, encrypting these connections can be easily done, because this feature is available in the BLE standard and therefore the S130. Some other important aspects of connections are now discussed in the next sections.

6.3.3.1 Connection Interval and Slave Latency

Bluetooth Low Energy connections use a scheduled wake-up technique that is coordinated between one master and multiple slaves. Both devices, master and slave, periodically wake up at an agreed interval for the time that is needed to transmit a packet. At each timeslot, the slave must resynchronize to the received signal from the master in order to determine the timeslot for the next connection event. It must also account for both the master's and its own clock inaccuracies by applying Window Widening as explained in section 6.3.3.2.

With the use of Slave Latency, the slave can control its wake-up times and can decide to omit some connection events. It will skip these events and will not listen to the master or send any information.

While the slave is able to offer his Preferred Connection Parameters to the master, these must not be taken into account by the master which means that the slave must strictly follow the settings dictated by the master in order to maintain a connection.

6.3.3.2 Window Widening and Drift

A slave must account for the drift of his and his master's sleep clock drift by increasing the time it listens for packets. The longer it does not receive a packet from its master, the more it must widen its window. With the maximum connection interval of 4 seconds, a slave must listen for an additional 4 ms at every connection event if we assume a drift of ± 500 ppm, the maximum drift that is allowed for the sleep clock, for both devices. The accuracy of the master's sleep clock is included in the connection initiation packet and should be used to calculate the Window Widening. Slave Latency will also result in a higher Window Widening.

6.3.3.3 Overlapping Connection Events

Because of the side effects from Slave Latency, drift, and channel interference it is possible that different Connection Events overlap. If this happens, the Central must reschedule the Connection Events with its Peripherals. Otherwise it might risk collisions that result in dropped connections. A Link Layer Connection Update Request (Bluetooth 4.2, Vol 6, Part B, 2.4.2.1) allows to define a time offset for all future Connection Events.

Because the S130 can act as slave and master at the same time, it may receive a Connection Update Request from its master which can collide with its own schedule. It is then possible that the offset is cascaded throughout the network until connection events do not overlap anymore. Intelligent scheduling of devices solely relies on the implementation in the SoftDevice and is not trivial. The release notes for the S130 1.0.0-alpha state, that the Link Layer may under certain circumstances not be able to maintain connections (Nordic Semiconductor ASA 2015a). This issue is hopefully resolved in future versions.

As an example, we do now assume a Connection Interval of one second and a Central that is connected to four devices. All five devices have a maximum drift of ±250 ppm. If they each send one packet with 20 bytes at their respective connection events (adding up to a total of 8 packets per second), this results in a combined 1.28 ms of data transmission per second and a listening time of 1.4 ms for each.

In this example, the collision probability is still very low but will of course increase with lower Connection Intervals, more Slave Latency or higher interference. It can be said that typical duty cycles for BLE applications are in the order of 0-3% which means that there should generally be a low probability for collisions if the internal scheduling is done right.

6.3.3.4 Energy Consumption

BLE devices consume most of their power while actively sending or listening. During sleep times, the power usage is very low because only the sleep clock and some RAM regions have to be kept active. Depending on the type of sleep clock that is used, connections will have to cope with a high drift, with all the consequences that have been discussed previously. It should therefore be evaluated if the selection of a more accurate clock could provide better results.

There are typically two different hardware clocks in a BLE device. For the nRF51 chipset, the High Frequency Clock (HFCLK) runs at 16 MHz and uses a few hundred µA. It has a small drift of only ±50 ppm. The Low Frequency Clock (LFCLK) is clocked at 32 KHz and uses less than one µA with a maximum drift of ±250 ppm. It can therefore only be used during sleep times because BLE requires a higher accuracy for radio activity. Using the High Frequency Clock during sleep times is not possible because of its excessive power usage. But having two connected devices with a drift of ±250 ppm means that these can drift up to 500 µs in either direction from each other if they have not had contact for one second. This is why an external 32 KHz crystal like the FC-135[7] should be used because it provides a much better accuracy of ±20 ppm at low currents. Using a DC / DC converter is also advised to improve the current consumption of the chipset.

In general, the energy consumption for BLE connections is very low because connected devices are duty-cycled. Sending packets through an active connection does not consume much energy because the time required to transmit a packet is very short with only 1 µs per bit. Connection setup on the other hand can cause a lot of battery drain because it involves sending multiple advertising packets and it requires the other device to be in the Initiation state. If there is no possibility of synchronizing these two devices they can only meet by chance. In consequence, reconnections should be avoided and routing a packet through multiple hops can better conserve energy than building a faster route. This should therefore be the preferred way of sending data.

6.3.3.5 Multiplexing Channel Access

Bluetooth LE connections do already provide a method that helps to prevent packet collisions. A Central uses time multiplexing to schedule its slaves and frequency multiplexing on 37 channels in the form of Adaptive Frequency Hopping to guard against interference from other devices. Unfortunately, it is not possible to have a common timeslot at which all slaves devices listen to the master, which means that a broadcast packet has to be sent to each slave individually.

According to (Wirepas Oy 2014), interference is local by nature. We should therefore distribute channel blacklists between adjacent nodes and configure them to use non-overlapping channel maps, which reduces packet collisions. Unfortunately, the current SoftDevice API does not have a method for manually setting the channel map.

After a connection has been set up, the transmission power should be decreased to further reduce interference in dense networks and to conserve energy. In contrast to other protocols, the BLE Controller does not dynamically set the TX power. This has to be done by the application. But setting an independent transmit power for each connection link is not yet possible with the S130 API as well.

[7] http://www5.epsondevice.com/en/quartz/product/crystal/tuning_fork/fc135.html

6.3.3.6 Topology and Routing

With Bluetooth 4.0 it was not possible to form arbitrary topologies with BLE connections. A device was only allowed to either act as a Central or as a Peripheral. A Central would then form a star topology with a number of slaves, which is called a Piconet (**Figure 6.1**). The number of slaves is not limited by the specification but is limited in most BLE stack implementations because each Peripheral adds to the memory usage and processing time. The number is also restricted by the available time because every Peripheral needs its own timeslot. The S120 SoftDevice for example does set an upper limit of 8 connected devices and the S110 Peripheral only allows one connection at a time.

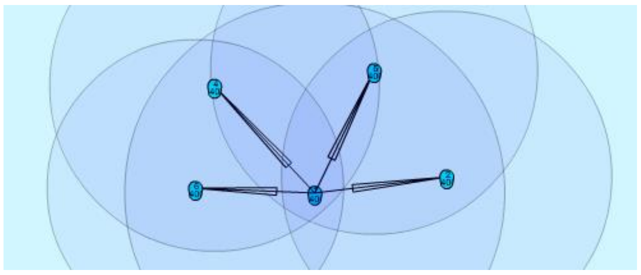


Figure 6.1: Piconet or star-topology
Source: Original work

With the Bluetooth 4.1 specification and later, a device is now allowed to assume both roles simultaneously. Additionally, a device can now be a slave in more than one Piconet at the same time (Bluetooth 4.2, Vol. 1, Part A, 4.1.2). This does - in theory - allow many different topologies. But the S130 only implements a subset of these features. In early versions, there have been some restrictions regarding the roles that it can assume concurrently. The latest 1.0 version has solved most of these issues but does only allow a 3:1 connection ratio which means that it can be master of up to three connections and can be slave in one connection. It is possible that future versions will allow more than three connections as master (Nordic Devzone 2015c).

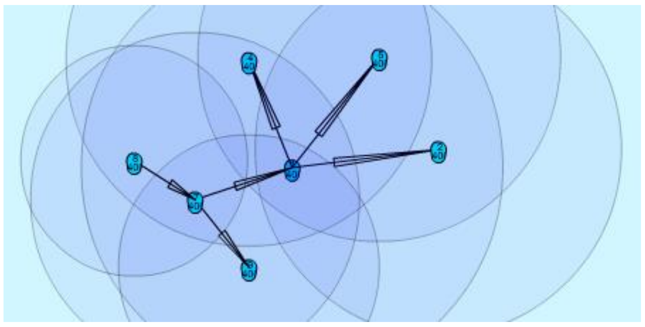


Figure 6.2: New topology possibility with Bluetooth 4.1
Source: Original work

Being a slave in more than one connection is believed to be very difficult because a slave must always follow the connection parameters given by its master. Having two masters at the same time will result in overlapping connection events with no possibility to react.

Having only one connection as a slave is a severe limitation when forming the network topology but makes the subsequent routing a lot easier. The 3:1 connection constraint does only allow for a spanning tree topology with the ability to form up to one cycle in the network. Routing in a tree is very easy and cannot lead to infinite loops in the message relaying method.

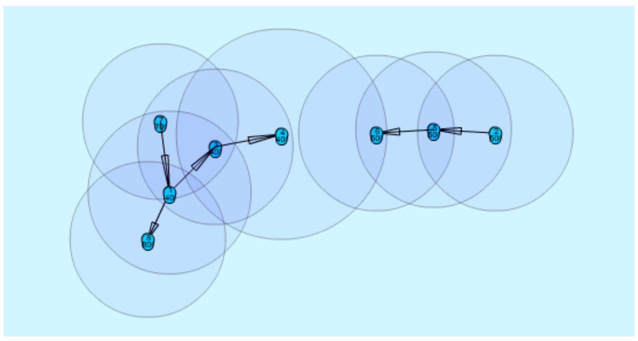


Figure 6.3: Impossible topology with the S130
Source: Original work

The biggest problem that we need to solve originates from the connection ratio imposed by the S130; it is depicted in **Figure 6.3**. Two devices that already have used up their incoming connection will not be able to connect to each other anymore.

6.3.3.7 Summary

Bluetooth Low Energy connections can use up to 37 data channels, which gives them an advantage over advertising. The synchronization and scheduling of connections is already integrated in the standard, just as encryption is. This reduces the implementation effort. The biggest problem that needs to be solved is the topology setup. Using the S130-alpha SoftDevice can pose some problems during implementation.

6.4 Discovery and Route Establishment

Setting up the network involves different phases:

1. In the **discovery phase**, nodes must learn about their neighbours, to which they can connect.
2. During **route establishment**, they must then form a topology that spans all nodes that want to take part in the network.
3. In the final **routing phase**, the data packets must be routed between all nodes, including sinks.

Theses phases occur multiple times and also simultaneously when the network is healed in case of interference or node loss. New nodes must be able to enter the network at any time and routes have to change if interference or node movement happen.

6.4.1 Deterministic Approach

The Wirepas Pino protocol uses only local decisions that are not guarded by a central instance. This is probably a good choice because there is no easy and energy conserving way to exchange data between distant nodes when forming the mesh. Using a central coordinator would pose increasing problems as the networks grows. Another interesting feature is the protocol's neighbour's neighbour discovery. Each node transmits its known neighbours in order to save some of the scanning overhead in every node.

After following these initial thoughts, a lot of research went into finding a deterministic solution to the network setup procedure. After not being successful for some time, it was abandoned and then followed by a different approach that is explained in section 6.4.2.

6.4.1.1 Initial Thoughts

We assume that it is possible to find a set of rules that enable a deterministic route establishment without the need for a central coordinator. The only information that is available to any node is of local nature. By using neighbour's neighbour information, we can broaden a node's view of its surroundings. We do further assume that most nodes are not moving and remain static for a long time. Simplified circular send ranges are used for evaluating possible scenarios which is different from the radio patterns in a real-world scenario. But it should provide good enough results for the purpose of finding an algorithm.

6.4.1.2 Problematic Scenarios

Under some circumstances it is not possible to find a connection graph that spans all nodes with the connection limitation imposed by the S130. **Figure 6.4** represent nodes as small blue circles together with solid walls in orange that cannot be penetrated by radio signals.

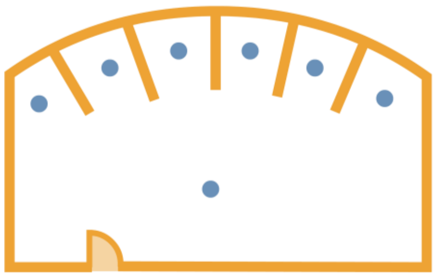


Figure 6.4: Radio signals are obstructed by walls
Source: Original work

This configuration of nodes cannot be connected because all nodes can only reach the node in the centre, which can only manage a total of four connections. Radio signals between the outer nodes are obstructed by the walls. This scenario is unlikely but can be solved by placing an additional node in the centre.

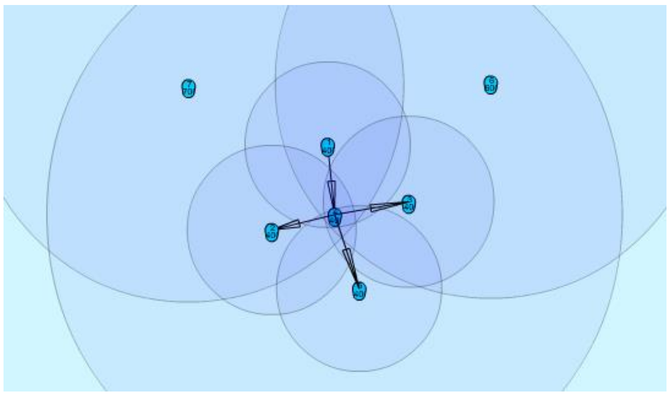


Figure 6.5: Connection problems due to radio range
Source: Original work

Figure 6.5 represents another scenario where some nodes cannot be connected to the network because they have widely different send ranges, represented by the circles around the nodes. In order to maintain a connection, two nodes must be within the send range of each other. The solution is again similar to the previous problem. In reality, most networks will be dense enough to have multiple connection partners for each node. Additionally, the send range of all nodes is expected to be of comparable size.

6.4.1.3 The Idea

Only solvable graphs have been considered and different node formations were evaluated in order to find a set of rules that can be used to enable a deterministic formation of a graph. A simulation was implemented to easily evaluate these formations and rules. **Figure 6.6** is a screenshot of this simulation with a very basic ruleset implemented. Green lines depict possible links while an arrow represents the direction of an actual connection.

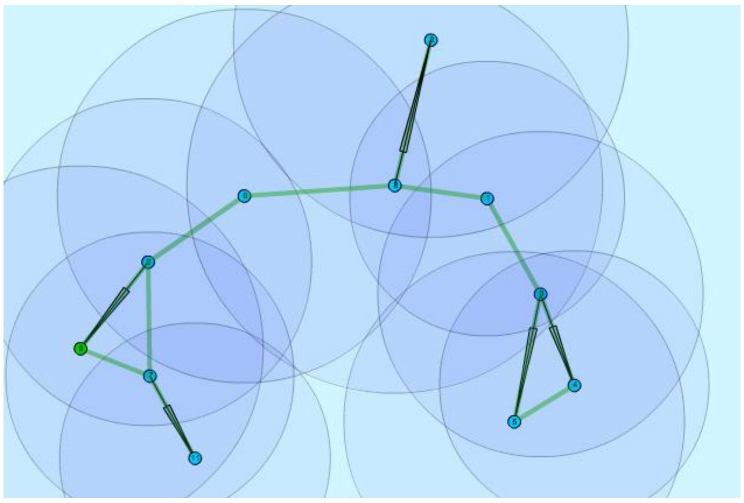


Figure 6.6: Deterministic solution experiment
Source: Simulation

The rules that were used for the simulation in **Figure 6.6** are the following:

If a node is only able to see one other node, it can accept an incoming connection from this node. Because incoming connections are the limiting factor, this does not restrict the resulting network setup. There is no second connection partner that could want to establish an outgoing connection to this node. This rule was used to set up the connection with node 11.

If the surrounding nodes can only see a subset of the own observation, we can connect to these as long as we have connections available. In the example:

- node 5 observes nodes {0, 4}
- node 4 observes nodes {0, 5}
- node 0 observes nodes {0, 1, 5}

Node 0 is also in knowledge of the observations from each of these nodes and can therefore make decisions about the connections it wants to build because it possesses the biggest knowledge. It can therefore control the connections of the other two nodes.

However, this poses some problems that are difficult to solve:

If we assume that a number of nodes are arranged in a ring, with each node only able to reach its two next neighbours, there is no way to find a deterministic starting point for the algorithm. All nodes possess the same kind of knowledge. Using other factors such as the node ids is not possible, because it is unknown which nodes take part in the network. One solution is to start connecting after a random timeout has been triggered in any of the nodes. But this can lead to unsolvable graphs and an error handling routine must be designed that handles graphs that have been improperly connected.

There were also practical problems that limited the original idea of neighbour's neighbour discovery. The advertising packets are size limited and cannot contain the ids of all neighbouring nodes in a dense network. Splitting the information over several packets would have meant an increased protocol complexity and implementation effort.

Furthermore, this algorithm does not work if nodes are switched on at different times and it does not provide a good method for handling moving nodes. It has therefore been abandoned in search of another algorithm. Because distributing neighbour's neighbour information was found to be too complex, it was decided that the final algorithm should only possess knowledge about directly connectable nodes.

6.4.2 Neighbour-only Clustering: FruityMesh

The final algorithm is presented here and offers a solution for building a graph which is only based on the knowledge of surrounding nodes. It enables automatic meshing that is functional and constructs a spanning tree with all reachable nodes. Loop building is effectively prevented. It is self-healing in case of node loss or short-lived node failure and can be optimized for a number of use-cases by customizing a cluster scoring function. This results in different spanning tree configurations. Nodes can be switched on at different times and even after the route setup has finished.

6.4.2.1 The Idea

Every node is always part of a cluster with a unique **Cluster Id** which is generated by the founding member. A single node with no connections represents a cluster with the size of one. Only nodes that are part of different clusters will connect to each other and exchange their cluster information. This information is evaluated in the **Cluster Score** function. The node that is part of the bigger cluster can decide which connections it wants to make and will convert nodes from smaller clusters to its own cluster. The increased cluster size must then be reported back to the rest of the cluster. Size information will be passed through the existing connections which is an energy efficient way of distributing information. If nodes disconnect from a cluster either voluntary or because of timeouts, the size of the cluster must be decreased according to the number of nodes that were lost. This requires each node to save the quantity of connected nodes for each active connection.

6.4.2.2 Node Initialisation

Before a node can join a network, it must be assigned a **Node Id** and a **Network Id**. In case of an encrypted network, it must also possess the network encryption key that will be shared among all nodes in the network. The Node Ids do not have to follow a scheme but must be unique for that network. The Network Id allows different networks to coexist in the same physical space. These values are configured in the enrolment process. After booting, a node generates a new **Cluster Id** based on its Node Id and the total number of connection losses. Connection losses are saved in persistent memory in an integer that may wrap. The initial **Cluster Size** of a node must be set to 1. A value called **Connected Cluster Size** must be saved for each of the possible connections (1 outgoing, 3 incoming) and is initialized with 0. **Figure 6.7** depicts a node that is directly connected with two other nodes and gives an overview of the stored values.

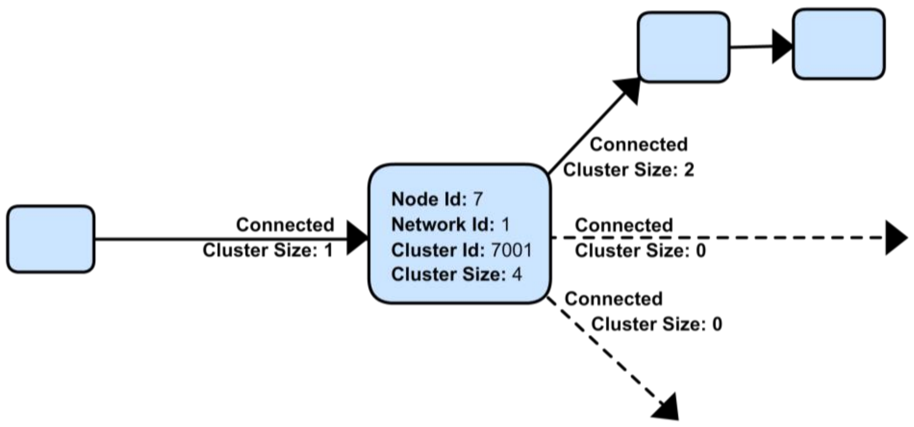

Node Id: 7
Network Id: 1
Cluster Id: 7001
Cluster Size: 4
Connected Cluster Size: 1
Connected Cluster Size: 2
Connected Cluster Size: 0
Connected Cluster Size: 0

Figure 6.7: Example of the stored values in a node
Source: Original work

6.4.2.3 Discovery

During the discovery phase, a node will constantly broadcast its Node Id, Cluster Id and its Cluster Size in an advertising packet. This packet has been called the **JOIN_ME** packet and can also include further attributes that other nodes use to calculate the **Cluster Score**. The current implementation of the simulation and the prototype broadcast the Cluster Size and the number of free incoming and outgoing connections (freeIn, freeOut). The advertising packet must be connectable if the incoming connection is still free, otherwise it is non-connectable.

Between sending these advertising packets, a node must scan on all advertising channels for packets of surrounding nodes. These are collected in a buffer and only the most recent packet from any node is saved with a timestamp of the reception time.

6.4.2.4 Route Setup

After a predefined amount of seconds, a node calculates its best connection partner based on the collected JOIN_ME packets. At first, it looks for nodes that have a freeIn connection and tries to connect as a master. Each packet is evaluated with the Cluster Score function, which returns a connection score. This score is zero if the node belongs to the same cluster or if the other cluster is bigger. (If both are similar in size, the Cluster Id is used to determine the winner). After processing all packets, the algorithm tries to connect to the highest ranked node. Therefore, it needs to listen for another advertising packet which it will answer with a connection request. In case of success, the two nodes must then perform a handshake.

Before the handshake is finished, the newly connected node cannot participate in the network and must not send any packets except those belonging to the handshake. Both nodes start the handshake by sending a **CLUSTER_WELCOME** packet which includes the latest information about their Cluster Size and Cluster Id because the available information might already be outdated. This can result in a few different cases:

If they already belong to the same cluster, the must disconnect because they are already linked through other nodes. This is done to ensure that there is not more than one link between nodes of the same cluster.

If they have different Cluster Ids, one of the nodes will have a smaller Cluster Size. This node must then disconnect all other connections and must join the bigger cluster. It saves its new Cluster Id and the Cluster Size which it increments by one. Afterwards, it sends a **CLUSTER_ACK** packet to its connection partner that confirms its entrance to the cluster. This marks the end of the handshake. After the partner has received this packet it will increment the Cluster Size and it will send a **CLUSTER_UPDATE** message to other nodes that informs them of the size change. The packet contains either a size increment or decrement and not the absolute size of the cluster. Sending the absolute value would not work because there is no guarantee that all nodes hold the same Cluster Size at one time.

Cluster Size decrements will be sent as soon as a node encounters a disconnection for any reason. Because it knows the number of nodes that were connected through this connection, it can send the appropriate decrement message. This even works in a situation where the other node fails to respond and a timeout is generated.

After disconnecting from nodes of the same cluster, there are two separated networks with the same Cluster Id. Because that would result in the forming of islands (disjoint clusters), the smaller cluster has to distribute a new Cluster Id. The new Cluster Id is sent with the same CLUSTER_UPDATE packet that contains the size decrement after a connection was lost. The new Cluster Id is based on the Node Id that just lost the connection. To generate the new Cluster Id, the connection loss counter is merged with the Node Id to ensure that the same Cluster Id is not in use twice which would result in two disjoint clusters with the same id.

6.4.2.5 Route Maintenance

Nodes that belong to big clusters can always decide which connections are made and will dissolve smaller clusters. If adjacent devices are switched on one by one, they can instantly connect to the mesh. But if two different cluster meet at one point, the bigger cluster will force the small cluster to reform until there is only one cluster left.

After the network has been formed, the JOIN_ME packets will all advertise the same Cluster Id. This will gradually cause all nodes to switch to a lower discovery mode when no changes are detected for a long time. They will then scan the network less frequently and send fewer advertising messages. It is possible to switch off discovery entirely after all known nodes have connected to the network but this will prevent any new nodes from joining the network. If a node is disconnected, it will be able to join again because its previous connection partner will notice the disconnection and will then switch on discovery.

The network will always try to reconnect to disconnected nodes because they will have a different Cluster Id after the disconnection. Alternative routes are formed as well which means that the network is self-healing.

6.4.2.6 Routing

Routing in the spanning tree of the network has not been a major focus of this thesis because of the variety of solutions that are already available. Some ideas and thoughts are provided for a number of different types of messages.

Broadcast messages do not require any form of routing. A broadcast message can originate from either a sink or from any node in the network. It is then retransmitted through every connection except the connection from where it was received.

Node to sink messages are considered to be another common message type. If we assume that the number of sinks in a network is a low number then each node can hold a simple routing table that contains the Node Id of each sink and the appropriate connection with which they can be reached. This information must be broadcasted by a sink when it initially joins the mesh and is propagated with the CLUSTER_UPDATE messages. When coupling this information with a hop counter, it is possible to route information to the closest sink when required.

One can also use a simpler implementation where each connection is only denoted with a Boolean that specifies whether any sink is reachable through this link or not.

Node to Node messages are the most complex scenario because a node may not be able to hold a full routing table with its limited memory resources. In this case, distributed routing tables, data centric routing (Ammari 2013, p. 215) or other techniques must be considered for optimization. Otherwise, messages will need to be broadcasted.

Sink to Node messages are similar to the previous category. In many scenarios, a sink will be connected to another network or to a device with more resources in terms of processing power and memory. This device can hold a number of network quality measurements and an inventory of connected nodes. If each node provides its connection partners it is then possible to build a full routing tree and use this information for more intelligent routing e.g. by adding routing information to a packet.

Some use-cases will also require to separate the network in a number of groups. If the number of groups is low enough to be kept in memory, each node may employ a routing table for routing to different groups.

6.4.2.7 Evaluation

A simulation has been implemented to evaluate the proposed algorithm. A number of improvements have been incorporated to tweak the algorithm's performance based on this simulation. In its final form, the algorithm produced a connected mesh network in all cases in a number of simulations. Routing has not been implemented and only broadcast messages are supported. Some of the simulation results are shown in the next chapter.

7 Simulation

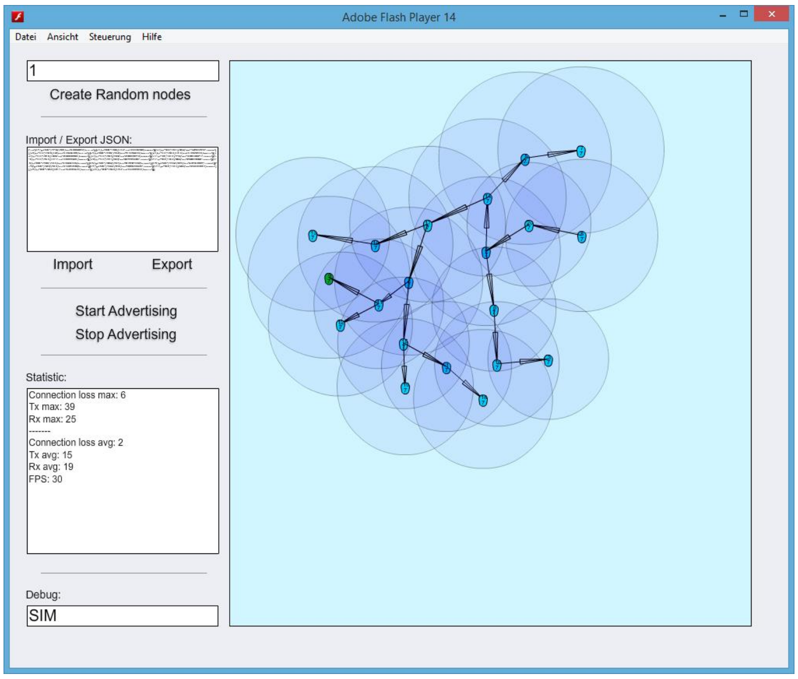
7.1 Implementation

There are a number of network simulators available, but complexity and limitations did not justify their usage over a custom implementation.

The final simulation provides a graphical user interface with which one can add nodes either manually or at random. It is frame-based and does not pay respect to timings, channel interference or collisions. It can therefore not show all side effects of a real-world scenario which is why a prototype was built as well. But when the number of frames is measured until all nodes have connected to a graph it can provide an estimate about how many connection events must pass until the final mesh is formed.

An explanation of the implementation details is out of scope and the reader is advised to look at the thoroughly documented source code instead.

7.2 Usage



```
Connection loss max: 6
Tx max: 39
Rx max: 25
-------
Connection loss avg: 2
Tx avg: 15
Rx avg: 19
FPS: 30
```

Figure 7.1: User interface of the FruityMesh simulation
Source: Application screenshot

After creating some nodes, either with the create button or by double clicking on the canvas, they can be put into advertising mode with a press on the start button. They will then send and receive advertising packets from other nodes based on their respective send ranges. The FruityMesh implementation will then build connections until the final graph is built.

Moving nodes is possible by dragging them and the send range can be customized with the mouse wheel. While hovering, the nodes provide a popup box with their status information. The final setup can then be exported and imported in JSON format by copying and pasting from the dedicated text field. Some of the status information is displayed in the GUI while other information is routed to the console output. The debug text field can be used to enter a series of comma separated commands that the simulator will follow. *SIM* will start the simulation mode and can be used in combination with *RESTART* to produce a series of measurements that are routed to an attached debugger. Some actions require a keyboard modifier to be pressed when clicking a node. Pressing *Strg* will send a broadcast message through the network. Pressing *Alt* will delete a node and pressing *Shift* allows individual nodes to start discovery.

7.3 Results

A number of different node setups have been evaluated and the simulation results have been plotted. All nodes were switched on at the same time.

The simulation shows that most networks look random at the beginning but once a bigger connected cluster has been established, it will begin to dissolve smaller clusters and will absorb them as seen in **Figure 7.2**, where the final cluster starts forming from the lower right.

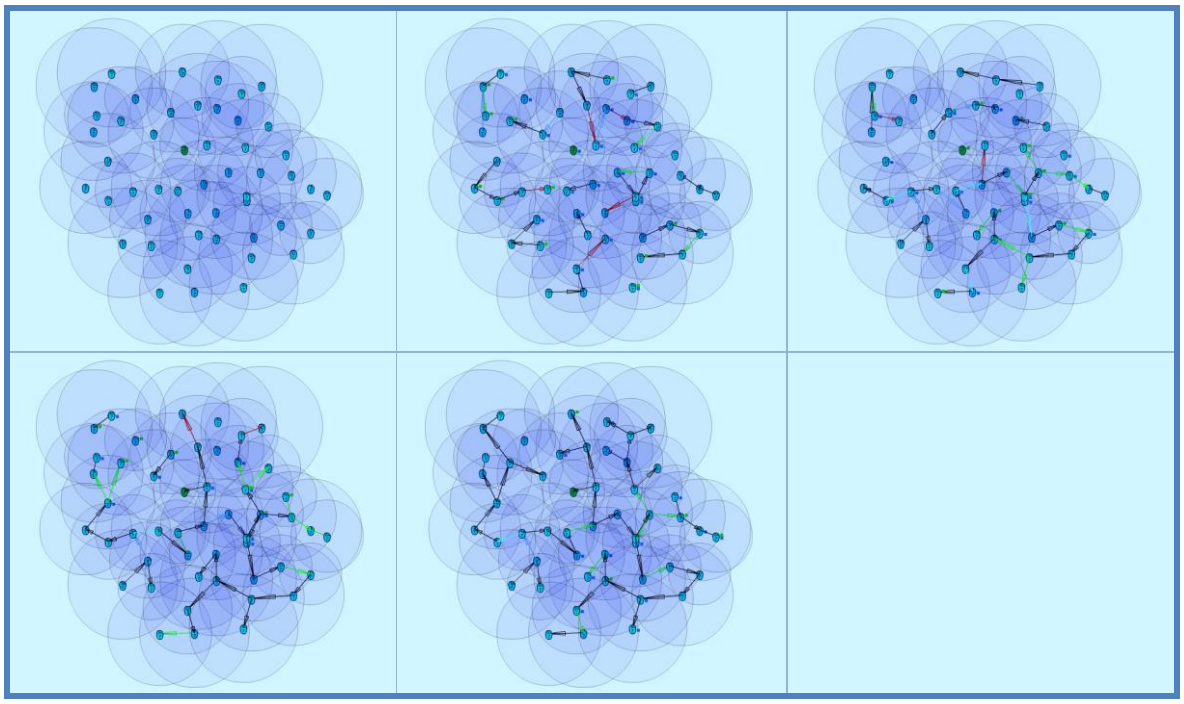


Figure 7.2: Captures of the clustering phase (top left to bottom right)
Source: Simulation

It is visible that smaller clusters dissolve and must restructure after connection with a bigger cluster. This could be avoided in some cases. Further research must show whether a new Cluster Id can be distributed in a way that leaves both clusters intact and joins them together.

7.3.1 Sparse and Dense Network Configuration

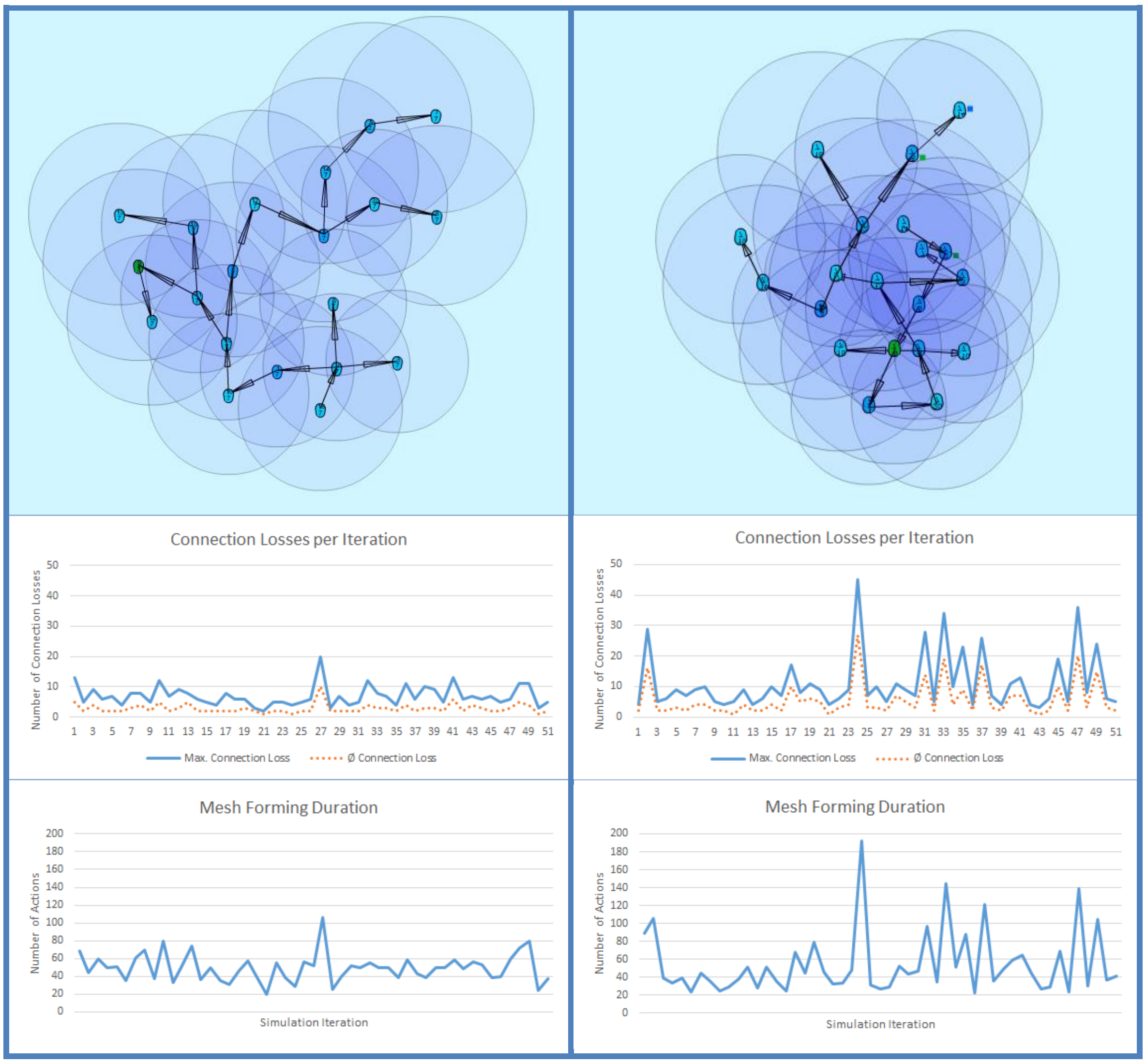


Figure 7.3: Comparing a sparse (left) and a dense (right) 20-node setup
Source: Simulation

A direct comparison between a sparse and a denser node setup shows that the connection losses and the average connection time are higher when each node can choose between many connection partners. This seems counter intuitive at first because more connection partners should result in a faster network setup. But it is visible that clusters tend to reconnect and dissolve often. The dense setup also shows that nodes do not connect to their nearest neighbours because the RSSI has not been used in the Cluster Score function.

Choosing better parameters should therefore be an important topic of future research as well.

7.3.2 High Number of Nodes

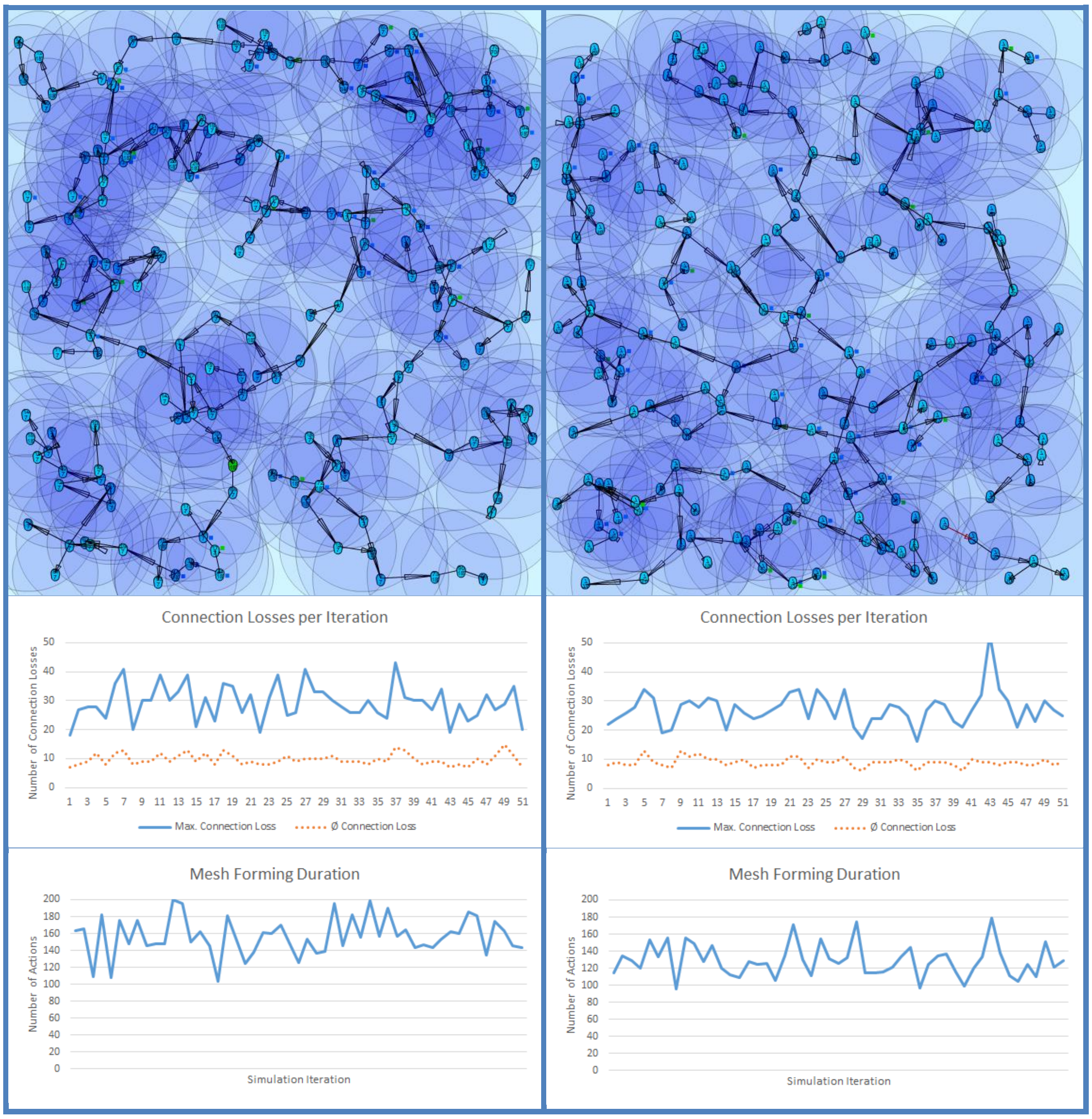


Figure 7.4: Comparing two different setups with 200 nodes
Source: Simulation

The time it takes to form the network increases approximately linearly with the number of nodes involved while the number of average connection losses will top out at about 10. (The maximum number of nodes that were simulated is 400.) The number of sent CLUSTER_UPDATE packets does also increase linearly with the size of the network.

This can possibly be improved by buffering and sending less packets if they are only used to report an increased cluster size.

7.3.3 Self-Healing

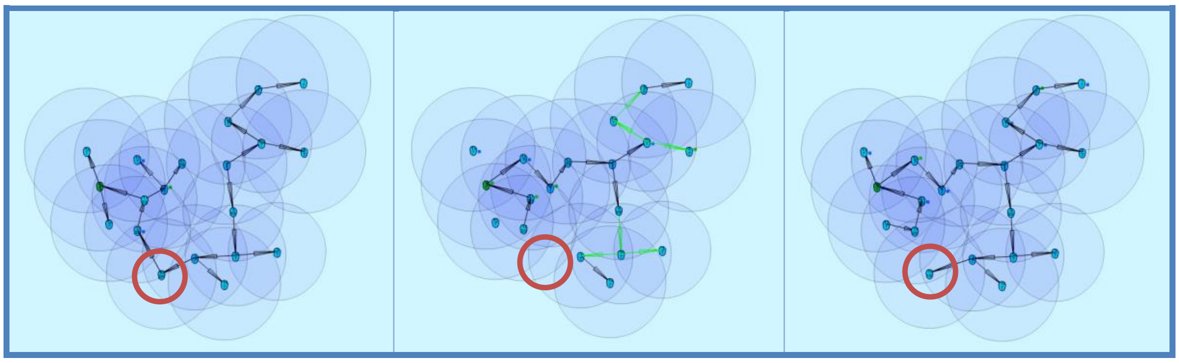
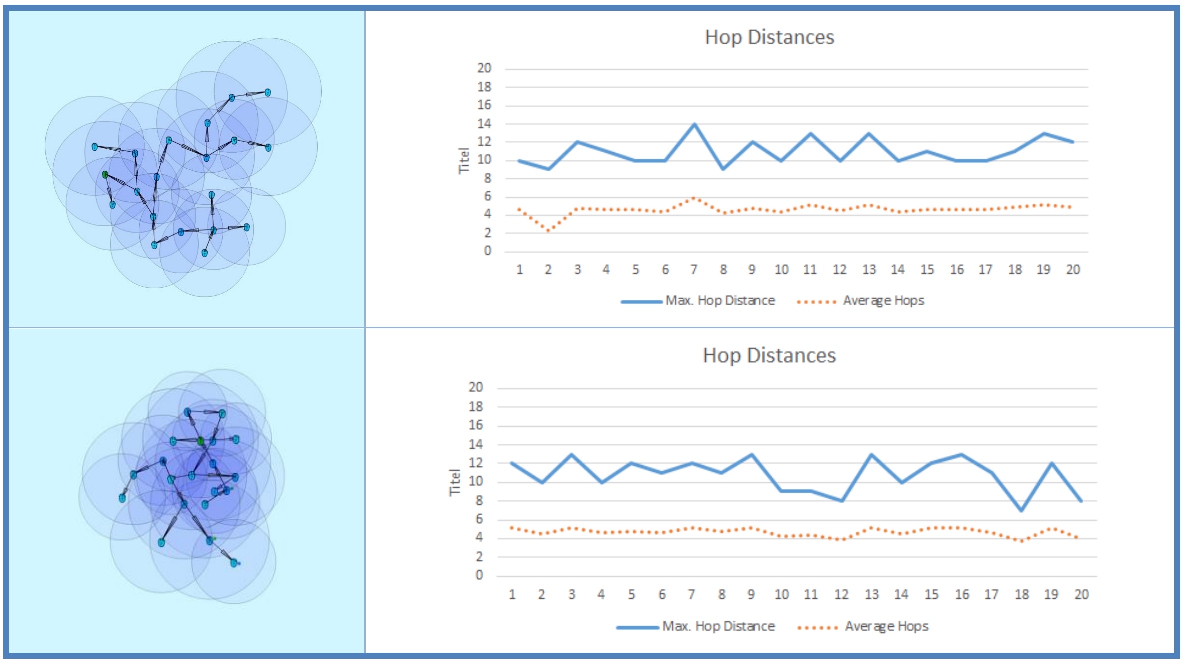


Figure 7.5: Self-healing process (left to right)
Source: Simulation

The algorithm provides self-healing capabilities. Once a node is removed (seen in the second picture of **Figure 7.5**), the nodes to the left will rearrange to join the cluster through a different route. If this previously removed node is switched on again it will be reconnected to the cluster, but only as a leaf node and not as an integral part of the network. This results in separation of failing nodes from the core of the network and therefore enhances the stability during its lifetime. Broadcasting a connection loss metric with the JOIN_ME packets could therefore allow the algorithm to form more stable networks by rating nodes based on their connection stability.

7.3.4 Hop Distances

The maximum number of hops was measured for two meshes with 20 nodes and one mesh with 200 nodes. At the same time, the average number of hops, which a packet must travel between any two random nodes, has been calculated.



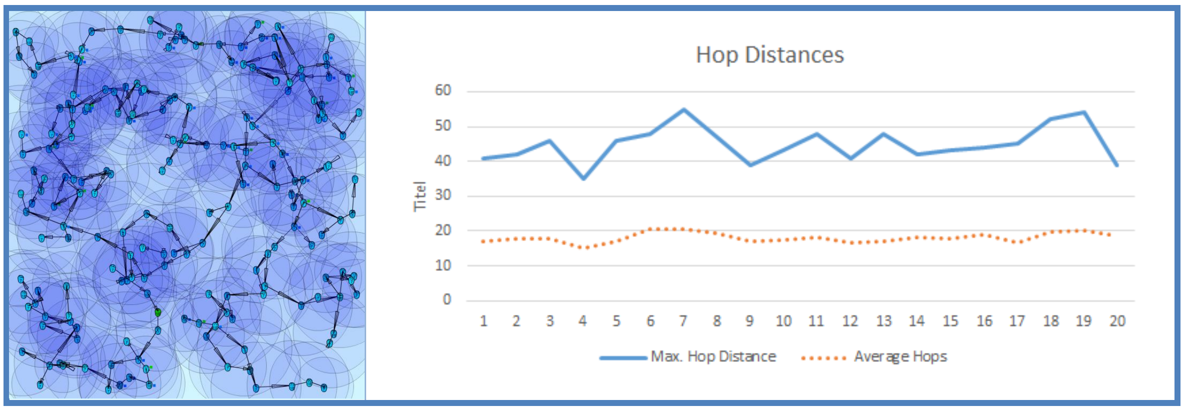


Figure 7.6: Hop distances for three different meshes
Source: Simulation

The average hop distance does not vary significantly between different simulation runs. Furthermore, it can be said that both a dense and a sparse setup produce equal results in terms of hop distance with the current cluster score function.

In another simulation, not shown here, the number of outgoing connections was increased from three to seven but the results did not show any improvement in the number of hops or in the network formation time.

7.3.5 Slave Connection Procedure

Sometimes, a node cannot connect to a smaller cluster because the only reachable node is already connected by another master and has thus used up its incoming connection. We must then tell the node to disconnect its current connection. This can have some side effects that are explained with **Figure 7.7**.

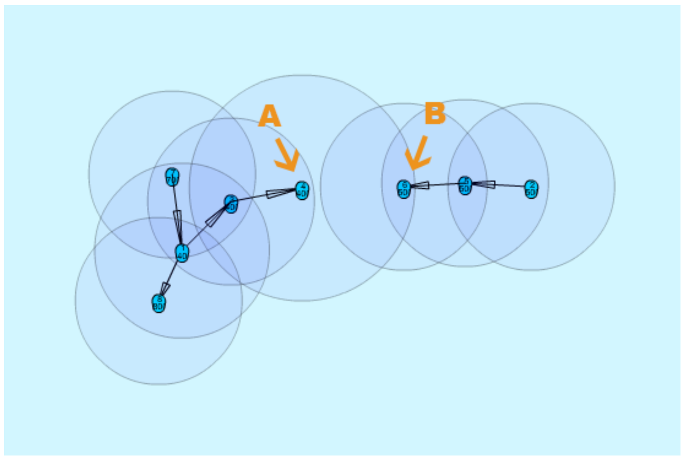


Figure 7.7: Problematic connection situation
Source: Original work

When node A broadcasts its JOIN_ME packet, node B must disconnect from its current cluster. After node B has disconnected, it will start advertising and node A will try to connect to it. This will fail because node B is not able to physically reach node A because of its limited send range. This results in permanent disconnections and battery depletion and must therefore be solved.

One solution is to add an ACK field to the JOIN_ME packet, which can contain a Node Id. Node A can set this field to the Node Id of node B, which it will only know if it already received JOIN_ME packets from node B previously.

Node B will then receive this packet and must now disconnect its connection and advertise its presence so that node A can establish a connection. This will again result in another problem because its previous cluster might try to connect again.

In the current implementation, node B will set the Node Id of node A in the ACK field of the JOIN_ME packets to signal its preferred connection partner, but it has to be evaluated whether directed advertising messages provide better results. Using the slave connection procedure takes more time because of this challenge and response scheme.

8 Prototype Implementation

8.1 Project Structure

Implementing a prototype on nRF51 hardware is not trivial and requires a good understanding of low-level programming. Developing with the S130 SoftDevice proved to be very time consuming because of its alpha state and missing documentation. The implemented prototype uses a combination of C and C++. The C language has been used to implement a lot of the fundamental functionality for advertising, scanning and setting up connections. It deals with all of the SoftDevice calls and provides an abstraction library around the SDK. The FruityMesh algorithm has then been implemented in C++ because of its advanced language features and mainly because of its ability to separate code in classes and methods. While this involved many additional problems from mixing C and C++ code, it made it easier to port the code from the simulation that had originally been developed in an object oriented language.

8.1.1 C Functions

This chapter provides a short overview of the low level C functions in order to simplify the understanding of the source code. All C functions are found in the folder `FruityMesh/src_c` and are denoted with lowercase filenames together with C++ files that are named in camel case. The corresponding include files are all found in the `FruityMesh/inc` folder.

The initialization code for the SoftDevice is found in the `initialising.c` file. Depending on the discovery state, the advertising packets need to be sent at different intervals or they have to be reconfigured with a different payload. All methods for this task can be found in `advertising.c`. SoftDevice calls for initializing and controlling the observing behaviour are wrapped in `observing.c`. Connections are handled in `connecting.c` and finally, the mesh service that is used to transmit data through BLE connections is declared in `services.c`.

The most important header files are `types.h`, which includes all custom type and packet declarations, and `config.h`, which contains all the timings and other user-configurable options.

8.1.2 C++ Functions

The mesh implementation is found in two classes:

Node.cpp

is used as a singleton and is responsible for most of the algorithm. It uses a state machine to switch between discovery, connecting, handshake and many more states. Connections are governed by a set of connection classes that are responsible for sending and buffering incoming and outgoing data packets.

Connection.cpp
holds a number of variables regarding a connection such as the connected Node Id and the Connected Cluster Size. It is able to queue multiple outgoing packets before sending them. The connection class will also relay packets but will only do so if it has finished the connection handshake.

8.2 Packet Structure

Because of the limited number of bytes that can – and should – be transmitted between devices, it is necessary to implement a binary communication protocol. All data types and packet declarations can be found in `inc/types.h`.

Some of the most common data types are:

The **nodeID** has a size of 2 bytes which allows for a theoretical limit of about 65,000 uniquely identifiable nodes per mesh network. This does also determine the maximum **clusterSize** with an identical length of 2 bytes.

The **clusterID** uses 3 bytes because it must include a nodeID and a **Connection Loss Counter**, which has a size of 1 byte.

The number of **freeIn** connections and **freeOut** connections has been given 1 byte to facilitate protocol development by not using bitmasks. But fewer bits are more appropriate for future implementations.

8.2.1 Advertising Packets

Advertising packets make use of the Manufacturer Specific Data AD Type to broadcast their mesh related data.

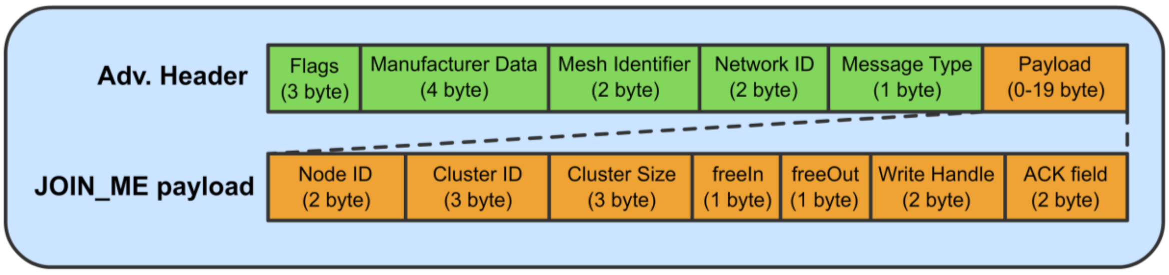


Figure 8.1: Structure of a JOIN_ME packet
Source: Original work

The 2 byte **Company Identifier** that is part of the Manufacturer Specific Data header has been set to a reserved value (0xABCD) as long as no official company registration has taken place. A company identifier should be registered with the Bluetooth SIG.

A **Mesh Identifier** has been selected with a length of two bytes that shall be used to check if the packet is intended for FruityMesh (0xCAFE). This allows for multiple protocols with the same Company Identifier.

The **Network Identifier** allows to have multiple networks in the same physical space and prevents mix-up of discovery packets.

The last value that belongs to the custom advertising message header is the **Message Type**, which allows to send a total of 256 different messages of which only 4 are currently defined. One of these messages is the JOIN_ME packet that is explained here as an example.

The JOIN_ME packet contains all the information that a receiver must know to decide whether it wants to connect to this node or not. This includes the Node Id, Cluster Id, Cluster Size and the number of free connections. Further research must show if additional data should be integrated in this packet.

The **Write Handle** is discussed later in section 8.3.3.1 whereas the usage of the ACK field for slave connections has already been explained in section 7.3.5.

8.2.2 Connection Packets

Connection packets do always include the Message Type, Node Id of the sender, and that of the receiver. It would be possible to omit the sender and receiver ids when transferring multiple packets with unchanged values. This would decrease the overhead by 20% but has not been done in this implementation to keep it simple.

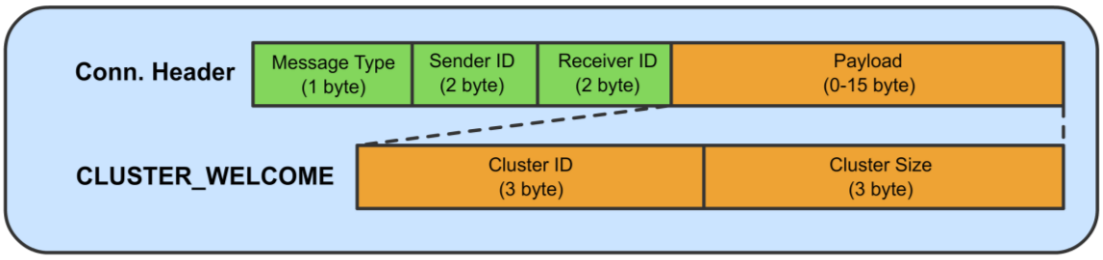


Figure 8.2: CLUSTER_WELCOME packet structure
Source: Original work

The shown CLUSTER_WELCOME packet in **Figure 8.2** is one of the packets that can be sent through connections, and it is the first one that is sent during the handshake once a connection has been build up.

8.3 Implementation

The implementation started as two separate projects that had to be flashed to different devices with one acting as a Peripheral and the other as a Central. In a later stage it was possible to integrate all functionality in one project and to allow parallel flashing on multiple dongles. Node Ids are currently linked to the serial number of the chip.

The next sections provide an overview of some implementation details.

8.3.1 Event Handling

The main function uses an event handling loop that is entered as soon as the initialization phase is completed. The event handler routine blocks as soon as there are no more events to process and lets the device sleep until an event is generated in the SoftDevice. This event handling is thread save and events will be processed in the order in which they are generated. An event will be dispatched to different event handlers for advertising, scanning, services and connections where it is either pre-processed and then delegated to the C++ classes or completely handled.

Because cluster size changes are communicated through increase or decrease messages, it is important that a node always handles these messages correctly because a failure to do so will result in inconsistent data.

8.3.2 Node States

The algorithm uses a state machine to manage its tasks, which is represented in a simplified form in **Figure 8.3**.

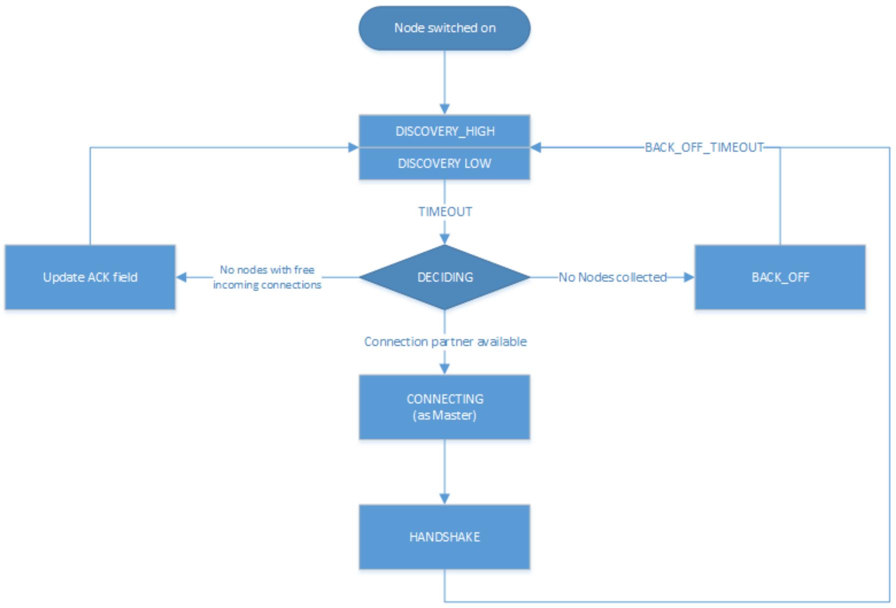


Figure 8.3: Simplified FruityMesh state flowchart
Source: Original work

After a node is initialized, it enters the DISCOVERY_HIGH state in which it tries to connect to other clusters as soon as possible. After a timeout of 5 seconds, it changes to the DECISION state where it decides whether it wants to connect to another cluster. If no other cluster was found, it sleeps for a few seconds in the BACK_OFF state to conserve energy. After several loops without finding another cluster, it will use the DISCOVERY_LOW state instead of DISCOVERY_HIGH. This allows the node to save more energy by scanning and advertising less and is important when a node is switched on in places without other nodes or if the network has been fully discovered. The node will change to the DISCOVERY_HIGH state again as soon as it receives a packet from a different cluster.

When a smaller cluster has been found, the node attempts to connect to it (CONNECTING) which is followed by a handshake procedure (HANDSHAKE).

The ACK field is updated as part of the Slave Connection Procedure.

8.3.2.1 DISCOVERY States

The DISCOVERY_HIGH and DISCOVERY_LOW state do only differ in the duty cycles of advertising and scanning operations and the timeouts.

In these states, a node alternates advertising and scanning for JOIN_ME packets. If a packet is received, it is saved to the JOIN_ME buffer with a timestamp. An older JOIN_ME packet from the same node is always overwritten in the buffer. Old packets are also replaced when there is no more space in the JOIN_ME buffer left. No further processing is done in the DISCOVERY states.

8.3.2.2 DECISION State

The buffered packets are evaluated with the cluster score function in order to determine the best connection partner. At first, the node tries to find a connection partner that can accept an incoming connection. It will try to connect if one is found. If there are no good candidates available, it tries to invoke the Slave Connection Procedure that has been explained in section 7.3.5. If there is a packet in the buffer that contains its Node Id in the ACK field, it must disconnect its connections and change to the DISCOVERY_HIGH state again.

8.3.2.3 BACK_OFF State

The BACK_OFF state helps to reduce energy consumption. It has a random timeout that prevents nodes from simultaneously entering the state multiple times. This would prevent nodes from discovering each other.

8.3.3 Connections

While the ATT protocol is not the best fit for managing data streams it can be made to work in such a way. This has been done by implementing a mesh Service with a single Characteristic that is used to transfer data. Both connected devices are allowed to send Write Commands which trigger an event handler on the receiving side. This enables a bidirectional connection. Write Requests have not been implemented but are required in the future to enable a reliable communication channel.

Because every device knows the Node Id of its connection partners, it is possible to implement a GATT bridge. This would allow us to exploit the strength of the GATT protocol and use all of its features over the mesh. With a GATT bridge, it is also possible to query existing BLE devices over the mesh that are not capable to work in a mesh. This is a key feature that should be implemented in a future version.

8.3.3.1 GATT Discovery

Service and Characteristic handles have to be discovered in a challenge and response scheme before accessing them. This is typically done after a connection has been made. Handles can be cached for bonded devices but must be acquired for each unknown connection partner. This consumes both time and energy. It is not possible to assume that handles are identical on all devices because this would lead to problems with different implementations and would violate the standard.

The current prototype performs a GATT discovery for every connection but it would be possible to broadcast the mesh write Characteristic handle in the JOIN_ME packet in order to skip the discovery procedure. This enables a much faster mesh setup.

8.3.3.2 Handshake

After discovering the handles, the mesh has to perform a handshake during which both nodes send their current cluster information (CLUSTER_WELCOME). The connection is only set active after this handshake has finished and is not used to relay any data in the meantime.

8.3.3.3 Active Connection

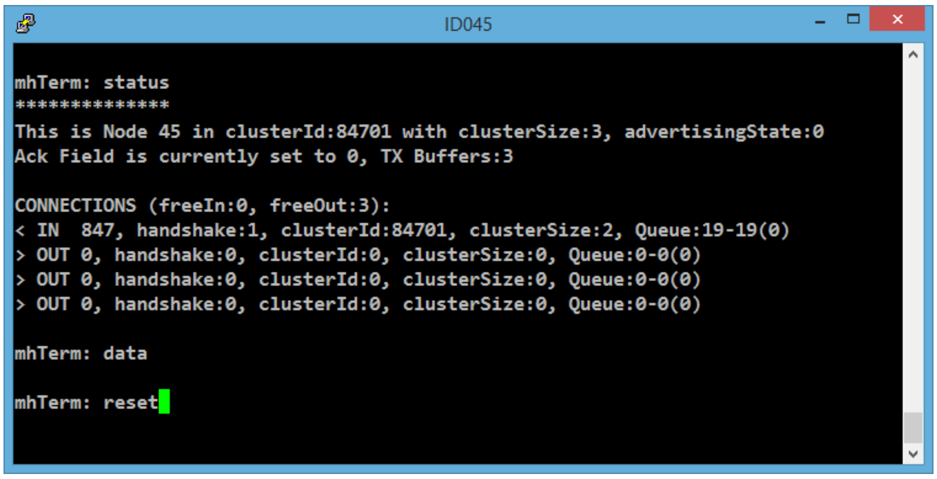
An active connection can receive data packets and relay them to other nodes. A message is relayed to all connections if the destination Node Id is set to 0, which signifies a broadcast packet. There are multiple types of possible messages. CLUSTER_INFO_UPDATE messages are used to transmit the latest cluster size and id. Raw data messages can be used to send different types of information while quality of service messages are used for reporting battery or connection states.

8.4 Implementation Caveats and Solutions

8.4.1 Testing and Debugging

While debugging is possible with either KEIL - or in an unstable form with Eclipse - it is not the ideal way to understand the program flow. Because the SoftDevice uses hardware timers and interrupts to implement the BLE stack, stepping through the application is only possible until a SoftDevice call is reached. The SoftDevice will then trigger an error handler or a Hard Fault that will reboot the device. Connections will of course be lost as soon as the application execution is interrupted. Code optimizations carried out by the ARM compiler make debugging even harder, because they eliminate function return values and error codes from the codebase.

There was no easy way to output a log file either, so it was necessary to develop a method that allows controlling the program flow and enables writing outputs to a console. This module has been called **mhTerm** and implements a UART terminal with input and output capabilities.



```
mhTerm: status
**************
This is Node 45 in clusterId:84701 with clusterSize:3, advertisingState:0
Ack Field is currently set to 0, TX Buffers:3

CONNECTIONS (freeIn:0, freeOut:3):
< IN  847, handshake:1, clusterId:84701, clusterSize:2, Queue:19-19(0)
> OUT 0, handshake:0, clusterId:0, clusterSize:0, Queue:0-0(0)
> OUT 0, handshake:0, clusterId:0, clusterSize:0, Queue:0-0(0)
> OUT 0, handshake:0, clusterId:0, clusterSize:0, Queue:0-0(0)

mhTerm: data

mhTerm: reset
```

Figure 8.4: mhTerm connected to PuTTY[8]
Source: Application screenshot

Several commands allow to control the application flow from a serial console client like PuTTY. Error codes are converted to human readable text and are written to the console. A variety of log functions allow to print error and program flow messages easily. In its first iteration, the terminal blocked code execution when waiting for a valid UART partner, which required a different application build for dongles that were not connected to USB. In its latest iteration, it will deactivate when it is plugged into a power source and will reactivate if it receives a valid input.

More testing was done with the nRF Sniffer and the Master Control Panel. In order to develop the Peripheral side of the prototype, the Master Control Panel was very helpful in providing a

[8] http://www.chiark.greenend.org.uk/~sgtatham/putty

reliable Central that could build up connections, enumerate Services and write or read Characteristics. Unfortunately, it cannot advertise or accept connections, so the Central side had to be developed against the existing codebase.

8.4.2 SoftDevice States

It has been found that Peripheral will stop advertising as soon as it receives an incoming connection. A Central will however still continue to broadcast advertising packets after connecting to a Peripheral. The SoftDevice generates a connection event in both cases but the application logic must decide whether the advertising has stopped. The same is true for scanning, which is stopped when the Central enters the Initiating state. A Peripheral, on the contrary, continues to scan when it receives an incoming connection. This behaviour stems from the fact that the link layer has multiple state machines running at the same time.

It is important to handle all events correctly because there is no API that can be used to read the current SoftDevice state. Calling a function in the wrong state results in an NRF_INVALID_STATE error being thrown. This has been a major issue in the development process because the internal state of the SoftDevice is not always known as the project size grows. Being able to access the internal state through an API would simplify the development process.

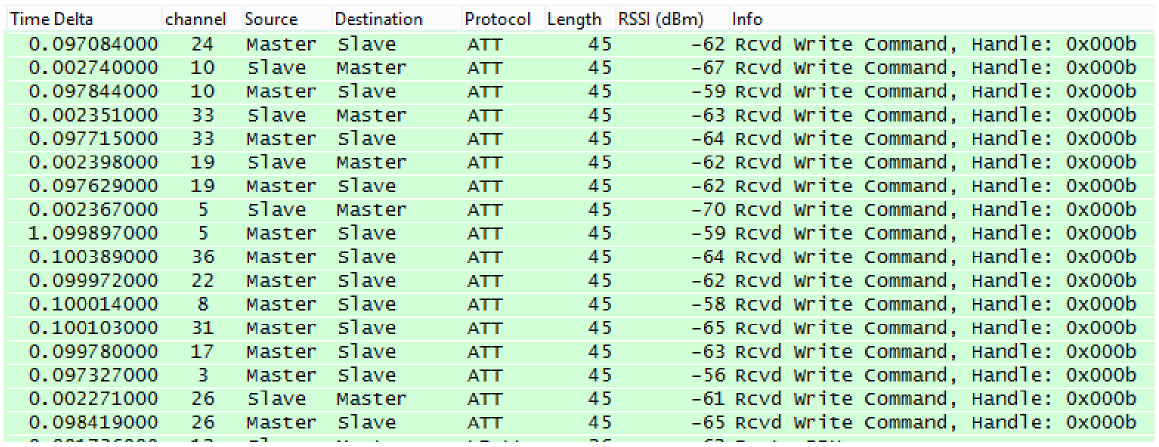
8.4.3 SoftDevice Error Handling

The SoftDevice implementation is very strict in following the BLE specification and includes a lot of error handling. Unfortunately, there is only a small range of error codes with no way to know the exact cause of the error. For example, trying to send connectable advertising packets when the S130 is already connected to a master will result in an NRF_INVALID_DATA error being thrown. This is the same error that is thrown when the advertising data is improperly formed.

8.4.4 Connection Events and Scheduling

In order to send data efficiently and with the least possible latency, it should be sent in batches. This means that multiple packets should be sent per Connection Event as seen in **Figure 2.9** during the second Connection Event. The fastest way to send data is to either use Write Commands or Notifications. These are unacknowledged and enable fast packet transmission with about 3-4 packets per connection event in the S110 SoftDevice implementation (Nordic Devzone 2014a).

After packet buffering had been implemented and did not show any performance improvements, further research with the nRF Sniffer suggested that the nodes do not support more than one packet per connection event. **Figure 8.5** shows that only one packet is sent per direction for every connection interval, which is 100 ms. This is due to the S130 implementation which is based on the S120, where this limit originates from (Nordic Devzone 2014c). Hopefully, this limit is fixed in one of the future versions.



Time Delta	channel	Source	Destination	Protocol	Length	RSSI (dBm)	Info
0.097084000	24	Master	Slave	ATT	45	-62	Rcvd Write Command, Handle: 0x000b
0.002740000	10	Slave	Master	ATT	45	-67	Rcvd Write Command, Handle: 0x000b
0.097844000	10	Master	Slave	ATT	45	-59	Rcvd Write Command, Handle: 0x000b
0.002351000	33	Slave	Master	ATT	45	-63	Rcvd Write Command, Handle: 0x000b
0.097715000	33	Master	Slave	ATT	45	-64	Rcvd Write Command, Handle: 0x000b
0.002398000	19	Slave	Master	ATT	45	-62	Rcvd Write Command, Handle: 0x000b
0.097629000	19	Master	Slave	ATT	45	-62	Rcvd Write Command, Handle: 0x000b
0.002367000	5	Slave	Master	ATT	45	-70	Rcvd Write Command, Handle: 0x000b
1.099897000	5	Master	Slave	ATT	45	-59	Rcvd Write Command, Handle: 0x000b
0.100389000	36	Master	Slave	ATT	45	-64	Rcvd Write Command, Handle: 0x000b
0.099972000	22	Master	Slave	ATT	45	-62	Rcvd Write Command, Handle: 0x000b
0.100014000	8	Master	Slave	ATT	45	-58	Rcvd Write Command, Handle: 0x000b
0.100103000	31	Master	Slave	ATT	45	-65	Rcvd Write Command, Handle: 0x000b
0.099780000	17	Master	Slave	ATT	45	-63	Rcvd Write Command, Handle: 0x000b
0.097327000	3	Master	Slave	ATT	45	-56	Rcvd Write Command, Handle: 0x000b
0.002271000	26	Slave	Master	ATT	45	-61	Rcvd Write Command, Handle: 0x000b
0.098419000	26	Master	Slave	ATT	45	-65	Rcvd Write Command, Handle: 0x000b

Figure 8.5: Two nodes with alternating transmissions (Time Delta in seconds)
Source: Wireshark capture screenshot

8.4.5 Adaptive Frequency Hopping

The S130 implements ADFH by choosing a different channel map for each new connection. In a mesh topology, this channel selection could be enhanced by setting the channel map through the application as allowed by the current BLE specification (Bluetooth 4.2, Vol. 6, Part B, 4.5.8.1). This is not yet possible with the S130 API but is planned in the future (Nordic Devzone 2014b).

8.4.6 Restricting Advertising and Scanning Channels

In some cases it is preferable to restrict advertising or scanning to a subset of the three available channels. This is currently not possible with the S130 SoftDevice but is already implemented in the nRF Sniffer firmware or the latest S110 v8 (Nordic Devzone 2015a).

8.4.7 Managing Channel Access

Most mesh technologies have control over the media access layer that manages frequency and time multiplexing. This enables them to work with a combination of point-to-point communication channels and a set of channels that are accessed by multiple devices at once to enable broadcast communication. The BLE specification mentions that "[u]nlike BR/EDR slaves, LE slaves do not share a common physical channel with the master. Each slave communicates on a separate physical channel with the master" (Bluetooth 4.2, Vol. 1, Part A, 4.1.2). Allowing a shared physical channel could enhance battery lifetime in a mesh topology without breaking backward compatibility with existing devices.

8.4.8 Link Layer Processing

Reacting on advertising messages is not possible from the application layer because of the strict timing requirements for sending a connection initialization packet once an advertising message has been received, which is a time span of only 125 µs (T_IFS). In contrast, receiving advertising packets is not possible while the link layer is in the Initiating state according to the specification (Bluetooth 4.2, Part 6, Part B, 4.3.4). This results in a lot of packets that are discarded and leads to longer scan times, which decrease the performance of the discovery process.

The current prototype uses longer discovery times for the ease of implementation but could be enhanced by populating a connection whitelist with a set of nodes that achieved a similar cluster score instead of communicating only the best connection partner to the link layer. This will reduce the scanning time that is needed in the Initiating state.

8.4.9 Defective Nodes

A defective node may be stuck in a reboot cycle, leading to battery depletion in other nodes. If discovery packets are sent often, other nodes will try to contact this node and will consume a lot of power because of being stuck in discovery mode. Therefore, a counter should be introduced that disables failing nodes after a series of faults.

8.4.10 Connection Timeouts

The current implementation experiences connection timeouts after around 30 minutes in a connected mesh when discovery is enabled. Connection timeouts did not occur when scanning and advertising were disabled. Because there is only little custom application logic running by the time the mesh has formed, it is believed that these timeouts are caused by the scheduling that happens inside the SoftDevice. The timings were all chosen in a way that should enable the scheduler to maintain connections without problems.

This issue could not be solved because the API does not provide sufficient information to investigate the cause of the problem and no measurement equipment was available.

8.5 Further Ideas

Some further ideas and concepts have been collected to improve the network performance but were not implemented because they were out of scope for this thesis.

By using **data aggregation**, it is possible to combine multiple packets in one packet. This reduces the overhead of the protocol and can be used to save energy. Advanced algorithms for data compression and filtering can be used as well. But these can all be implemented on top of the protocol without altering its functionality.

The Cluster Score function can be altered to take several aspects into account. For example broadcasting **the remaining battery resources in the JOIN_ME packet** will result in a network that is first built with devices with extended power supplies. After the core network has been built, it will be joined by the remaining devices with less battery capacity. These will have to manage a lower number of links and data packets. It is also preferable to **include the Received Signal Strength Indicator (RSSI)** in this packet in combination with a **dynamically configured TX** power in order to achieve stable links that need less transmission power and have a higher robustness against interference.

Smartphones and other moving devices should broadcast a **mobile-flag** in their discovery packets in order to be excluded from the network's core. It is better to add moving devices as edges to the network tree so that the network does not need to constantly reform.

Timeouts must be chosen in a way that connection loss is only encountered sparsely to prevent the network from reconfiguring. The timeout should depend on the environment in terms of interference and physical facts and on the speed at which the network must react to changes.

Connection properties like Slave Latency and Connection Interval can be handled between two nodes independently from the network. If both are connected to power they can agree on a small connection interval. This would reduce the hop latency and improve overall network performance. Another improvement that is currently not implemented would require nodes to use a low connection interval for fast network setup and scale down afterwards in order to shorten the discovery time.

9 Mesh Performance

This chapter discusses the results and the expected performance of the algorithm.

9.1 Discovery Time

The time it takes to discover other nodes is very dependent on the duration of the DISCOVERY and BACK_OFF states as well as the scanning and advertising schedule. These values have been manually selected in the implementation and provided good results where nodes would typically connect within 5 seconds to an available connection partner; the duration that has been chosen for the DISCOVERY_HIGH state. The office environment that was used for testing had a high Wi-Fi traffic and some Bluetooth Beacons that were transmitting continually. In case where multiple free mesh nodes were available, the connection time decreased. There is more research required to find a good set of durations and intervals that are independent of the environment. It would also be possible to adapt these values to the environment based on scanning results.

The used settings in the prototype are the following:

- Time from DISCOVERY_HIGH to DECISION state: 5 seconds
- BACK_OFF state timeout: 3-8 seconds at random
- Advertising interval: 100 milliseconds
- Scan interval: 100 milliseconds with a window of 40 milliseconds

The simulation results from chapter 7 have been used to calculate approximate times for the initial network configuration by comparing multiple simulations of 6 nodes with the average performance of the actual prototype. This resulted in a conversation ratio of 3:5 from simulation time in frames to observed time in seconds. These numbers should at least provide an idea of what can be achieved with the current prototype implementation.

The sparse network configuration with 20 nodes as seen in **Figure 7.3** connected in about 80 seconds while the dense configuration took 90 seconds. With 200 nodes, this time increased to approximately 4 minutes while building a mesh with 400 nodes has been simulated as only 3 minutes.

9.2 Bluetooth Low Energy Throughput

Bluetooth Low Energy has not been designed for high data throughput, which is always in stark contrast to battery efficiency. It is however important to investigate how short term data peaks can affect the mesh.

The BLE physical layer transmits with a symbol rate of 1 Mbit/s. After accounting for protocol overhead and limitations, this allows for a maximum theoretical throughput of 236.7 Kbit/s in Bluetooth 4.0 according to (Gomez et al. 2011):

$$\delta_{max} = \frac{symbolRate}{connectionInterval} * \frac{connectionInterval}{T_{Data} + T_{ACK} + 2 * T_{IFS}} * payloadBytes \qquad (1)$$

With Bluetooth 4.2, the maximum Data Channel PDU has been increased and does now support up to 215 bytes of payload with the LE Data Packet Length Extension (Bluetooth 4.2, Vol. 6, Part B, 4.6.6). Both devices must first agree on their link layer capabilities with the appropriate Link Layer Control messages. If we use these values to calculate the maximum throughput, we get a much higher 800 Kbit/s. The LE Data Length Extension can be used to achieve a better performance when using IPv6 over BLE which was also introduced with Bluetooth 4.2. But these values are never reached in actual implementations as the data rate decreases significantly with bit errors and implementation specifics. For Bluegiga BLE modules, the real world maximum is around 60 Kbit/s, which will decrease further in a noisy environment (Bluegiga 2012).

The maximum throughput for the Nordic nRF51 chipset in combination with a SoftDevice can be obtained from the corresponding specification. The S110 v8.0.0 can send write commands with 125 Kbit/s (Nordic Semiconductor ASA 2015d, Section 13) while the S120 v2.0.0 limits sending and receiving to 1 packet per connection event which results in a limit of 8 Kbit/s for each of the – up to 8 - connections (Nordic Semiconductor ASA 2015c, Section 13). The throughput for the S130 SoftDevice is not yet specified when both Central and Peripheral connections are used (Nordic Semiconductor ASA 2015d, Section 12) but is believed to be similar to the limits imposed by the S120.

In addition, there is an API restrictions that limits the throughput of the S130 SoftDevice. The application does only have one write buffer for all connections that can currently fit two data packets. In case there are four connection partners A, B, C and D and the buffer is filled with packets destined to partner B, the SoftDevice will skip all connection events for A, C and D until this packet is sent. The application has no knowledge about the connection link that will be processed next and must fill the write buffer at random. This causes many connection events to be skipped. (Nordic Devzone 2015b)

9.3 Latency

The latency for delivering a message throughout the mesh depends on the topology, connection intervals, interference and connection scheduling.

The topology that is formed depends on the used Cluster Score function, the environment and on the succession in which devices are activated. This can either lead to a mesh that forms a long connection line with multiple hops or to a deep tree where most nodes use all of their connections.

Because of the API and S130 restrictions for packet transmission that were mentioned previously, there is a higher hop latency in the current implementation than what should be achievable. With an improved API, a hop should not take more time than the configured Connection Interval. This does of course not consider dropped packets because of interference.

Synchronizing nodes through the mesh and using advertising messages to overcome limitations in the mesh topology would be possible but has not been evaluated in this thesis. It would also be possible to use gateway nodes that route packets through a different network and feed it back into the network at a different point to optimize latency.

9.4 Battery Lifetime

One of the key aspects for a low energy mesh technology is without a doubt the expected battery lifetime. With one active connection and a Connection Interval of one second, the nRF51 chipset is expected to be operational for 1.3 years on a typical CR2032 battery according to the values from the official specification. When operating in a mesh topology, this time is expected to decrease because multiple connections need to be managed.

An accurate prediction depends on a running mesh with a high number of nodes that are evaluated in different environments under different signal interference. The power drain of every mesh node would need to be observed over a long period of time. This has not been done in this thesis because it would have required:

- More nodes than the seven that were available
- Expensive oscilloscopes to measure the current flow
- Different controlled testing environments
- A lot of time

It is possible to estimate the battery drain based on a few assumptions. If we assume a mesh with static nodes and if the node density and connection supervision time are chosen according to the environment, the mesh will stay stable over very long periods of time. Mesh restructuring will only occur rarely because of device failures.

If the mesh is in a stable state, a configuration message can be sent to disable discovery for new nodes. Losing a node would result in mesh discovery being enabled again. In a semi-dynamic mesh, discovery can be left active at all times to enable the discovery of new nodes or to enable a faster mesh healing.

9.4.1 Phase 1: Network Discovery

Nodes will consume a lot of energy during discovery. If the highest duty cycle is chosen, the runtime will be around 10 to 20 hours for the nRF51 chipset according to (Trond Einar Snekvik 2014). This is not critical because the discovery phase will only consume time in the order of seconds or minutes depending on the number of nodes involved.

9.4.2 Phase 2: Stable Mesh

In a stable mesh, each node will be in connection with up to four nodes. At this point, the S130 does not have a finished technical specification. The S110 (Nordic Semiconductor ASA 2015b) and S120 (Nordic Semiconductor ASA 2015c) specifications were instead used to approximate the current consumption (cf. Appendix 1). The used parameters were:

- Transmit Power: 0 dBm
- Connection Interval: 500 ms
- Master and Slave Sleep Clock Accuracy: 20 ppm

This results in an average current consumption of 27 µA to maintain one connection. A typical coin cell battery with a capacity of 230 mAh could therefore maintain this connection for about 1 year. The accurate consumption depends on a few more factors (whether it is a master or slave connection, which hardware peripherals are used, the chipset, post processing time, etc.) and cannot be determined exactly without actual measurements.

If we send periodical discovery messages every 5 seconds, this will result in an additional current consumption of about 6 µA. Scanning consumes 13 mA and will therefore add an average 65 µA if it is used 0.5% of the time. It is therefore advisable to keep discovery off or at a low schedule. Continuously scanning for other devices, such as Smartphones, is impossible when operating on batteries.

Maintaining multiple connections will result in a shorter battery lifetime, e.g. 4 months for three active connections. It is possible that the SoftDevice schedules these connections in a way that enables a longer battery lifetime because it does not have to switch to sleep mode between every connection and does therefore not require a start-up current each time. This is visible in nRFgo studio when simulating different connection intervals. Using an interval of 250 ms instead of 500 ms does not double the required current which means that the results are expected to be better. Using two AA batteries will increase the runtime of the mesh by a factor of 10 and will therefore allow truly battery powered nodes that run multiple years. And with recent advances in energy harvesting - which allows to collect the energy that is transported by surrounding radio waves from other technologies - it is possible to build nodes that run infinitely.

9.5 Security

The current implementation does not have encryption built-in because of the implementation labour needed. But because it uses standard Bluetooth connections, this should pose no problem.

BLE connections benefit from having link layer encryption integrated. AES-128 is used in conjunction with the CCM encryption mode. There are currently no known attacks against this encryption scheme but there have been efforts to undermine BLE security by targeting the connection setup process. One example is crackle (Mike Ryan 2013). The attack works because there is only a low number of randomness involved in the key used to pair two devices. This security hole can be overcome by using an out of band method for transmitting the encryption keys.

Before a node can join a mesh network it must know which network it should join. This is done by giving it a Network Id and a unique Node Id for that network. In the same process, the Network Key should be configured. Using BLE for the setup process works but risks that an attacker intercepts the messages as a man-in-the-middle and will therefore come in possession of the network key.

Using an out-of-band method for this process provides additional security. Two different methods are proposed here:

Each dongle can be equipped with a **QR code** that includes the device access address and a public key. A user can then use an app that scans this code and automatically configures the node for the network by setting Network Id, Node Id and the Network Encryption Key.

A second method is to use a **USB connector** on the dongle to transmit the network configuration. This can be preferable if a large number of nodes must be set up at once. The USB connection can be used to update the nodes to the newest firmware and to flash the network configuration at the same time.

Another security risk with the current implementation is found in the mesh discovery messages which are unencrypted and can be easily forged. While it is not possible to join the network without possessing the Network Encryption Key, an attacker can confuse the network by sending forged discovery messages that will cause nodes to break up their mesh connections and render the network unusable. Solving this issue, e.g. with a nonce, involves more work because there is no standardized way of using encryption or authentication for advertising packets. The nRF51 platform does only expose the very basic AES-ECB mode while the SoftDevice is active.

10 Showcases

Two showcases have been implemented with the prototype and are briefly presented in this chapter.

10.1 Connection Monitoring and Meshing

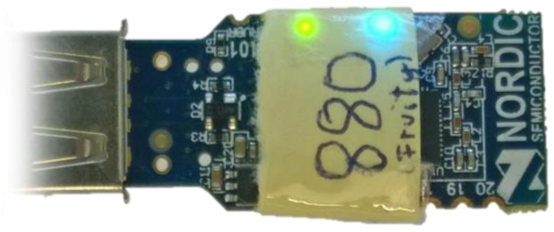


Figure 10.1: Fruity mesh dongle signalling its connection state
Source: Original work

When a dongle, equipped with FruityMesh, is plugged into a USB power source it will at first try to contact a valid UART terminal. If it does not find one, it will start its discovery phase and look for other devices. It signals this with a red blinking LED. As soon as it discovers that another device is active within its range it will try to build up a connection. A green LED will signal that a connection has been made before turning blue after the handshake has finished. The number of connected nodes is indicated by the blink pattern which is a sequence of flashes before a longer delay.

10.2 Campaign Management

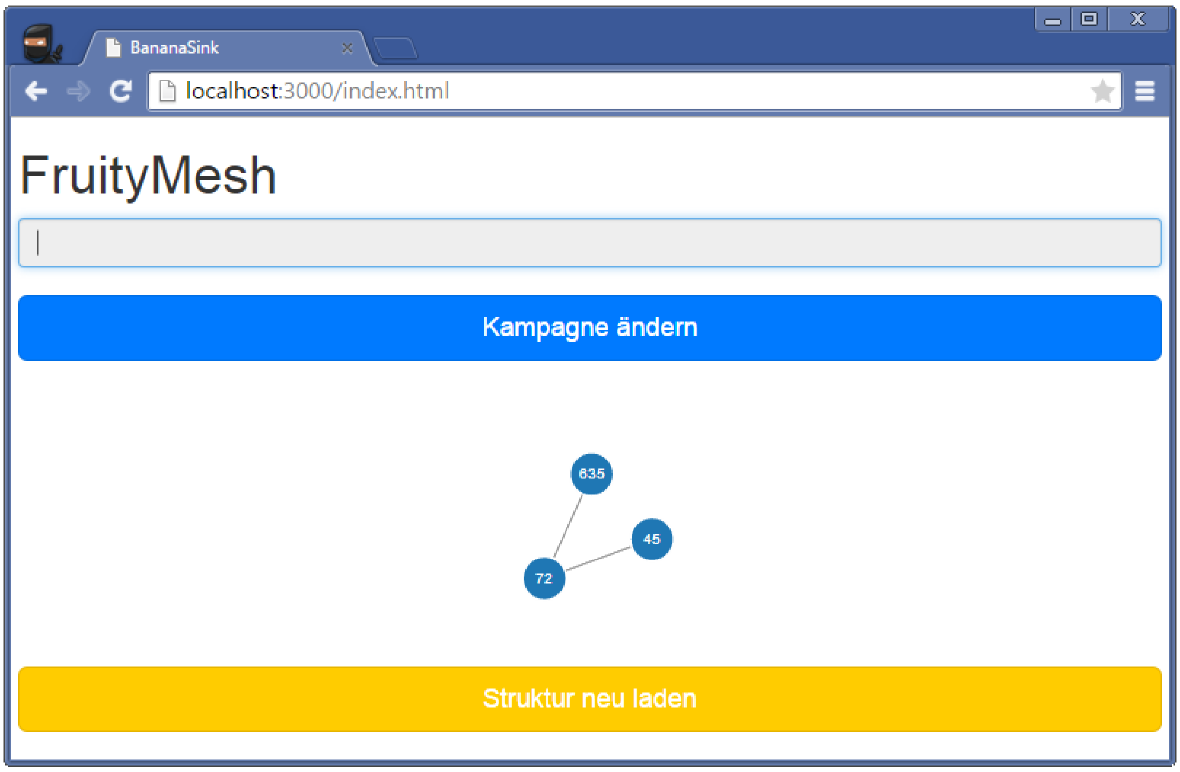


Figure 10.2: BananaSink web interface for monitoring connections
Source: Application screenshot

A more advanced showcase can be used to monitor the mesh structure and to distribute an advertising campaign with the mesh. The setup process is detailed in Appendix 3.

Once some dongles have been plugged into a power source, and after they have connected to the mesh, it is possible to monitor their connections with a web interface as seen in **Figure 10.2**. With the same interface, it is then possible to distribute a short advertising campaign throughout the mesh.

Figure 10.3: iOS campaign viewer app
Source: Application screenshot

Once this has been done, the provided iPhone app depicted in **Figure 10.3** can be used to display the current campaign message. As soon as the app is started, it will start to scan for nearby FruityMesh nodes and read the advertising messages if available.

This showcase has been implemented in the current FruityMesh firmware and uses a Node.js webserver coupled with serial port communication to read and write data through UART. The web interface is implemented in JavaScript and HTML.

11 Conclusion

The current implementation of the prototype and the simulation have shown that it is possible to implement a mesh network on top of the existing Bluetooth standard. As of writing, this is the only known implementation building on top of the current standard that can be used with battery-powered nodes.

Multiple topics of future research have been pointed out in this thesis and a great number of ideas still have to be implemented and tested. The most important topics are the evaluation of different Cluster Score functions and to test and optimize the prototype with more devices in different environments. Some of the API restrictions currently limit the functionality in terms of connection timeouts and throughput. These should be resolved in future implementations through changes to the S130 or by developing workarounds.

For now, the FruityMesh algorithm has provided promising results during simulation and must now prove its usefulness in real-world use-cases.

A. Glossary

AD Structure	Advertising structure.
ADFH	Adaptive Frequency Hopping. Describes a sequence in which two devices synchronously change channel frequencies in order to avoid interference.
AES-CCM	Counter with CBC-MAC. An advanced AES mode that provides authentication and confidentiality.
AES-ECB	Electronic Codebook mode for the AES encryption is the most basic mode that does not use chaining.
Bluetooth SIG	Bluetooth Special Interest Group. A group that oversees the development and standardization of the Bluetooth standard.
CRC	Cyclic Redundancy Check. Used to check that a message does not contain any bit errors.
DC/DC	A DC to DC converter is an electronic component that transforms a direct current voltage to another.
DFU	Device Firmware Upgrade. Can be done over different transports, either wired or wirelessly.
GPIO	General purpose input and output ports are used to either read a signal from a pin connector or to send a signal. They can be configured with software.
HCI	Host Controller Interface. A standardized communication protocol that is used for communication between the host and the controller.
IEEE	Institute of Electrical and Electronic Engineers, international organization for scientists and engineers with background in Electronics and Computer Science.
IoT	Internet of Things. A common term used for all kinds of devices of the everyday live that are able to access the internet.
MIC	Message Integrity Check. Like a CRC but for verifying encrypted messages.
QoS	Quality of Service. Describes different qualities like latency, throughput or error rate and can be used to measure the quality of a link.

RFU	Reserved for Future Use. Fields in the protocol that should be set to zero and should be ignored when received.
RSSI	Received Signal Strength Index. Can be used to approximate the distance between two devices.
RX	Abbreviation of the verb receive.
Sensor Tag	A device that contains sensors and a BLE transceiver to communicate its measurements.
Sink	A sink is usually a data collection point. This can additionally be a gateway to another network technology.
SoC	System on a Chip. A single chip solution that packages all the different functionality for a specific task on a single self-contained chip.
TX	Abbreviation of the verb transmit.
UART	Universal Asynchronous Receiver / Transmitter. A protocol used for transferring bidirectional serial data, e.g. to a terminal or between chips.
UUID	An ISO standard for Universally Unique Identifiers.

B. List of Illustrations

C. Publication bibliography

Bluetooth 4.2, 12/2/2014: Bluetooth Core Specification. Available online at https://www.bluetooth.org/en-us/specification/adopted-specifications, checked on 5/18/2015.

ZigBee RF4CE Specification 1.01, January 2010: https://docs.zigbee.org/zigbee-docs/dcn/09/docs-09-5262-01-0rsc-zigbee-rf4ce-specification-public.pdf. Available online at https://docs.zigbee.org/zigbee-docs/dcn/09/docs-09-5262-01-0rsc-zigbee-rf4ce-specification-public.pdf, checked on 4/27/2015.

Advanticsys (2015): 802.15.4 Mote Modules. Available online at http://www.advanticsys.com/shop/wireless-sensor-networks-802154-mote-modules-c-7_3.html, checked on 4/8/2015.

AIRcable.net (2014): Smart Mesh Networking with Bluetooth Low Energy = CSRmesh™. Inside the CSRmesh™ 1.0. Available online at http://www.aircable.net/online/index.php?route=information/information&information_id=12, checked on 3/23/2015.

Ammari, Habib M. (2013): The Art of Wireless Sensor Networks. Dordrecht: Springer (Signals and Communication Technology).

Android Open Source Project (2015): Android 5.0 APIs. API Level: 21. Available online at https://developer.android.com/about/versions/android-5.0.html, checked on 3/31/2015.

ANT+ Alliance (2015): ANT in phones. Available online at www.thisisant.com/consumer/ant-101/ant-in-phones, checked on 4/8/2015.

Apple Inc. (2013): Bluetooth Accessory Design Guidelines for Apple Products (R7). Available online at https://developer.apple.com/hardwaredrivers/BluetoothDesignGuidelines.pdf, updated on 9/18/2013, checked on 3/31/2015.

Bluegiga (12/1/2011): Bluegiga enables the development of Bluetooth® 4.0 accessories for iPhone 4S. Espoo, Finland. Bluegiga Technologies. Available online at https://www.bluegiga.com/en-US/news/2011/bluegiga-enables-the-development/, checked on 3/31/2015.

Bluegiga (2012): Maximum throughput with Bluegiga modules. Available online at https://bluegiga.zendesk.com/entries/22400867--HOW-TO-Maximize-throughput-with-BLE-modules, updated on 11/15/2012, checked on 3/25/2015.

Bluegiga (2015): BLE121LR Bluetooth Smart Long Range Module. Available online at https://www.bluegiga.com/en-US/products/ble121lr-bluetooth-smart-long/, checked on 3/25/2015.

Bluetooth SIG (2015a): Bluetooth Smart and smart Ready products now available. Available online at http://www.bluetooth.com/Pages/Bluetooth-Smart-Devices-List.aspx, checked on 3/31/2015.

Bluetooth SIG (2015b): Skyrocketing demand for Bluetooth appcessories for latest phones. Available online at http://www.bluetooth.com/Pages/Mobile-Telephony-Market.aspx, checked on 3/23/2015.

Bluetooth SIG (2/24/2015): February 24th: Bluetooth® Technology Adding Mesh Networking to Spur New Wave of Innovation. Kirkland, Washington, USA. Available online at http://www.bluetooth.com/Pages/Press-Releases-Detail.aspx?ItemID=224, checked on 4/7/2015.

Boukerche, Azzedine (2009): Algorithms and protocols for wireless and mobile ad hoc networks. Hoboken, N.J.: Wiley (Wiley series on parallel and distributed computing).

Burri, Nicolas; Rickenbach, Pascal von; Wattenhofer, Roger: Dozer: Ultra-Low Power Data Gathering in Sensor Networks. In : 2007 6th International Symposium on Information Processing in Sensor Networks. Cambridge, MA, USA, pp. 450–459.

c|net (2015): Lollipop nibbles off 1.6 percent of Android devices. Edited by Lance Whitney. c|net. Available online at http://www.cnet.com/news/lollipop-nibbles-off-1-6-percent-of-android-devices/, updated on 2/3/2015, checked on 3/31/2015.

CSR (2/25/2014): Game-changing Bluetooth® Smart solution enables whole home control from the smartphone for the first time. CAMBRIDGE, UK AND SAN JOSE, CALIFORNIA, USA, checked on 4/7/2015.

Dementyev, A.; Hodges, S.; Taylor, S.; Smith, J. (2013): Power consumption analysis of Bluetooth Low Energy, ZigBee and ANT sensor nodes in a cyclic sleep scenario. In : Wireless Symposium (IWS), 2013 IEEE International, pp. 1–4. Available online at http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6616827.

Dialog Semiconductors (2015): Multiple Piconet and Master / Slave. Available online at http://support.dialog-semiconductor.com/multiple-piconet-and-master-slave, checked on 4/1/2015.

EETimes (3/12/2014): Bluetooth 4.2 Unveiled: No Mesh Yet, But Big on IoT. Available online at http://www.eetimes.com/document.asp?doc_id=1324835, checked on 4/8/2015.

Forbes (10/21/2014): Qualcomm Acquires CSR To Accelerate Its Growth In IoT. Available online at http://www.forbes.com/sites/greatspeculations/2014/10/21/qualcomm-acquires-csr-to-accelerate-its-growth-in-iot/, checked on 4/8/2015.

Gomez, C.; Demirkol, I.; Paradells, J. (2011): Modeling the Maximum Throughput of Bluetooth Low Energy in an Error-Prone Link. In *Communications Letters, IEEE* 15 (11), pp. 1187–1189. DOI: 10.1109/LCOMM.2011.092011.111314.

Heinzelman, W. R.; Chandrakasan, A.; Balakrishnan, H. (2000): Energy-efficient communication protocol for wireless microsensor networks. In : System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on, pp. 10 pp. vol.2. Available online at http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=926982.

Intel (1/15/2015): Intel Reports Record Full-Year Revenue of $55.9 Billion. Trey Campbell. Available online at http://www.intc.com/releasedetail.cfm?ReleaseID=891521, checked on 3/23/2015.

Jelicic, Vana; Cesarini, Daniel; Bilas, Vedran (2013): Enhancing performance of wireless sensor networks in glacial environments using wake-up receivers. In *J. Phys.: Conf. Ser.* 450, p. 012045. DOI: 10.1088/1742-6596/450/1/012045.

Laird (2013): BLE Oveview. Available online at http://www.summitdata.com/blog/ble-overview/, checked on 4/27/2015.

Lindsey, S.; Raghavendra, C. S.: PEGASIS: Power-efficient gathering in sensor information systems. In : 2002 IEEE Aerospace Conference. Big Sky, MT, USA, 9-16 March 2002, pp. 3-1125.

Microsoft (2015): Bluetooth LE API. Available online at https://msdn.microsoft.com/de-de/library/windows/apps/windows.devices.bluetooth.bluetoothledevice.aspx, checked on 3/31/2015.

Mike Ryan (2013): crackle, crack Bluetooth Smart (BLE) encryption. Available online at https://lacklustre.net/projects/crackle/, updated on 2/15/2013, checked on 3/25/2015.

Mišić, Jelena; Mišić, Vojislav B. (2006): Performance modeling and analysis of bluetooth networks. Networks formation, polling, scheduling, and traffic control. London: Auerbach.

Mouser Electronics (2015): Part supplier. Available online at http://www.mouser.de/, checked on 4/1/2015.

Nordic Devzone (2014a): dealing large data packet's through ble. Available online at https://devzone.nordicsemi.com/question/1741/dealing-large-data-packets-through-ble/, checked on 4/22/2015.

Nordic Devzone (2014b): Frequency hopping on soft device for nrf51822. Available online at https://devzone.nordicsemi.com/question/5015/frequency-hopping-on-soft-device-for-nrf51822/, checked on 4/24/2015.

Nordic Devzone (2014c): Send multiple notifications per connection interval. Available online at https://devzone.nordicsemi.com/question/5121/send-multiple-notifications-per-connection-interval/, checked on 4/22/2015.

Nordic Devzone (2015a): Advertising on a single channel. Available online at https://devzone.nordicsemi.com/question/13384/advertising-on-a-single-channel/, checked on 4/24/2015.

Nordic Devzone (2015b): Get Softdevice schedule for connections. With assistance of Marius Heil, Stefan Birnir Sverrisson. Available online at https://devzone.nordicsemi.com/question/30345/get-softdevice-schedule-for-connections/, checked on 3/25/2015.

Nordic Devzone (2015c): SD130 restriction: 1 connection as slave and 3 connections as master. Available online at https://devzone.nordicsemi.com/question/28626/sd130-restriction-1-connection-as-slave-and-3-connections-as-master/, checked on 4/17/2015.

Nordic Semiconductor ASA (12/8/2014): Mesh networking platform uses Nordic Semiconductor nRF51822 SoCs to target Internet of Things applications. Oslo, Norway. Available online at https://www.nordicsemi.com/eng/News/News-releases/Product-Related-News/Mesh-networking-platform-uses-Nordic-Semiconductor-nRF51822-SoCs-to-target-Internet-of-Things-applications.

Nordic Semiconductor ASA (2015a): S130 Release Notes 1.0.0-3.alpha. Available online at https://www.nordicsemi.com/eng/Products/nRF51-Dongle#Download, checked on 4/13/2015.

Nordic Semiconductor ASA (2015b): SoftDevice S110 Specification. 2.0th ed. Available online at https://www.nordicsemi.com/eng/nordic/download_resource/38621/15/93740197, checked on 5/18/2015.

Nordic Semiconductor ASA (2015c): SoftDevice S120 Specification. 2.1st ed. Available online at https://www.nordicsemi.com/eng/nordic/download_resource/38623/14/56250474, checked on 5/18/2015.

Nordic Semiconductor ASA (2015d): SoftDevice S130 Specification. 0.5th ed. Available online at https://www.nordicsemi.com/eng/nordic/download_resource/38619/4/95719525, checked on 5/18/2015.

Permasense: Permasense Project. A Project of ETH Zurich, Uni Basel and Uni Zurich. Available online at http://data.permasense.ch/topology.html#topology, checked on 4/8/2015.

The Guardian (9/29/2014): FireChat – the messaging app that's powering the Hong Kong protests. Available online at http://www.theguardian.com/world/2014/sep/29/firechat-messaging-app-powering-hong-kong-protests, checked on 5/3/2015.

Townsend, Kevin (2014): Getting started with Bluetooth low energy. Sebastopol, CA: O'Reilly.

Trond Einar Snekvik (2014): Mesh topology in Bluetooth Low Energy. Prestudy. Norwegian University of Science and Technology.

Wikipedia (2015): Wireless ad hoc network. Available online at http://en.wikipedia.org/wiki/Wireless_ad_hoc_network, checked on 4/7/2015.

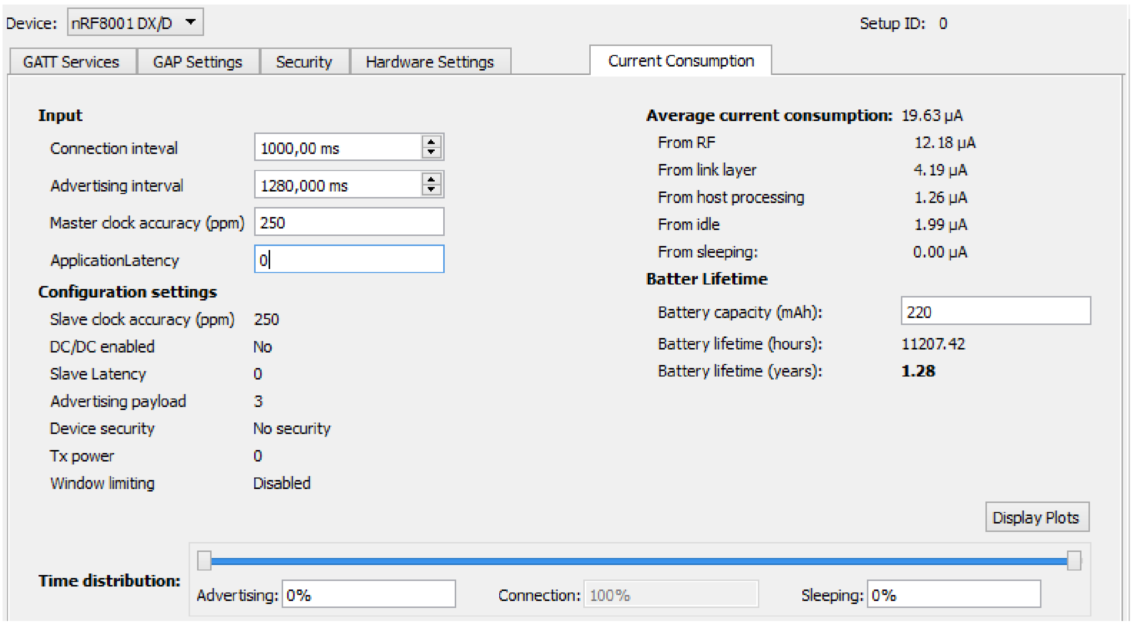
Wirepas Oy (2014): Wirepas Ltd. White Paper: Wirepas Pino(tm) Technology Overview. Available online at http://www.wirepas.com/files/6714/0747/9916/Wirepas_Ltd._White_Paper_-_Wirepas_Technology_Overview_-_Short_-_2014-02-06_v1.0_Public.pdf, checked on 4/7/2015.

Wirepas Oy (10/14/2014): Wirepas Introduces the First Fully Automatic and Distributed Wireless Multi-Hop Mesh IoT Network in the Market. Tampere, Finland. Teppo Hemiä. Available online at http://www.wirepas.com/news/wirepas-introduces-first-fully-automatic-distributed-wireless-multi-hop-mesh-iot-network-market/.

ZigBee Alliance (2015): ZigBee Home Automation. Available online at http://www.zigbee.org/zigbee-for-developers/applicationstandards/zigbeehomeautomation/, checked on 3/23/2015.

Appendix 1 Battery Consumption

The following simulation was made with nRFgo Studio 1.17.1.3252 that is distributed by Nordic Semiconductor ASA. The expected lifetime on a typical coin cell with 220 mAh is 1.28 years for the nRF8001 chipset when sending a packet each second.



Device: nRF8001 DX/D
Setup ID: 0

Input

Connection inteval	1000,00 ms
Advertising interval	1280,000 ms
Master clock accuracy (ppm)	250
ApplicationLatency	0

Configuration settings

Slave clock accuracy (ppm)	250
DC/DC enabled	No
Slave Latency	0
Advertising payload	3
Device security	No security
Tx power	0
Window limiting	Disabled

Average current consumption: 19.63 µA

From RF	12.18 µA
From link layer	4.19 µA
From host processing	1.26 µA
From idle	1.99 µA
From sleeping:	0.00 µA

Batter Lifetime

Battery capacity (mAh):	220
Battery lifetime (hours):	11207.42
Battery lifetime (years):	**1.28**

Time distribution: Advertising: 0% Connection: 100% Sleeping: 0%

The current consumption of the nRF51 chipset can be calculated with the values given in the SoftDevice and nRF51 specifications or with an excel sheet that is provided here:

https://devzone.nordicsemi.com/question/36110/excel-sheet-for-power-consumption

Appendix 2 Mail Correspondence with CSR

Hi Marius,

We checked this for you however please note this specification is not as yet open to the public. It is currently being reviewed by the Bluetooth SIG who will then as a group determine how we make it open/public to all. This could take 12-18 months.

Sorry that we cannot help further.

Best regards,

CSR Web Support

A user has submitted the following information on the CSR website:

Name:	Marius Heil
Email:	mh218@hdm-stuttgart.de
Company:	HdM Stuttgart
Job Title:	Student
Address:	Nobelstrasse 10
Country:	Germany
CSRProduct:	CSRmesh
Describe Your Application:	Hello, I was not able to respond to the mail that I received from support (DNS Error: Address resolution ofinternal.csr.com. failed: Domain name not found), so I'll state my case again with this webform.
Specific Support Request:	I want to write my master thesis about Bluetooth Low Energy mesh networks and I would like to evaluate and maybe build upon the CSR mesh. I gathered from the first webinar video that the protocol specification will be made public. Is it possible that I get access to this specification? Is this specification only available if I buy the development kit? My university already has access to a different kind of hardware and I don't know whether they could provide me with the development kit. I would be very glad if you could help me, Marius Heil

Appendix 3 Project File Locations

The data archive that has been submitted together with this thesis contains the source code and project files. Following is a short presentation of the projects, their location and how they can be used.

Simulation

The simulation has been written in ActionScript 3 and is found in the folders `FruityMesh\GraphSimulation` (For the deterministic algorithm simulation from section 6.4.1) and `FruityMesh\GraphSimulation2` (for the final FruityMesh algorithm). Both folders contain a GraphSimulation.html file that can be opened in any browser with an installed Flash Player 11 or above.

The usage of the simulation has already been explained in section 7.2.

Prototype

The prototype is located in the folder `FruityMesh\FruityMesh` and can be compiled either with KEIL or with an Eclipse GCC setup. Setting up this workflow is very complicated and I advise interested readers to download the KEIL-MDK-Lite from www.keil.com, install the nRF51 SDK from www.developer.nordicsemi.com and open the project file `FruityMesh.uvprojx` with the IDE. The deploy files in `FruityMesh\FruityMesh\deploy` can then be used to flash the SoftDevice and the compiled code.

Location of the source code and its structure has already ben explained in section 8.1.

Campaign Management Showcase Setup Instructions

The hardware requirements for this showcase are a set of at least 3 nRF51 Dongles equipped with the latest FruityMesh build, some USB chargers, a computer with a free USB port running Windows 7 or later and an iPhone 5 or newer with at least iOS7.

The Objective-C Xcode project **AppleCampaignViewer** can be found in `FruityMesh\AppleCampaignViewer`. It needs to be installed on the iPhone.

The following Software has to be installed on the computer:

- Google Chrome
- Nordic nRFgo Studio
- Node.js
- A recent build of MongoDB

All of the nRF51 dongles must be flashed with the latest FruityMesh build and one of these must then be plugged in the USB port of the computer. It should register as a virtual COM device in the device manager. This port must be entered at the beginning of the `server.js file` which is found in `FruityMesh\BananaSink`.

Afterwards, the MongoDB must be started, followed by the `server.js` file which has to be opened with Node.js. Node.js will then open port 3000 on the local computer. Google Chrome can be used to view the BananaSink frontend at the following URL: `http://localhost:3000/index.html`.

After all nRF51 dongles have been plugged into either a USB port or a charger, the page can be reloaded until all dongles have been found. By entering a text into the text field before pressing the "Kampagne ändern" button, it is possible to deploy a campaign. The iPhone app has to be opened with activated Bluetooth to view the campaign.